Large Network Representation Learning Project

Fredrik Diffner **Valter Lundegårdh** **Bing Bai** **Lukas Widén**

Abstract

In this report we attempted to replicate and reproduce parts of the "A Comparative Study for Unsupervised Network Representation Learning" paper. Five different graph embedding methods, DeepWalk, Node2Vec, NetMF, LINE-1 and GraphSAGE, were used and applied to several real-world data sets and then evaluated using link prediction and node classification. Although no exact numeric value was reproduced our results showed that some of the conclusions made in the comparative study still held true for the previously used as well as new data sets. Some claims were also found to be incorrect but these we reasoned could be due to differences in the experimental setups. We ended this report by suggesting improvements to the comparative study as well as reflecting on what we learned.

1 Introduction

In the "A Comparative Study of Unsupervised Network Representation Learning" paper, further referred to as "the comparative study", the authors studied 10 different unsupervised graph embedding algorithms' performance on 4 different downstream tasks. These tasks were link prediction (LP), node classification (NC), graph clustering and graph reconstruction which were performed on the embedded graphs. In essence, an embedded graph is the original graph's transformed nodes. In this study the nodes were converted into 128-dimensional vectors. The graphs came from 11 different data sets of large graphs where 5 were undirected and 6 directed.

For graphs with a high clustering coefficient, high transitivity and high reciprocity, e.g. BlogCatalog, that were evaluated using LP, biased walk based methods, like Node2Vec, showed advantages. However, for a graph like BlogCatalog, with a high clustering coefficient and low transitivity, NetMF outperformed DeepWalk, Node2Vec and LINE-1. GraphSAGE was also shown to outperform DeepWalk for both directed and undirected graphs.

Regarding NC the key findings were that random walked based methods were the most robust methods for the task. DeepWalk was e.g. either the best algorithm or competing for a spot in the top for all data sets. Furthermore, NetMF was close to DeepWalk in all cases but had the best Macro-F1 score for BlogCatalog. There were additional findings but these were the ones that were studied in our project. [1]

1.1 Our project

In our project we began with collecting the data sets used in the comparative study. Out of the 11 we successfully found 6 data sets, which were *BlogCatalog*, *Flickr*, *Youtube*, *Reddit*, *Epinion* and *PubMed* respectively. We also used new data sets called *Facebook*, *BitcoinAlpha* and *lastfm* to see if some of the findings from the report could be replicated. Some of their structural properties are presented in table (1).

We also tried to verify that the data sets we found were the same by checking some of their properties like amount of vertices, edges and labels. PubMed was similar in almost all regards its reciprocity was 0.05% instead of 0.07%. We decided to accept this as we could not find any better PubMed data set. We then implemented the *DeepWalk*, *Node2Vec*, *NetMF*, *LINE-1* and *GraphSAGE* algorithms and evaluated these algorithms, using LP and NC, on the data we collected.

2 Methods

2.1 DeepWalk

DeepWalk is a random walk based graph embedding approach from a 2014 paper by Perozzi et al. [2]. Given a graph $G = (V, E)$ the algorithm will sample an decided amount of uniform random walks from each node. Uniform, in this case, means that all the neighbouring nodes of the current node have the same probability of being the next node in the walk.

It then uses the SkipGram algorithm to create an embedding using hierarchical softmax to save some computation time and get an approximation of the optimal embedding. SkipGram does not take edge directionality or labels of the node's into account and does therefore only learn from the undirected topology of the graph [2].

2.2 Node2Vec

This embedding algorithm is similar to the DeepWalk algorithm but samples the random walk in a slightly different way [3]. While DeepWalk could be interpreted as using a *Depth First Search* (DFS) approach, Node2Vec uses a mix between DFS and *Breadth First Search* (BFS) approach.

Compared to DeepWalk, Node2Vec also uses the more efficient *Negative Sampling* [4] method instead of Hierarchical Softmax in the SkipGram algorithm.

2.3 NetMF

As shown in the paper by Qui et al. from 2018 the SkipGram algorithm with negative sampling for word embedding is an implicit factorization of a certain word-context matrix. Thus, network embedding models such as DeepWalk, Node2Vec and LINE are in theory all performing implicit matrix factorization which can be computed explicitly using closed-form expressions. [5]

NetMF is built upon the DeepWalk matrix as that one is more general than the LINE matrix and computationally more efficient than the Node2Vec matrix. After computing this matrix, making it sparse and consistent, singular value decomposition is used to extract a lower dimensional embedding. This makes up algorithm 3 in the NetMF paper [5].

However, for certain parameter settings the time to compute this embedding becomes too large and hence NetMF also offers another option, called algorithm 4, where an approximation of the DeepWalk matrix is computed [5].

In the comparative study, both algorithm 3 and 4 were used but only the best result was reported.

Table 1: Structural properties of the new graphs. The properties are from left to right the amount of vertices (|*V*|), the amount of edges (|*E*|), the reciprocity (*r*), the clustering coefficient (*clus*) and finally the transitivity (*T*).

Data set	|*V*|	|*E*|	*r*	*clus*	*T*
Facebook	4039	88324	n.a.	0.606	0.519
BitcoinAlpha	3873	24186	83.21%	0.1766	0.078
lastfm	7624	27806	n.a.	0.2194	0.1786

Finally, an important feature of NetMF is that it can only create an embedding for undirected graphs. [1]

2.4 LINE-1

The -1 after LINE (Large-scale Information Network Embedding) stands for LINE with first-order proximity. First-order proximity represents the local pairwise proximity between two vertices. To preserve the first-order proximity, the KL-divergence is minimized and we get an objective function that is optimized using negative sampling. Like NetMF, LINE-1 can only be applied to undirected graphs. [6]

2.5 GraphSAGE

GraphSAGE uses an aggregation function to represent a node as a combination of its own feature information and aggregated feature information from nearby nodes k hops/layers away. This is illustrated in figure 1 from the paper "Inductive Representation Learning on Large Graphs" [7]. The aggregation functions used in the comparative study were: Mean, MeanPool, MaxPool, LSTM and a graph convolution network (GCN) [1].

In summary, there are three highlights of the GraphSAGE method: (1) it can leverage node feature information; (2) it is an inductive method which can be applied to unseen nodes and the transformation parameter matrices that are learned from sub-graphs can be used for the other parts of the graph; (3) the parameters can be learnt by unsupervised training or supervised training when the downstream task is specified.

2.6 Link Prediction

To conduct LP the comparative study first removed half of the positive edges, meaning edges that actually exist, of a given graph. The remaining graph, which was ensured to be connected, was then used for training while the removed edges were placed in a test set. They then *balanced out* the test set with negative edges by adding edges to the test set that did not exist between the nodes in the original graph. [1]

To be able to test the algorithms' ability to predict edge directionality, the authors reversed some positive edges which replaced a fraction of the negative edges in the test set for directed graphs. So if a random positive edge (a, b) was present in G, they also investigated if the reversed edge (b, a) was present in G. If not, the edge (b, a) replaced a random (non reversed) negative edge in the test set. This was done for 0%, 50%, and 100% of the negative edges in the test set.

The algorithms were then applied to the training graph which produced a network embedding. The ROC-AUC (Area Under the Receiver Operating Characteristic Curve) score was used to evaluate the algorithms' predictions of the edges in the test set. As a measurement of a link existence probability between two nodes, the sigmoid function of the inner product of their embedded normalised vectors, was used. If this value was then greater than 0.5 then it was said that there should be an edge between them. [1]

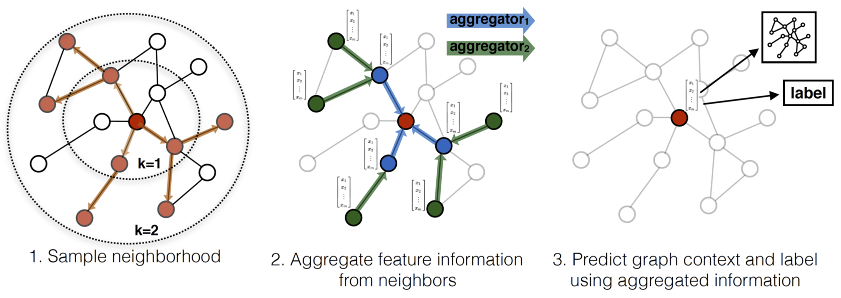
2.7 Node Classification

To evaluate the performance of the different algorithms on the NC task the authors of the comparative study split the work up into 3 parts. [1]

Firstly, after reading in the embedding of a graph, made by an algorithm, they split the vectors into 5 random and as equal-sized parts as possible. One at a time a part was used as test data while the rest was used to train a classifier.

Secondly, the classifier used was a multi-label classifier with one-vs-rest logistic regression. I.e., for each of the N different labels this classifier was used to find the best θ. More specifically, θ is the parameter vector which is optimised during the training of the classifier. The sigmoid of a node embedded vector dotted with θ then returns a value between 0 and 1 which is considered to be the probability of a node having one of the N labels.

Thirdly, to get a quantitative measure of how well the algorithms performed the comparative study used the micro and macro F1-score. This is basically two different ways of evaluating the accuracy of the algorithms using a classifier's precision and recall. The reported result was then the average of the micro and macro F1-score for the 5 separate rounds. [1]



aggregator1
aggregator2
label
k=1
k=2
1. Sample neighborhood
2. Aggregate feature information from neighbors
3. Predict graph context and label using aggregated information

Figure 1: Visual illustration of the GraphSAGE sample and aggregation approach [7].

3 Our implementation

In this section decisions regarding the methods we implemented are presented. Methods which implementations were clear-cut are not mentioned and can thus be assumed to have been implemented as they were described in section 2.

3.1 Overall restrictions

One restraint that haunted us throughout the project was that we, unlike the authors of a comparative study, did not have access to the same hardware i.e. an 80 core CPU with 1 TB RAM. The best free option we found was an 8 core CPU with 30 GB RAM from Google Cloud Platform. This meant that it might be harder, if not impossible, to acquire results from algorithms which required a lot of computing time and or memory, when working on the larger graphs like Epinion, Flickr, Reddit and Youtube. [1]

3.2 DeepWalk and Node2Vec

One common feature in DeepWalk and Node2Vec is the SkipGram algorithm, which is implemented in the Word2Vec method, found in the gensim library [8]. However, due to a recent update, which implemented sub-sampling and negative sampling, our results could differ from the original paper's outcome. These two additions to Word2Vec did not only improve the quality of produce embedded vectors but also decreased the time it takes for it to compute these [4]. We are also fairly certain that the comparative study did not use this new version but instead the 0.10.2 version of the gensim library, which contains the old Word2Vec function. We believe this because their results, on the BlogCatalog data set for NC, are almost the exact same as the score the DeepWalk paper got, which used the old version [1, 2, 8].

Thus, a perfect reproduction of the comparative study's results might require the old Word2Vec function. Sadly, with our hardware, the execution time of the old version is too slow. Hence, the newer version of gensim was used and kept in mind when investigating our results.

Furthermore, in the comparative study it is written that "...DeepWalk and Node2Vec are still ignoring edge directionality, even if they operate on directed random walks." [1]. Now this sounds as if DeepWalk and Node2Vec can operate on directed graphs. However, this is in contradiction to an issue raised on the DeepWalk's Github page stating that DeepWalk only works for unweighted and undirected graphs [9]. The answer written there was that to make it applicable to directed graphs one would need to change the Word2Vec code so that it only considers future words and not previous words in a sentence.

Yet, if one investigates the DeepWalk Github page further it is clear that if the algorithm reaches a leaf node it simply breaks the path before it has not reached its max length. Leaf nodes are otherwise one legitimate reason for why DeepWalk cannot handle directed graphs but breaking the path, as is done now, intuitively solves this problem [10]. The GitHub code for DeepWalk also seems to have other recently added code segments which makes it able to handle directed graphs which is in contradiction to the raised issue. This is however hard to ensure as their paper never used DeepWalk for directed graphs. [2]

To conclude, it is difficult to reach a conclusion of how our paper handled directed graphs regarding NC compared to LP where there is a clear separation of the data sets. Because of this we decided to report the node NC results for PubMed, the only directed data set used in NC, when it was directed and converted to an undirected graph. What is noteworthy is that we did not attempt to edit the Word2Vec library as our paper never mentioned it explicitly [1].

For the new data sets we also decided to set p to 0.25 and q to 4 as these were the values used for the majority of the data sets in the comparative study [1].

3.3 LINE-1

In order to have time to run LINE-1, we had to limit the amount of samples that were used for training. We only had time to run 2% of the 10 billion samples that they used, i.e. 200 million samples. This took approximately 60 hours, so we did not have time to run LINE-1 for all graphs. LINE-1 was only trained with 20 million samples for PubMed since the graph was small and did not need as many samples, in addition to time constraints. Moreover, as we did not understand how the comparative study was able to produce results for LINE-1 for directed graphs in LP, as it is stated in the LINE paper that the algorithm can only be used for undirected graphs, we decided to not run LINE-1 for directed graphs [6, 11].

3.4 GraphSAGE

In line with the comparative study [1] we implemented an unsupervised and transductive version of GraphSAGE. The aggregation functions used in this project were: Mean, MeanPool, MaxPool, LSTM. Like the comparative study we also only present the result from the best aggregator.

GraphSAGE aggregate node features, and when no features are present in the data set the authors of GraphSAGE [7] give several suggestions of what to use as features: node degrees, node ID, node label etc. However, it was hard to know which kind of input features the comparative study [1] used when the data set did not contain any node features. For this study, we always used a random initialization of the node embedding as features to maintain consistency and comparability with other methods.

The authors of GraphSAGE also downsampled the node degrees of the input graph. For simplicity this was omitted in this study but this is not expected to change the results significantly, as shown in figure 2B in the paper "Inductive Representation Learning on Large Graphs" [7]. Finally, the positive edges used in the loss function were generated from 50 random walks of length 5, as recommended by the authors of GraphSAGE [7]. This generated a large amount of edges, which became a problem for the LSTM aggregator on the BlogCatalog data set due to a lack of memory. When this happened we limited the amount of negative edges and positive edges used by the loss function. However, we do not expect that this has affected our results as the tuning of these parameters are not believed to contradict any of the findings in the comparative study [1].

3.5 Link Prediction

When implementing LP the only question mark we had was what the authors meant by "The test split is balanced with negative edges..." [1]. We thought this could mean one out of two things. (1) They added negative edges to the test set until they had as many negative edges as positive, thus making it twice as large as the training set. (2) They added negative edges while removing positive edges from the test set until the test set contained 50% positive and 50% negative edges. We decided to test both cases. The results, found in table (2) and (3), were however so similar for both cases, i.e. within a $\pm$ 3% interval, that we decided to only report the scores for case (1).

Thus, when performing LP on a graph G, we created equally sized training and test sets by randomly transferring 50% of the edges in G to the test set. The remaining edges then became the training set. While doing this we ensured that no node in the training set became isolated as that would make certain algorithms, such as DeepWalk, fail. The test set was then balanced with random negative edges according to either case (1) or (2). The rest of the LP was implemented as described in section 2.6.

3.6 Node Classification

The one uncertainty we had, regarding NC, was which one-vs-rest logistic classifier the comparative study used. Hence, we decided to use one already implemented by Scikit-learn [12]. Moreover, we implemented the multi-label part partly by ourselves and partly by taking inspiration from the *TopKRanker* class from the DeepWalk paper as we assumed that the comparative study once again might have taken inspiration from them, just as with the old Word2Vec library we mentioned in section 3.2 [11]. However, when analysing their NC algorithm we found it a bit counter-intuitive as it assumed it knew how many labels, k, each node had. It then simply assigned an unlabelled node in the test set to the k most likely labels, even if the probability was low. This we thought could be a data leak but we nonetheless implemented it like this to reproduce and replicate the comparative study, but we also discussed it further in section 5.5.4 [13].

4 Results

In table (2) the LP results for undirected graphs are presented, in table (3) the LP results for the directed graphs and in table (4) the NC score. An "mem." indicates that the method was unable to finish computations on the specified data set due to a lack of available memory. A "-" indicates that no attempt was made to calculate that score due to it not being possible. Finally a "T" indicates that the training took too long and we were unable to acquire it before the hand-in date with the available hardware. Due to our hardware restrictions we were also not able to run any algorithm on the Reddit and Youtube data sets due to their size. Therefore these were excluded from our results.

5 Discussion

5.1 Replication and reproduction of results

To begin with it is worth noting that not a single of our LP or NC scores are the same as the corresponding score in the comparative study, even if we in some cases are extremely close, so in some sense we were unable to reproduce their results. Even so, our findings do still justify and or contradict some of the conclusions the comparative study made.

Regarding the conclusion that Node2Vec should be advantageous for graphs with a high clustering coefficient, transitivity and high reciprocity we can see that this is indeed not the case for the Facebook graph in table 2. In the same table it can be seen that NetMF does in fact not outperform DeepWalk but only Node2Vec and LINE-1 for BlogCatalog. We can also see from table (2) and (3) that for LP GraphSAGE is not better than DeepWalk for undirected graphs, although the scores being very close for BlogCatalog. For directed LP however GraphSAGE is marginally better than DeepWalk which is in agreement with the authors' conclusion. These results might however be due to the updated Word2Vec method, which we reflect more upon in section 5.4.

For NC, see table (4), it is clear that DeepWalk is still a robust method for all the old data sets as well as the new lastfm data set. This makes us think that the authors' statement that DeepWalk is a robust method for NC is correct. NetMF was as also able to get a better macro F1-score for BlogCatalog.

Furthermore, LINE-1 getting a worse score, for most data sets, might be due to it being trained using, as mentioned in section 3.3, only a fraction of all of the samples. Having started on this path let us continue with discussing things that affected or could have affected our results and some potential solutions to these.

5.2 Out of memory

A troublesome fact that makes it hard to reproduce the results from the original paper was that we ran into memory issues when using the algorithms on larger data sets. The easy solution to this would of course be to get more RAM. If one wants to get more technical however another solution might be possible.

For DeepWalk and Node2Vec this was simply a consequence of us trying to save all the random walks in one array. To avoid this we considered writing each individual walk to a file when it was completed. This method was however not implemented due to lack of time, and other memory related problems we ran into for the bigger data sets. GraphSAGE does also make use of several random walks to generate positive edges for its loss function, which creates the same problem as for DeepWalk and Node2Vec. In the NetMF case we are not aware of any potential workarounds regarding this issue as an adjacency matrix has to be computed which has the size $V \times V$.

Node2Vec also computes a $V \times V$ matrix but this for storing the probabilities of visiting the neighbours of each node. This is however a bit wasteful as all of the probabilities are not needed at all times. Especially, as a massive amount of RAM is required for larger data sets, e.g. Flickr which requires 48.3 GiB for this matrix, one potential solution for the Node2Vec case could be to calculate the probabilities as the random walk is happening instead of before. This could be implemented with an array containing only some of the probabilities which swaps out elements as they become irrelevant, sort of like a RAM within the program. The issue here however is that it would increase the amount of calculations as recalculations of probabilities would be required, thus increasing the computation time even further which we could not afford as we were already in a time crunch. Hence, this is left as future work for anyone who wishes to create network embeddings of large graphs using Node2Vec and basic hardware. Yet, one would still need to solve the issue of storing the random walks as described earlier.

5.3 PubMed data set

From table (4) there is one thing that really stands out which is that we get a roughly 10% increase in the NC accuracy for DeepWalk, Node2Vec, NetMF and LINE-1 on PubMed, when it is converted to an undirected graph, while we on the other data sets get pretty much the same score. Initially we thought this might be because of the conversion of the directed PubMed graph to an undirected graph. However, as can also be seen, the directed graph returned around a 10% worst score.

Thus if we were to make an educated guess we would say that the comparative study converted PubMed into an undirected graph. We believe this since the scores by DeepWalk, Node2Vec and NetMF are then all roughly the same which is inline with the original paper, even if they are all 10% larger [1]. This argument is further backed

Table 2: Link prediction results on undirected graphs using the ROC-AUC score.

method	BlogCat.	Flickr	Facebook
DeepWalk	0.728	0.933	0.991
Node2Vec	0.656	mem.	0.985
NetMF	0.691	mem.	0.989
LINE-1	0.691	T	T
GraphSAGE	0.721	mem.	0.662

Table 3: Link prediction results on directed graphs using the ROC-AUC score.

	Epinion			BitcoinAlpha		
method	0%	50%	100%	0%	50%	100%
DeepWalk	0.865	0.712	0.671	0.853	0.720	0.721
Node2Vec	mem.	mem.	mem.	0.784	0.671	0.673
GraphSAGE	mem.	mem.	mem.	0.859	0.739	0.7438

up by the fact that NetMF always operates on undirected graphs and thus the score from that can be seen as a benchmark, especially as it gets approximately the same score on the BlogCatalog data set. Thus, the reason for why algorithms get a better score on PubMed could be due to the fact that, as mentioned in section 1.1, PubMed's reciprocity is different. It is also possible that the graph should not be converted but instead a change to Word2Vec, as described in section 3.2 is required to get the same score.

GraphSAGE does not follow this pattern. Comparing our results to the comparative study, GraphSAGE's score is significantly worse for the task of NC on PubMed [1]. This is more consistent with GraphSAGE's NC performance on other data sets, both in the comparative study and in our study. We suspect this is a consequence of different input features. When we tested to run GraphSAGE on PubMed with the node labels as input features, we got a micro and macro score of 83,81 and 82,29 for the mean aggregator, which seems to be more in line with the comparative study. However, using the labels as features for training is simply another data leak and thus something we decided to not pursue any further [13].

5.4 Word2Vec version affecting results

As mentioned in section 3.2 a newer version of the Word2Vec method was used in this project which should increase performance and speed [4]. However, from our results it would seem as if this version did not change the scores for NC, assuming that the reason for PubMed's scores being different is one of the ones described in section 5.3. Yet, for LP there is a clear improvement to the DeepWalk and Node2Vec score for all data sets that were evaluated. This makes us believe that perhaps the updated Word2Vec version only improves the LP score.

To verify this we ran a small test where we ran the original DeepWalk code and and our DeepWalk code, both with the updated Word2Vec method, 10 times each on the BlogCatalog data set and looked at the average LP score. The parameters were not set to the proper values as to make it run faster. The result was that their code got a score of 0.704 our code got 0.7047. This we thought was a clear indication that our two implementations are almost the exact same and hence the only difference, other than hardware, is the version of the Word2Vec method. Hence, this is likely what lies behind the different results for the LP task for DeepWalk and Node2Vec. This is also verified by NetMF, which does not use Word2Vec, returning almost the same score as the comparative study for the LP task.

5.5 Evaluation of the comparative study

5.5.1 Their approach

We think that the general goal and aim of the comparative study was splendid. As the number of graph embedding algorithms are increasing, evaluations of them, and a common evaluation metric are vital for practitioners and researchers to be able to select the right algorithm for their work. Although their results are wonderful, there are some things in their report we believe could be improved.

5.5.2 Data sets

To begin with it is rather hard to find the data sets they have used. For the report to be reproducible the data sets should be provided to the reader so that they themselves do not have to scour the internet.

5.5.3 Reversing edges

When it comes to LP we believe their implementation can be improved to display a more accurate score. If what they wrote is true the reversed

Table 4: Node classification results given in Micro and Macro F1 score for the implemented methods.

	BlogCatalog		PubMed (undir.)		PubMed (dir.)		Flickr		lastfm	
method	micro	macro	micro	macro	micro	macro	micro	macro	micro	macro
DeepWalk	41.79	27.81	80.99	78.68	60.67	53.59	40.74	27.85	87.32	79.75
Node2Vec	41.36	27.09	80.95	78.70	56.58	46.88	mem.	mem.	86.31	78.41
NetMF	43.12	28.03	81.12	79.06	-	-	mem.	mem.	86.92	77.60
LINE-1	37.42	22.08	65.42	61.74	-	-	T	T	T	T
GraphSAGE	23.24	6.34	46.90	38.96	46.85	38.31	mem.	mem.	54.92	38.34

edges are reversed positive edges in the test set. This can potentially be harmful as reversing a positive edge in the test set, while not removing the same edge from the test set, which is not mentioned, can impact the score massively. As two embedded nodes dotted with each other will always return the same value it means that the prediction of a positive edge and its reversed counterpart will be the same. Thus for the case when all the negative edges are reversed positive edges, assuming that all the positive edges can be reversed, the accuracy should be exactly 50%. This is because we will either correctly predict a positive edge to exist in the graph while incorrectly predicting that the reversed edge exists in the graph or vice versa. One way to solve this could be to reverse edges from the training set and not the test set. [1]

5.5.4 Multi-label classification

Furthermore, we found their implementation of NC, more specifically how multi-label classification was done, odd. As mentioned in section 3.6 their way of assigning a node to labels could return a misleading score and is a classical case of data leakage [13]. When someone desires to know which labels a new node should be given, where no knowledge about how many labels it should be assigned, the algorithm will also fail. We thought a better and more accurate way to conduct NC could be to only assign a node to a label if the probability of it belonging to that label is greater than or equal to 0.5. If all of the probabilities of a node belonging to any label is lower than 0.5 the most likely single label can be picked so that all nodes have at least one label.

5.6 What we learned

The most important things we learned during this project was that it is, to begin with, very hard to reproduce a study in a short amount of time. It is especially challenging to understand all the specifics of each algorithm when not every nitty-gritty detail, like what *balanced out* regarding LP means, is described.

Yet, after all of this we believe that our capability of taking on new challenging tasks has increased as we now know that even daunting tasks can be achieved if one perseveres through the hardship.

References

[1] Megha Khosla, Avishek Anand, and Vinay Setty. "A Comprehensive Comparison of Unsupervised Network Representation Learning Methods". In: *CoRR* abs/1903.07902 (2019). arXiv: 1903.07902. URL: http://arxiv.org/abs/1903.07902.

[2] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: KDD '14 (2014), pp. 701–710. DOI: 10.1145/2623330.2623732. URL: http://doi.acm.org/10.1145/2623330.2623732.

[3] Aditya Grover and Jure Leskovec. "Node2vec: Scalable Feature Learning for Networks". In: *KDD '16 (2016)*. ACM, p. 855864. ISBN: 9781450342322. DOI: 10.1145/2939672.2939754. URL: https://doi.org/10.1145/2939672.2939754.

[4] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and their Compositionality". In: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[5] Jiezhong Qiu et al. "Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec". In: ACM, 2018, pp. 459–467. URL: https://arxiv.org/abs/1710.02971.

[6] Jian Tang et al. "LINE: Large-Scale Information Network Embedding". In: *WWW '15 (2015)*, p. 10671077. ISBN: 9781450334693. DOI: 10.1145/2736277.2741093. URL: https://doi.org/10.1145/2736277.2741093.

[7] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216. URL: http://arxiv.org/abs/1706.02216.

[8] *Gensim*. Accessed: 2020-12-29. 2009. URL: https://radimrehurek.com/gensim/auto_examples/core/run_core_concepts.html.

[9] *Is the code just only applicable to undirected and unweighted graphs? 52*. Accessed: 2020-01-06. 2017. URL: https://github.com/phanein/deepwalk/issues/52.

[10] *Igraph.py*. Accessed: 2020-01-06. 2014. URL: https://github.com/phanein/deepwalk/blob/master/deepwalk/graph.py.

[11] *scoring.py: Script that demonstrates the multi-label classification used*. Accessed: 2020-12-30. 2017. URL: https://github.com/phanein/deepwalk/blob/master/example_graphs/scoring.py.

[12] *Linear models*. Accessed: 2020-01-06. 2020. URL: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression.

[13] *How to Avoid Data Leakage When Performing Data Preparation*. Accessed: 2020-01-11. 2020. URL: https://machinelearningmastery.com/data-preparation-without-data-leakage/.