# Report: Node Embeddings

**Karuna Bhaila** **Mamata Shrestha**

## 1 Problem Statement

The task we address for our project is node classification. In a traditional machine learning task, data is usually available in the form of records where each row represents an example. In a real world setting, such records may also be accompanied by a network that connects examples with other examples. This network structure may have information that is not represented by attributes of the examples and can be used to enhance classification tasks.

In a node classification problem, the goal is to infer a fully labeled graph from a partially labeled one by leveraging node relationships and network properties. Network properties commonly observed in real-world networks include homophily and structural equivalence. Homophily refers to the principle that nodes tend to be connected to other nodes that have similar features whereas structural equivalence refers to the similarity between two nodes' structural roles in the network and their connections to other nodes in their neighborhood. It is likely that in a network exhibiting homophily, connected nodes belong to the same class. Similarly, structurally equivalent nodes may have the same class labels.

### 1.1 Approach

Network data is usually represented in the form of edges. Additional edge attributes such as weight, edge type, etc. may also be available sometimes. However, this type of information cannot be fed directly into a classification model as input. Classifiers require rows of features as input in order to calculate class probabilities. Therefore, the information from a network must be converted into a node feature format acceptable by a classifier.

This can be obtained using feature learning on the network structure as proposed in [1]. For our project, we will follow this workflow to first learn representations for nodes in a network such that they incorporate nodes' relationships with their neighborhood and the network structure. These representations are obtained by mapping the network to a low dimensional embedding space. Then we will use the representations as inputs for a downstream classification task.

## 2 node2vec

We use the node2vec [1] model to generate embeddings for each node in the network. Node2vec is a semi-supervised feature learning algorithm. It generates embeddings based on a node's neighborhood and the network structure by extending Word2Vec's [2] skipgram architecture to a networked setting.

The skipgram model is a shallow 2-layer neural network. It takes context neighborhood of a word in a text corpus as input which is then split into pairs of word and context word. The neural network takes these pairs as the training examples and predicts the context words for each input. Once the model is trained, the hidden layer weights are retrieved as the feature representations for each word. The authors formulate this for a network by maximizing the log probability of each node's neighborhood conditioned on its feature representations [1].

For a graph G=(V,E), let $f : V \rightarrow \mathbb{R}^d$ be a mapping function that maps each node $u$ to its representation $f(u)$. For every node u, the authors define a neighborhood $N_s(u)$ generated using sampling

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

strategy S. The objective function is then given as:

$$\max_{f} = \sum_{u \epsilon v} \log \Pr(N_s(u)|f(u)) \tag{1}$$

However, nodes in a network do not have a linear context neighborhood as in a text corpus. So, the authors in [1] implement a sampling strategy that captures both node neighborhood and network structure in terms of homophily and structural equivalence. They achieve this by implementing a second-order biased random walk which generates $n$ number of random walks of fixed length $l$ for all nodes in the network.For source node $c_0 = u$, the $i^{th}$ node, $c_i$ is obtained from the distribution:

$$P(c_i = x|c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if } (v,x)\epsilon E \\ 0 & \text{otherwise} \end{cases}$$

Here, $\pi_{vx}$ is the transition probability of the edge connecting node $v$ to $x$. This probability is calculated based on two parameters $p$ and $q$. The return parameter $p$ controls the likelihood that the random walk will backtrack to the previous node. Having a low $p$ value encourages backtracking and keeps the exploration local. Setting it to a higher value reduces the chances of re-sampling the same node. The in-out parameter $q$ dictates how far the walk can move from the source node. Having a higher $q$ value encourages BFS like exploration and keeps the neighborhood local. This helps to accurately portray the local neighborhood and determine a node's structural role in the network, if any. On the other hand, a lower value inclines the random walk towards a DFS-like exploration which allows the embedding to capture the homophilic tendencies of the network.

Thus, the node2vec algorithm outputs an embedding space where nodes that are homophilous or structurally equivalent are embedded close together depending on the parameters.

## 3 Experiment

### 3.1 Dataset

We used the Cora [3] dataset which is a publicly available citation network consisting of 2708 nodes and 5429 edges. Nodes represent scientific publications and edges represent a citation link between two papers. Each paper has been categorized into one of the following 7 classes (Neural networks, Probabilistic Methods, Genetic Algorithms, Theory, Case Based, Reinforcement Learning, Rule Learning) which we use for a multi-class classification problem. The dataset also comes with features representing 0/1 valued vectors indicating the absence or presence of a dictionary word. However, we do not use this information as the node2vec model does not take any features as input.

### 3.2 Setup

For our project, we will start by doing an initial run of node2vec algorithm and evaluate the performance of different standard classifier models which include Logistic Regression, Support Vector Machine, KNearestNeighbor. Then, we will pick one classifier model to optimize the hyperparameters for the node2vec model. We will compare performances for low and high value of the walk length parameter. We expect that the cora network would exhibit a high homophilic tendency and almost no structural equivalence. Keeping this in mind, we will try to manipulate $p$ and $q$ values so that the embeddings reflect this information. Finally we will optimize a classification model with the best embeddings that we can achieve using node2vec.

### 3.3 Results

The results included in the presentation were generated with a random seed. We were not able to reproduce the exact results when we ran the model with the same parameters again. Therefore, the results discussed in the report will be different from the ones shown in the presentation. All of the results discussed here have been generated with random seed = 0 for comparison and reproducibility.

Figure 1 shows the performance of five different classification models trained and tested on a 80-20 split of the embeddings generated by running node2vec with its default parameters i.e.: dimensions=128, walk_length=80, num_walks=10, $p$=1, $q$=1.

| | classifier | accuracy | f1_micro | f1_macro |
|---|---|---|---|---|
| 0 | KNN | 0.800738 | 0.800738 | 0.795095 |
| 1 | LR | 0.804428 | 0.804428 | 0.792206 |
| 2 | SVM | 0.806273 | 0.806273 | 0.799754 |
| 3 | NB | 0.743542 | 0.743542 | 0.744498 |
| 4 | DT | 0.647601 | 0.647601 | 0.619054 |

Figure 1: Classifier Scores

| | dimensions | walk_length | num_walks | p | q | accuracy | time |
|---|---|---|---|---|---|---|---|
| 0 | 128 | 80 | 50 | 0.250000 | 0.250000 | 0.841587 | 365.462466 |
| 1 | 128 | 80 | 50 | 0.250000 | 0.500000 | 0.837526 | 354.827043 |
| 2 | 128 | 80 | 50 | 0.250000 | 1.000000 | 0.831251 | 338.969633 |
| 3 | 128 | 80 | 50 | 0.250000 | 2.000000 | 0.833096 | 320.228759 |
| 4 | 128 | 80 | 50 | 0.250000 | 4.000000 | 0.827187 | 304.844336 |
| 5 | 128 | 80 | 50 | 0.500000 | 0.250000 | 0.843064 | 374.866015 |
| 6 | 128 | 80 | 50 | 0.500000 | 0.500000 | 0.839743 | 369.816785 |
| 7 | 128 | 80 | 50 | 0.500000 | 1.000000 | 0.837528 | 358.433106 |
| 8 | 128 | 80 | 50 | 0.500000 | 2.000000 | 0.831249 | 342.237712 |
| 9 | 128 | 80 | 50 | 0.500000 | 4.000000 | 0.832359 | 322.873894 |
| 10 | 128 | 80 | 50 | 1.000000 | 0.250000 | 0.841587 | 378.746608 |
| 11 | 128 | 80 | 50 | 1.000000 | 0.500000 | 0.842695 | 374.781177 |
| 12 | 128 | 80 | 50 | 1.000000 | 1.000000 | 0.846020 | 368.150597 |
| 13 | 128 | 80 | 50 | 1.000000 | 2.000000 | 0.836789 | 355.922886 |
| 14 | 128 | 80 | 50 | 1.000000 | 4.000000 | 0.832357 | 337.858209 |
| 15 | 128 | 80 | 50 | 2.000000 | 0.250000 | 0.843801 | 377.803664 |
| 16 | 128 | 80 | 50 | 2.000000 | 0.500000 | 0.844172 | 374.187151 |
| 17 | 128 | 80 | 50 | 2.000000 | 1.000000 | 0.847495 | 370.470108 |
| 18 | 128 | 80 | 50 | 2.000000 | 2.000000 | 0.839742 | 361.168774 |
| 19 | 128 | 80 | 50 | 2.000000 | 4.000000 | 0.839741 | 346.972243 |
| 20 | 128 | 80 | 50 | 4.000000 | 0.250000 | 0.838631 | 381.117395 |
| 21 | 128 | 80 | 50 | 4.000000 | 0.500000 | 0.841586 | 380.112637 |
| 22 | 128 | 80 | 50 | 4.000000 | 1.000000 | 0.842695 | 376.058452 |
| 23 | 128 | 80 | 50 | 4.000000 | 2.000000 | 0.840848 | 369.618545 |
| 24 | 128 | 80 | 50 | 4.000000 | 4.000000 | 0.843064 | 358.335377 |

(a) $d = 128$, num_walk=80, walk_length=50

| | dimensions | walk_length | num_walks | p | q | accuracy | time |
|---|---|---|---|---|---|---|---|
| 0 | 128 | 20 | 100 | 0.250000 | 0.250000 | 84.600000 | 182.880000 |
| 1 | 128 | 20 | 100 | 0.250000 | 0.500000 | 83.940000 | 178.270000 |
| 2 | 128 | 20 | 100 | 0.250000 | 1.000000 | 83.310000 | 171.440000 |
| 3 | 128 | 20 | 100 | 0.250000 | 2.000000 | 84.310000 | 164.920000 |
| 4 | 128 | 20 | 100 | 0.250000 | 4.000000 | 83.270000 | 162.170000 |
| 5 | 128 | 20 | 100 | 0.500000 | 0.250000 | 84.080000 | 195.680000 |
| 6 | 128 | 20 | 100 | 0.500000 | 0.500000 | 84.120000 | 182.950000 |
| 7 | 128 | 20 | 100 | 0.500000 | 1.000000 | 84.160000 | 177.170000 |
| 8 | 128 | 20 | 100 | 0.500000 | 2.000000 | 84.230000 | 171.840000 |
| 9 | 128 | 20 | 100 | 0.500000 | 4.000000 | 83.720000 | 162.970000 |
| 10 | 128 | 20 | 100 | 1.000000 | 0.250000 | 84.420000 | 187.130000 |
| 11 | 128 | 20 | 100 | 1.000000 | 0.500000 | 84.230000 | 184.350000 |
| 12 | 128 | 20 | 100 | 1.000000 | 1.000000 | 84.570000 | 181.250000 |
| 13 | 128 | 20 | 100 | 1.000000 | 2.000000 | 84.340000 | 175.870000 |
| 14 | 128 | 20 | 100 | 1.000000 | 4.000000 | 84.010000 | 168.510000 |
| 15 | 128 | 20 | 100 | 2.000000 | 0.250000 | 84.340000 | 186.680000 |
| 16 | 128 | 20 | 100 | 2.000000 | 0.500000 | 84.340000 | 188.250000 |
| 17 | 128 | 20 | 100 | 2.000000 | 1.000000 | 84.570000 | 184.490000 |
| 18 | 128 | 20 | 100 | 2.000000 | 2.000000 | 84.340000 | 180.720000 |
| 19 | 128 | 20 | 100 | 2.000000 | 4.000000 | 84.230000 | 174.990000 |
| 20 | 128 | 20 | 100 | 4.000000 | 0.250000 | 84.820000 | 188.360000 |
| 21 | 128 | 20 | 100 | 4.000000 | 0.500000 | 84.310000 | 187.510000 |
| 22 | 128 | 20 | 100 | 4.000000 | 1.000000 | 84.790000 | 185.910000 |
| 23 | 128 | 20 | 100 | 4.000000 | 2.000000 | 84.080000 | 182.960000 |
| 24 | 128 | 20 | 100 | 4.000000 | 4.000000 | 84.420000 | 178.800000 |

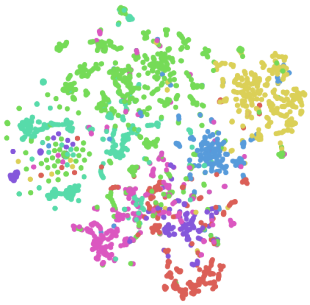
(b) $d = 128$, num_walk=20, walk_length=100

Figure 2: Grid Search for node2vec parameters $p, q$

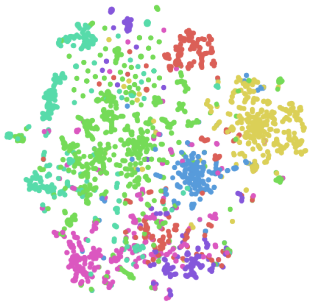
The classification models were defined using packages from scikit-learn[4]. We decided to use the default parameters for these models as well. For KNN, we used 4 as the $k$ value, as that was the average node degree of the network. We received similar accuracy and macro scores for KNN, Logistic Regression, and SVM with a linear kernel. We decided to conduct further testing using KNN as we expect the Cora network to be homophilous thus resulting in embeddings where nodes with the same labels are closer in the embedding space.

Figure 2 shows the results of two grid searches for node2vec parameters mainly $p$ and $q$. We performed the first grid search with a higher value of walk_length at 80 and the second one with a lower value at 20. The accuracy scores between the two searches are not significantly different but the model runs faster with a lower value of walk_length. It seems that having a higher number and length of walks per node does not improve performance for this dataset.

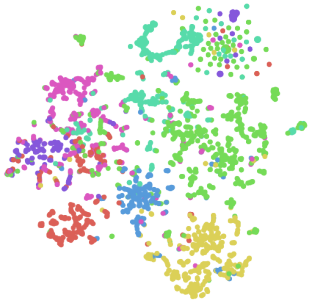
We also observed that the trend of the accuracy score follows a similar pattern in both searches based on the values of $p$ and $q$. The higher accuracy scores shown in darker colors in the tables are mostly achieved at $p \geq 1$ and $q \leq 1$. In these cases, the algorithm encourages outward exploration with a DFS-like behaviour. This type of exploration is preferable for capturing homophily in a network. Also, a low value of $q$ does not capture the structural equivalence in the network. However, we do not



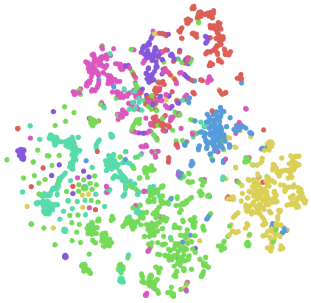
(a) $p = 1$ and $q = 1$



(b) $p = 1$ and $q = 4$



(c) $p = 4$ and $q = 1$



(d) $p = 4$ and $q = 4$

Figure 3: TSNE plots with parameters dimensions=128, walk_length=20, num_walks=100

expect structurally equivalent nodes in cora to have the same label so, the embeddings do not need to represent this information.

Figure 3 shows TSNE plots for four different embeddings obtained with varying $p$ and $q$ parameters. The colors in the plot represent the true class labels of the nodes. The plots on the left with a lower $q$ value have a clearer distinction between the clusters formed on the embedding space and are more representative of the homophily between papers in the cora network.

Then we performed a grid search using embeddings generated with node2vec parameters: dimesions=128, walk_length=20, num_walks=100, $p$=4, and $q$=1 for a KNN classfier and an SVM classifier. The results for the grid search are shown in figure 4. The best performance is achieved by SVM with an rbf kernel and penalty value 5 for an accuracy of 86%.

## 4 Groupwork

For this project we didn't really divide the workload distinctly between the two of us but rather brainstormed the idea and wrote the code together. We started by reading the node2vec paper and then looked for a suitably sized dataset. We then experimented with the model individually and discussed its performance and how to go about improving it. For parameter tuning, we explained our reasoning to each other and tried to come up with an explanation that best fits our observation. For the presentation and report, we worked on sections and then reviewed each other's work.

| | algorithm | n_neighbors | Accuracy |
|---|---|---|---|
| 0 | ball_tree | 4 | 0.822755 |
| 1 | ball_tree | 8 | 0.814998 |
| 2 | ball_tree | 10 | 0.810934 |
| 3 | kd_tree | 4 | 0.822755 |
| 4 | kd_tree | 8 | 0.814998 |
| 5 | kd_tree | 10 | 0.810934 |
| 6 | brute | 4 | 0.822755 |
| 7 | brute | 8 | 0.814998 |
| 8 | brute | 10 | 0.810934 |

(a) $KNN$

| | C | kernel | Accuracy |
|---|---|---|---|
| 0 | 0.500000 | linear | 0.839005 |
| 1 | 0.500000 | rbf | 0.828283 |
| 2 | 1.000000 | linear | 0.838636 |
| 3 | 1.000000 | rbf | 0.848229 |
| 4 | 5.000000 | linear | 0.815001 |
| 5 | 5.000000 | rbf | 0.860050 |
| 6 | 10.000000 | linear | 0.810563 |
| 7 | 10.000000 | rbf | 0.854510 |

(b) $SVM$

Figure 4: Grid Search

## References


[1] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In KDD, 2016.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In ICLR, 2013.

[3] Cora. *LINQS*, https://linqs.soe.ucsc.edu/data.

[4] Pedregosa, Fabian et al. Scikit-learn: Machine Learning in Python. 2012.