Supplementary: On the Use of Unrealistic Predictions in Hundreds of Papers Evaluating Graph Representations

Li-Chung Lin[1], Cheng-Hung Liu[1], Chih-Ming Chen[2], Kai-Chin Hsu[3],
I-Feng Wu[4], Ming-Feng Tsai[2], and Chih-Jen Lin[1]

[1]National Taiwan University
[2]National Chengchi University
[3]University of Southern California
[4]ASUS Intelligent Cloud Services
r08922141@ntu.edu.tw, ericliu8168@gmail.com, 104761501@nccu.edu.tw, kaichinh@usc.edu,
ifengwu1518@gmail.com, mftsai@nccu.edu.tw, cjlin@csie.ntu.edu.tw

1 Proofs of the Main Paper

In this section we show the proofs of our main paper. To facilatate the discussion, we consider

$$i : \text{index of test instances, and } j : \text{index of labels.}$$

We further assume that for test instance i

$$\begin{aligned} &K_i : \text{true number of labels,} \\ &\hat{K}_i : \text{predicted number of labels.} \end{aligned} \tag{10}$$

Define

$$\text{Micro-F1} = \frac{2 \times \text{TP sum}}{\text{TP sum + FP sum + TP sum + FN sum}}, \tag{11}$$

where "sum" means the accumulation of prediction results over all binary problems. Next we prove an upper bound of Micro-F1.

1.1 Proof of Theorem 1

Proof. Let TP_i, FP_i, and FN_i be respectively true positive, false positive and false negative from the predictions of instance i. The definition of K_i and $\hat{K}_i$ in (10) implies

$$\text{TP}_i + \text{FP}_i = \hat{K}_i \text{ and } \text{TP}_i + \text{FN}_i = K_i. \tag{12}$$

From (11) and (12),

$$\text{Micro-F1} = \frac{2 \times \text{TP sum}}{\sum_{i=1}^{l}\big(K_i + \hat{K}_i\big)} = \frac{2 \times \sum_{i=1}^{l} \text{TP}_i}{\sum_{i=1}^{l}\big(K_i + \hat{K}_i\big)}. \tag{13}$$

From (12), $\text{TP}_i \le \hat{K}_i$ and $\text{TP}_i \le K_i$, so the result in (7) follows from

$$\text{TP}_i \le \min\big(\hat{K}_i, K_i\big). \tag{14}$$

The second inequality in (7) easily follows from $\min\big(\hat{K}_i, K_i\big) \le \hat{K}_i$ and K_i. Finally, when $\hat{K}_i = K_i$,

$$\frac{2 \times \sum_{i=1}^{l} \min\big(\hat{K}_i, K_i\big)}{\sum_{i=1}^{l}\big(K_i + \hat{K}_i\big)} = \frac{2 \times \sum_{i=1}^{l} K_i}{\sum_{i=1}^{l}\big(K_i + K_i\big)} = 1 \tag{15}$$

achieves the maximum. □

1.2 Proof of Theorem 2

Proof. By the assumption on decision values, selecting the top ones leads to

$$\text{TP}_i = \begin{cases} \hat{K}_i & \text{if } \hat{K}_i \le K_i, \\ K_i & \text{otherwise.} \end{cases}$$

Thus the upper bound of TP_i in (14) is achieved. Then (9) follows from (13). Moreover, from (15) and (9), Micro-F1 $= 1$ when $\hat{K}_i = K_i$. □

1.3 Proof of Theorem 3

Proof. By properties of multi-class problems mentioned before the theorem statement,

$$K_i = \hat{K}_i = 1. \tag{16}$$

The TP sum in (13) is now the same as the number of correct predictions among all instances. With (16), the proof follows from (13). □

	#nodes	#edges	#degrees per node	Edge Type
BlogCatalog[1]	10,312	333,983	64.78	undirected
Flickr[1]	80,513	5,899,882	146.56	undirected
YouTube[1]	1,138,499	2,990,443	5.25	undirected
PPI[2]	56,944	409,358	14.38	undirected

[1] http://leitang.net/social_dimension.html
[2] http://snap.stanford.edu/graphsage/

Table 4: Dataset Statistics

2 Details of Generating Embedding Vectors

Table 4 shows the data source used in experiments. The data are first preprocessed to the data format used in the various graph embedding toolkits. Since the objective of the graph embedding models is to learn and obtain an informative embedding for each node from the observed edges of the given graph, the label data is not used in this stage. The learned embeddings are then used in node classification. Due to the modeling process being unsupervised (no labels for the downstream tasks), the hyperparameters have to be picked carefully. To obtain consistent and comparable results, we have collected the applied hyperparameter settings from several representative papers in Tables 5, 6, and 7, and stated what values we set. In our experiments, all the generated embeddings are run by the toolkit released by the authors, which are listed in the second row of each table.

3 Additional Implementation Details

For training logistic regression, we use the Newton method (option -s 0) in LIBLINEAR (Fan et al., 2008), with stopping tolerance -e 0.0001. For methods that use LIBLINEAR's parameter-selection functionality (e.g., cost-sensitive), the parameter -C is used. Otherwise, either a C value is specified by -c (e.g., cost-sensitive-simple), or the default cost -c 1 is used. For all cross validations, three folds were used.

3.1 Execution Environment for Graph Representation Learning

The upstream graph representation learning task is performed on one machine with Intel Xeon Gold 6230 CPU @ 2.10GHz. The software version used is Python 3.8.10 and g++ 9.3.0.

3.2 Execution Environment for Classification Task

The downstream classification task is performed by distributing workload to four machines with Intel Xeon E5-2620 v4 @ 2.10GHz and nine machines with Intel Xeon E5-2620 0 @ 2.00GHz. The software versions used on all machines are Python 3.9.6, g++ 11.1.0 and matlab R2019b.

4 Complete Experimental Results

In Tables 8, 9 and 10 we compare, respectively, the Macro-F1, Micro-F1 and Instance-F1 of all the training/prediction methods performed on the embedding vectors generated by the three methods of representation learning.

	Dimension	Window Size	Walk Length	Walk Times	NS or HS[1]
Defaults in the tool by DeepWalk authors[2]	64	5	40	10	HS
Perozzi et al. (2014)	128	10	-	80	HS
Tang et al. (2015)	128	10	40	40	HS
Grover and Leskovec (2016)	128	10	80	10	-
Khosla et al. (2021)	128	10	40	80	-
Our Setting	128	10	40	80	HS

[1] NS denotes Negative Sampling, and HS denotes Hierarchical Softmax.
[2] https://github.com/phanein/deepwalk. This is the source code provided by Perozzi et al. (2014).

Table 5: DeepWalk's hyperparameters

	Dimension	Window Size	Walk Length	Walk Times	(p, q)
Defaults in the tool by Node2vec authors[1]	128	10	80	10	$(1, 1)$
Grover and Leskovec (2016)	128	10	-	80	$p, q \in \{0.25, 0.5, 1, 2, 4\}$
Khosla et al. (2021)	128	10	40	80	$p, q \in \{0.25, 0.5, 1, 2, 4\}$
Our Setting	128	10	40	80	$(4, 0.25)$[2]

[1] https://github.com/aditya-grover/node2vec. This is the source code provided by Grover and Leskovec (2016).
[2] According to the assumption of Node2vec, $(4, 0.25)$ encourages the model to generate the DFS-style random walks, while $(0.25, 4)$ generates the BFS-style random walks. The DFS one is generally better so we adopt it in the main paper.

Table 6: Node2vec's hyperparameters

	Dimension (1st + 2nd)	Sample Times	Negative Samples
Defaults in the tool by LINE authors[1]	100 + 100	-	5
Tang et al. (2015)	128 + 128[†]	10 Billion	5
Grover and Leskovec (2016)	64 + 64	$10\times\|E\|$[‡]	5
Khosla et al. (2021)	64 + 64	10 Billion	5
Our Setting	64 + 64	10 Billion	5

[1] https://github.com/tangjianpku/LINE. This is the source code provided by Tang et al. (2015).
[†] Most subsequent works set the dimension to be 64 instead of 128 for each module (*i.e.* LINE-1st and LINE-2nd). The purpose is to use the same 128 dimensions as other models.
[‡] $|E|$ represents the number of edges in the given graph.

Table 7: LINE's hyperparameters

We further observe that

- LINE has a larger improvement with one-vs-rest-basic-C over one-vs-rest-basic. This is because for binary logistic regression, there is a C value under which complete underfitting occurs (Chu et al., 2015), given by
$$C < \frac{1}{l \max_i ||\boldsymbol{x}_i||^2}.$$
For DeepWalk and Node2vec, no normalization is done, and the resulting embedding vectors have varying L2-norm dependent on the data set. In our experiments, we observed that
$$10 < \max_i ||\boldsymbol{x}_i||^2 < 500.$$
For the LINE setting, as shown in Table 7, we choose Line-1st + LINE-2nd, so each embedding vector is a concatenation of two unit L2-norm vectors both with half the embedding vector dimension. Thus, we have
$$\max_i ||\boldsymbol{x}_i||^2 = 2.$$
We see that LINE is the closest to underfitting with $C = 1$ in one-vs-rest-basic, and therefore, LINE also benefits from one-vs-rest-basic-C more than the other methods.

4.1 Choice of CV Splits

A simple way to implement cost-sensitive-simple is as follows

- For each (C, t)
  - Split data to folds
  - Sequentially validate each fold

Under each (C, t), a standard CV procedure is conducted and and one can call the CV functionality in a package such as LIBLINEAR for the inner loop in the pseudocode. However, if we decide to use the same data splits in all CV processes, we generally need to maintain the data split by ourselves. The procedure is as follows

- Split data to folds
- For each (C, t)
  - Sequentially validate each fold

Therefore maintaining the same data splits causes less flexibility in implementation.

However, as was mentioned previously, we suspect that different data splits may cause a higher variation of the results. We conduct a comparison here to see if this conjecture is true. In the tables, we denote the method of having different data splits for each (C, t) as cost-sensitive-simple-rand.

We observe that the difference between cost-sensitive-simple and cost-sensitive-simple-rand is very small in all cases. This robustness against the choice of CV splits allows the implementation flexibility.

4.2 Choice of (C, t) values for cost-sensitive-simple

For the cost-sensitive family of methods, there is a problem of choosing the grid of (C, t) values. A denser grid generally yields better performance at the cost of longer running time. Here we discuss how the final grid is chosen for cost-sensitive-simple.

In Tables 11, 12, 13 we compare, respectively, the Macro-F1, Micro-F1 and Instance-F1 of choosing various ranges of C values with the same range of t values.[1] We observe that in most cases there is a marginal improvement with an increase of the number of C values used. However, the training time for higher C values are drastically longer, in some cases up to 70 times longer than $C = 1$. Due to the insignificant improvement at a high cost, we chose to use a single value $C = 1$.

Parambath et al. (2014) gave a theoretical bound relating the density of t and the performance of cost-sensitive. However, the bound is very loose and cannot be used as a practical rule to choose a proper range of t. In Tables 14, 15 and 16 we compare the Macro-F1, Micro-F1 and Instance-F1 of different numbers of t values chosen equidistantly from the range $(0, 1]$ with $C = 1$. We observe that Macro-F1 is slightly better with more t values. However, Micro-F1 and Instance-F1 are usually better with a smaller number of t values. In the end we decide to choose seven t values for cost-sensitive-simple based on balance of Macro-F1 performance and running time. The justification for emphasizing Macro-F1 is given in Section 4.3.

4.3 The Macro-F1 and Micro-F1 Tradeoff

In this section we explain that observations from Tables 14–16 may be related to the different best t value settings for Macro-F1 and Micro-F1. Note that optimizing Macro-F1 and Micro-F1 is often a tradeoff. Because Macro-F1 is the average of label-wise F1 scores, the performance on a rare label is equally important to the performance on a frequent label. In contrast, Micro-F1 is the overall F1 score by considering all labels together. To confirm this difference, we consider another cost-sensitive method to optimize Micro-F1 (Parambath et al., 2014):

- cost-sensitive-simple-micro: The same (C, t) pair is used for all labels and the best is chosen by cross-validation Micro-F1 score. The grid of (C, t) pairs is the same as cost-sensitive-simple.

In Tables 17, 18 and 19 we compare cost-sensitive-simple and cost-sensitive-simple-micro by checking their Macro-F1, Micro-F1 and Instance-F1. We use $C = 1$ and consider seven t values chosen equidistantly from $(0, 1]$. We observe that cost-sensitive-simple-micro is significantly better at optimizing Micro-F1, but significantly worse at optimizing Macro-F1. The same observations hold for other settings of choosing t values.

We also observe that Instance-F1 has the same trend as Micro-F1. This result may be because that similar to Micro-F1, Instance-F1 is defined without particularly checking the predictions of rare labels.

5 Papers or Libraries Assuming the Number of Labels is Known in the Prediction Stage

In this section we list papers or libraries assuming the number of labels is known, either explicitly stated through text or visible in their evaluation code. We also include a section of likely suspects that implicitly cite these papers or libraries in their evaluation procedures.

[1] Seven values chosen equidistantly from $(0, 1]$

Training and prediction methods	Macro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
unrealistic	0.276 ± 0.005	0.294 ± 0.004	0.239 ± 0.008
one-vs-rest-basic	0.190 ± 0.010	0.203 ± 0.010	0.150 ± 0.003
one-vs-rest-basic-C	0.208 ± 0.008	0.220 ± 0.008	0.195 ± 0.007
one-vs-rest-no-empty	0.241 ± 0.011	0.265 ± 0.007	0.204 ± 0.011
thresholding	0.269 ± 0.006	0.283 ± 0.006	0.221 ± 0.006
cost-sensitive	**0.270** ± 0.003	**0.283** ± 0.006	0.250 ± 0.007
cost-sensitive-no-empty	0.268 ± 0.004	0.280 ± 0.007	**0.251** ± 0.007
cost-sensitive-simple	0.266 ± 0.005	0.275 ± 0.005	0.251 ± 0.007
cost-sensitive-simple-rand	0.261 ± 0.006	0.276 ± 0.005	0.251 ± 0.009
		Flickr	
unrealistic	0.304 ± 0.003	0.306 ± 0.001	0.258 ± 0.005
one-vs-rest-basic	0.195 ± 0.003	0.191 ± 0.002	0.128 ± 0.004
one-vs-rest-basic-C	0.209 ± 0.001	0.208 ± 0.002	0.188 ± 0.003
one-vs-rest-no-empty	0.256 ± 0.001	0.259 ± 0.002	0.206 ± 0.004
thresholding	**0.299** ± 0.004	**0.302** ± 0.002	0.264 ± 0.002
cost-sensitive	0.297 ± 0.000	0.301 ± 0.002	0.279 ± 0.003
cost-sensitive-no-empty	0.298 ± 0.000	0.301 ± 0.002	**0.284** ± 0.006
cost-sensitive-simple	0.294 ± 0.001	0.296 ± 0.002	0.279 ± 0.002
cost-sensitive-simple-rand	0.293 ± 0.001	0.295 ± 0.002	0.279 ± 0.003
		YouTube	
unrealistic	0.397 ± 0.007	0.415 ± 0.009	0.412 ± 0.005
one-vs-rest-basic	0.213 ± 0.003	0.240 ± 0.006	0.230 ± 0.004
one-vs-rest-basic-C	0.217 ± 0.004	0.242 ± 0.006	0.246 ± 0.005
one-vs-rest-no-empty	0.263 ± 0.003	0.284 ± 0.006	0.281 ± 0.004
thresholding	0.358 ± 0.005	0.374 ± 0.006	**0.377** ± 0.004
cost-sensitive	**0.360** ± 0.005	**0.375** ± 0.003	0.374 ± 0.004
cost-sensitive-no-empty	0.359 ± 0.005	0.374 ± 0.003	0.375 ± 0.004
cost-sensitive-simple	0.349 ± 0.005	0.369 ± 0.005	0.371 ± 0.005
cost-sensitive-simple-rand	0.349 ± 0.005	0.368 ± 0.005	0.372 ± 0.004
		PPI	
unrealistic	0.483 ± 0.003	0.442 ± 0.003	0.504 ± 0.003
one-vs-rest-basic	0.181 ± 0.001	0.148 ± 0.000	0.232 ± 0.002
one-vs-rest-basic-C	0.183 ± 0.001	0.150 ± 0.000	0.243 ± 0.002
one-vs-rest-no-empty	0.181 ± 0.001	0.148 ± 0.000	0.232 ± 0.002
thresholding	**0.482** ± 0.002	0.457 ± 0.002	**0.498** ± 0.001
cost-sensitive	0.482 ± 0.002	**0.461** ± 0.002	0.495 ± 0.002
cost-sensitive-no-empty	0.482 ± 0.002	0.461 ± 0.002	0.495 ± 0.002
cost-sensitive-simple	0.481 ± 0.002	0.460 ± 0.002	0.494 ± 0.002
cost-sensitive-simple-rand	0.481 ± 0.002	0.460 ± 0.002	0.494 ± 0.002

Table 8: Macro-F1 of various training/prediction techniques on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method (excluding unrealistic) is bold-faced.

Training and prediction methods	Micro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
unrealistic	0.417 ± 0.005	0.426 ± 0.006	0.406 ± 0.007
one-vs-rest-basic	0.334 ± 0.007	0.348 ± 0.012	0.296 ± 0.005
one-vs-rest-basic-C	0.344 ± 0.006	0.355 ± 0.012	0.335 ± 0.005
one-vs-rest-no-empty	0.390 ± 0.003	**0.404** ± 0.006	**0.370** ± 0.007
thresholding	**0.390** ± 0.003	0.396 ± 0.010	0.353 ± 0.005
cost-sensitive	0.366 ± 0.007	0.371 ± 0.003	0.341 ± 0.004
cost-sensitive-no-empty	0.351 ± 0.010	0.360 ± 0.007	0.324 ± 0.009
cost-sensitive-simple	0.351 ± 0.006	0.362 ± 0.017	0.337 ± 0.004
cost-sensitive-simple-rand	0.353 ± 0.010	0.363 ± 0.006	0.332 ± 0.002
		Flickr	
unrealistic	0.416 ± 0.002	0.420 ± 0.005	0.409 ± 0.004
one-vs-rest-basic	0.283 ± 0.003	0.288 ± 0.002	0.271 ± 0.004
one-vs-rest-basic-C	0.291 ± 0.002	0.296 ± 0.002	0.289 ± 0.006
one-vs-rest-no-empty	**0.377** ± 0.002	**0.382** ± 0.004	**0.373** ± 0.004
thresholding	0.370 ± 0.002	0.376 ± 0.003	0.364 ± 0.001
cost-sensitive	0.352 ± 0.003	0.358 ± 0.002	0.354 ± 0.005
cost-sensitive-no-empty	0.343 ± 0.006	0.355 ± 0.006	0.348 ± 0.013
cost-sensitive-simple	0.355 ± 0.003	0.360 ± 0.004	0.357 ± 0.004
cost-sensitive-simple-rand	0.356 ± 0.002	0.359 ± 0.005	0.356 ± 0.006
		YouTube	
unrealistic	0.470 ± 0.008	0.482 ± 0.008	0.480 ± 0.004
one-vs-rest-basic	0.287 ± 0.008	0.313 ± 0.007	0.315 ± 0.004
one-vs-rest-basic-C	0.290 ± 0.009	0.314 ± 0.007	0.325 ± 0.006
one-vs-rest-no-empty	0.382 ± 0.005	0.399 ± 0.004	0.402 ± 0.004
thresholding	**0.387** ± 0.006	**0.409** ± 0.003	**0.412** ± 0.005
cost-sensitive	0.374 ± 0.010	0.400 ± 0.006	0.403 ± 0.004
cost-sensitive-no-empty	0.372 ± 0.008	0.400 ± 0.005	0.404 ± 0.004
cost-sensitive-simple	0.365 ± 0.006	0.390 ± 0.007	0.400 ± 0.005
cost-sensitive-simple-rand	0.368 ± 0.010	0.390 ± 0.003	0.401 ± 0.004
		PPI	
unrealistic	0.641 ± 0.001	0.626 ± 0.001	0.647 ± 0.001
one-vs-rest-basic	0.449 ± 0.001	0.433 ± 0.001	0.479 ± 0.001
one-vs-rest-basic-C	0.458 ± 0.001	0.441 ± 0.001	0.489 ± 0.002
one-vs-rest-no-empty	0.449 ± 0.001	0.433 ± 0.001	0.479 ± 0.001
thresholding	**0.535** ± 0.002	0.482 ± 0.002	**0.553** ± 0.001
cost-sensitive	0.533 ± 0.002	**0.495** ± 0.002	0.548 ± 0.002
cost-sensitive-no-empty	0.533 ± 0.002	**0.495** ± 0.002	0.548 ± 0.002
cost-sensitive-simple	0.529 ± 0.002	0.495 ± 0.002	0.547 ± 0.002
cost-sensitive-simple-rand	0.529 ± 0.002	0.494 ± 0.002	0.548 ± 0.002

Table 9: Micro-F1 of various training/prediction techniques on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method (excluding unrealistic) is bold-faced.

Training and prediction methods	Instance-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
unrealistic	0.422 ± 0.004	0.433 ± 0.006	0.408 ± 0.005
one-vs-rest-basic	0.255 ± 0.004	0.265 ± 0.009	0.222 ± 0.004
one-vs-rest-basic-C	0.267 ± 0.003	0.275 ± 0.009	0.256 ± 0.003
one-vs-rest-no-empty	**0.403** ± 0.003	**0.416** ± 0.006	**0.389** ± 0.006
thresholding	0.347 ± 0.006	0.354 ± 0.012	0.315 ± 0.002
cost-sensitive	0.340 ± 0.006	0.348 ± 0.003	0.314 ± 0.005
cost-sensitive-no-empty	0.352 ± 0.011	0.363 ± 0.011	0.325 ± 0.015
cost-sensitive-simple	0.327 ± 0.004	0.335 ± 0.011	0.313 ± 0.006
cost-sensitive-simple-rand	0.327 ± 0.003	0.334 ± 0.007	0.311 ± 0.004
		Flickr	
unrealistic	0.406 ± 0.002	0.410 ± 0.004	0.394 ± 0.004
one-vs-rest-basic	0.211 ± 0.002	0.212 ± 0.002	0.185 ± 0.002
one-vs-rest-basic-C	0.217 ± 0.001	0.220 ± 0.002	0.207 ± 0.003
one-vs-rest-no-empty	**0.386** ± 0.002	**0.390** ± 0.004	**0.376** ± 0.004
thresholding	0.352 ± 0.002	0.353 ± 0.005	0.325 ± 0.002
cost-sensitive	0.342 ± 0.001	0.343 ± 0.003	0.326 ± 0.002
cost-sensitive-no-empty	0.345 ± 0.008	0.357 ± 0.008	0.349 ± 0.018
cost-sensitive-simple	0.344 ± 0.006	0.345 ± 0.005	0.329 ± 0.008
cost-sensitive-simple-rand	0.346 ± 0.005	0.343 ± 0.007	0.328 ± 0.008
		YouTube	
unrealistic	0.454 ± 0.007	0.467 ± 0.006	0.464 ± 0.003
one-vs-rest-basic	0.245 ± 0.009	0.262 ± 0.008	0.254 ± 0.004
one-vs-rest-basic-C	0.247 ± 0.010	0.263 ± 0.008	0.263 ± 0.005
one-vs-rest-no-empty	**0.412** ± 0.005	**0.427** ± 0.005	**0.428** ± 0.003
thresholding	0.389 ± 0.006	0.400 ± 0.006	0.398 ± 0.006
cost-sensitive	0.380 ± 0.011	0.399 ± 0.006	0.394 ± 0.004
cost-sensitive-no-empty	0.388 ± 0.010	0.414 ± 0.005	0.416 ± 0.006
cost-sensitive-simple	0.372 ± 0.009	0.388 ± 0.007	0.388 ± 0.004
cost-sensitive-simple-rand	0.377 ± 0.012	0.388 ± 0.006	0.388 ± 0.006
		PPI	
unrealistic	0.577 ± 0.002	0.565 ± 0.001	0.583 ± 0.002
one-vs-rest-basic	0.449 ± 0.001	0.438 ± 0.001	0.464 ± 0.001
one-vs-rest-basic-C	0.457 ± 0.001	0.445 ± 0.002	0.475 ± 0.001
one-vs-rest-no-empty	0.449 ± 0.001	0.438 ± 0.001	0.464 ± 0.001
thresholding	**0.503** ± 0.002	0.456 ± 0.002	**0.517** ± 0.001
cost-sensitive	0.501 ± 0.002	0.467 ± 0.002	0.513 ± 0.002
cost-sensitive-no-empty	0.501 ± 0.002	**0.467** ± 0.002	0.513 ± 0.002
cost-sensitive-simple	0.496 ± 0.002	0.466 ± 0.002	0.512 ± 0.002
cost-sensitive-simple-rand	0.497 ± 0.002	0.466 ± 0.002	0.513 ± 0.002

Table 10: Instance-F1 of various training/prediction techniques on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method (excluding unrealistic) is bold-faced.

Choice of C values	Macro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
1	**0.266** ± 0.005	0.275 ± 0.005	**0.251** ± 0.007
1, 10, 100	0.266 ± 0.005	0.276 ± 0.006	0.249 ± 0.006
0.01, 0.1, 1, 10, 100	0.266 ± 0.004	**0.282** ± 0.005	0.250 ± 0.007
		Flickr	
1	0.294 ± 0.001	0.296 ± 0.002	0.279 ± 0.002
1, 10, 100	0.293 ± 0.001	0.295 ± 0.003	0.278 ± 0.002
0.01, 0.1, 1, 10, 100	**0.297** ± 0.002	**0.299** ± 0.002	**0.279** ± 0.002
		YouTube	
1	0.349 ± 0.005	0.369 ± 0.005	0.371 ± 0.005
1, 10, 100	0.348 ± 0.005	0.368 ± 0.005	0.371 ± 0.004
0.01, 0.1, 1, 10, 100	**0.357** ± 0.004	**0.374** ± 0.005	**0.374** ± 0.004
		PPI	
1	0.481 ± 0.002	0.460 ± 0.002	0.494 ± 0.002
1, 10, 100	0.481 ± 0.002	0.460 ± 0.002	**0.494** ± 0.002
0.01, 0.1, 1, 10, 100	**0.481** ± 0.002	**0.460** ± 0.002	0.494 ± 0.002

Table 11: Macro-F1 of various choices of C values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

Choice of C values	Micro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
1	0.351 ± 0.006	0.362 ± 0.017	0.337 ± 0.004
1, 10, 100	**0.355** ± 0.006	0.367 ± 0.017	0.336 ± 0.009
0.01, 0.1, 1, 10, 100	0.353 ± 0.010	**0.371** ± 0.011	**0.342** ± 0.007
		Flickr	
1	0.355 ± 0.003	0.360 ± 0.004	0.357 ± 0.004
1, 10, 100	0.355 ± 0.003	**0.361** ± 0.005	**0.360** ± 0.005
0.01, 0.1, 1, 10, 100	**0.358** ± 0.003	0.361 ± 0.003	0.358 ± 0.005
		YouTube	
1	0.365 ± 0.006	0.390 ± 0.007	0.400 ± 0.005
1, 10, 100	0.365 ± 0.006	0.391 ± 0.008	**0.401** ± 0.005
0.01, 0.1, 1, 10, 100	**0.372** ± 0.007	**0.395** ± 0.006	0.400 ± 0.004
		PPI	
1	0.529 ± 0.002	0.495 ± 0.002	0.547 ± 0.002
1, 10, 100	**0.529** ± 0.002	0.495 ± 0.002	**0.547** ± 0.002
0.01, 0.1, 1, 10, 100	0.529 ± 0.002	**0.495** ± 0.002	0.547 ± 0.002

Table 12: Micro-F1 of various choices of C values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

Choice of C values	Instance-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
1	0.327 ± 0.004	0.335 ± 0.011	**0.313** ± 0.006
1, 10, 100	0.329 ± 0.003	0.340 ± 0.010	0.307 ± 0.007
0.01, 0.1, 1, 10, 100	**0.333** ± 0.004	**0.350** ± 0.006	0.311 ± 0.004
		Flickr	
1	0.344 ± 0.006	0.345 ± 0.005	0.329 ± 0.008
1, 10, 100	0.344 ± 0.005	0.345 ± 0.005	0.330 ± 0.008
0.01, 0.1, 1, 10, 100	**0.344** ± 0.006	**0.345** ± 0.005	**0.331** ± 0.007
		YouTube	
1	0.372 ± 0.009	0.388 ± 0.007	0.388 ± 0.004
1, 10, 100	0.373 ± 0.010	0.389 ± 0.010	**0.388** ± 0.005
0.01, 0.1, 1, 10, 100	**0.378** ± 0.009	**0.392** ± 0.008	0.384 ± 0.006
		PPI	
1	0.496 ± 0.002	0.466 ± 0.002	0.512 ± 0.002
1, 10, 100	0.496 ± 0.002	0.466 ± 0.002	**0.513** ± 0.002
0.01, 0.1, 1, 10, 100	**0.497** ± 0.002	**0.466** ± 0.002	0.512 ± 0.002

Table 13: Instance-F1 of various choices of C values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

5.1 Papers Explicitly Assuming the Number of Labels is Known

1. Tang and Liu (2009a): The paper is at https://dl.acm.org/doi/pdf/10.1145/1557019.1557109. "It has been shown that different thresholding strategies lead to quite different performance (Fan and Lin, 2007; Tang et al., 2009). To avoid the affection of thresholding, we assume the number of labels on the test data are already known and check how the top ranking predictions match with the true labels. Two commonly used measures Micro-F1 and Macro-F1 are adopted to evaluate the classification performance."

2. Tang and Liu (2009b): The paper is at https://dl.acm.org/doi/pdf/10.1145/1645953.1646094. "Note that our prediction problem is essentially multi-label. It is empirically shown that thresholding can affect the final prediction performance drastically (Fan and Lin, 2007; Tang et al., 2009). For evaluation purpose, we assume the number of labels of unobserved nodes is already known and check the match of the top-ranking labels with the truth. Such a scheme has been adopted for other multi-label evaluation works Liu et al. (2006). We randomly sample a portion of nodes as labeled and report the average performance of 10 runs in terms of Micro-F1 and Macro-F1. We use the same setting as in Tang and Liu (2009a) for the baseline methods for Flickr and BlogCatalog, thus the performance on the two data sets are reported here directly."

3. Tang et al. (2010): The paper is at https://dl.acm.org/doi/pdf/10.1145/1964858.1964861. "We follow the same evaluation procedure as in Tang and Liu (2009a,b)."

4. Menon and Elkan (2010): The paper is at https://link.springer.com/content/pdf/10.1007/s10618-010-0189-3.pdf. "Following Tang and Liu (2009a), for multilabel data we assume that the number of labels are

# of t values	Macro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
4	0.265 ± 0.004	0.274 ± 0.010	0.245 ± 0.010
7	**0.266** ± 0.005	**0.275** ± 0.005	0.251 ± 0.007
10	0.264 ± 0.007	0.274 ± 0.009	**0.252** ± 0.012
		Flickr	
4	0.291 ± 0.002	0.294 ± 0.004	0.276 ± 0.002
7	**0.294** ± 0.001	0.296 ± 0.002	0.279 ± 0.002
10	0.294 ± 0.002	**0.297** ± 0.003	**0.280** ± 0.003
		YouTube	
4	0.345 ± 0.004	0.365 ± 0.005	0.371 ± 0.005
7	0.349 ± 0.005	0.369 ± 0.005	0.371 ± 0.005
10	**0.350** ± 0.005	**0.370** ± 0.004	**0.372** ± 0.004
		PPI	
4	0.477 ± 0.002	0.459 ± 0.002	0.490 ± 0.002
7	0.481 ± 0.002	0.460 ± 0.002	0.494 ± 0.002
10	**0.482** ± 0.002	**0.461** ± 0.002	**0.495** ± 0.002

Table 14: Macro-F1 of various number of t values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

# of t values	Micro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
4	**0.377** ± 0.008	**0.385** ± 0.005	**0.356** ± 0.005
7	0.351 ± 0.006	0.362 ± 0.017	0.337 ± 0.004
10	0.334 ± 0.009	0.349 ± 0.010	0.320 ± 0.011
		Flickr	
4	**0.374** ± 0.002	**0.376** ± 0.002	**0.373** ± 0.002
7	0.355 ± 0.003	0.360 ± 0.004	0.357 ± 0.004
10	0.352 ± 0.007	0.361 ± 0.003	0.356 ± 0.005
		YouTube	
4	**0.392** ± 0.004	**0.408** ± 0.006	**0.410** ± 0.003
7	0.365 ± 0.006	0.390 ± 0.007	0.400 ± 0.005
10	0.362 ± 0.009	0.400 ± 0.007	0.406 ± 0.004
		PPI	
4	0.517 ± 0.002	0.491 ± 0.002	0.540 ± 0.002
7	0.529 ± 0.002	0.495 ± 0.002	**0.547** ± 0.002
10	**0.533** ± 0.002	**0.495** ± 0.002	0.547 ± 0.002

Table 15: Micro-F1 of various number of t values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

# of t values	Instance-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
4	**0.338** ± 0.009	**0.344** ± 0.003	**0.319** ± 0.003
7	0.327 ± 0.004	0.335 ± 0.011	0.313 ± 0.006
10	0.321 ± 0.007	0.330 ± 0.006	0.304 ± 0.007
		Flickr	
4	**0.353** ± 0.001	**0.351** ± 0.002	**0.337** ± 0.002
7	0.344 ± 0.006	0.345 ± 0.005	0.329 ± 0.008
10	0.341 ± 0.004	0.343 ± 0.001	0.329 ± 0.004
		YouTube	
4	**0.393** ± 0.006	**0.404** ± 0.005	**0.399** ± 0.005
7	0.372 ± 0.009	0.388 ± 0.007	0.388 ± 0.004
10	0.373 ± 0.008	0.399 ± 0.006	0.395 ± 0.005
		PPI	
4	0.485 ± 0.002	0.462 ± 0.002	0.505 ± 0.002
7	0.496 ± 0.002	0.466 ± 0.002	0.512 ± 0.002
10	**0.501** ± 0.002	**0.467** ± 0.002	**0.513** ± 0.003

Table 16: Instance-F1 of various number of t values for cost-sensitive-simple on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

known, and we measure how well the predicted score for each tag agrees with the true label. Agreement is measured using the F1 micro and macro scores."

5. Tang and Liu (2011): The paper is at https://link.springer.com/content/pdf/10.1007/s10618-010-0210-x.pdf. "It has been shown that different thresholding strategies lead to quite different performances (Fan and Lin, 2007; Tang et al., 2009). To avoid the effect of thresholding, we assume the number of labels on the test data is already known and check how the top-ranking predictions match with the true labels."

6. Wang and Sukthankar (2011): The paper is at https://ieeexplore.ieee.org/abstract/document/6113224. "Since our classification problem is essentially a multi-label task, during the prediction procedure, we assume that the number of labels for the unlabeled nodes is already known and assign the labels according to the top-ranking class. Such a scheme has been adopted for multi-label evaluation in social network datasets (Tang and Liu, 2009a,b)."

7. Tang et al. (2012): The paper is at https://ieeexplore.ieee.org/abstract/document/5710923. "It is empirically shown that thresholding can affect the final prediction performance drastically (Fan and Lin, 2007; Tang et al., 2009). For evaluation purposes, we assume the number of labels of unobserved nodes is already known, and check whether the top-ranking predicted labels match with the actual labels."

8. Wang and Sukthankar (2013): The paper is at https://dl.acm.org/doi/abs/10.1145/2487575.2487610. "Since our problem is essentially a multi-label classification task, we assume that the number of labels for the unlabeled nodes is already known (e.g., based on the output of a separate classifier) and assign the labels according to the top-ranking set of classes at the conclusion of the inference process. Such a scheme

Training and prediction methods	Macro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
cost-sensitive-simple	**0.266** ± 0.005	**0.275** ± 0.005	**0.251** ± 0.007
cost-sensitive-simple-micro	0.251 ± 0.006	0.259 ± 0.008	0.231 ± 0.008
		Flickr	
cost-sensitive-simple	**0.294** ± 0.001	**0.296** ± 0.002	**0.279** ± 0.002
cost-sensitive-simple-micro	0.279 ± 0.002	0.283 ± 0.002	0.255 ± 0.003
		YouTube	
cost-sensitive-simple	**0.349** ± 0.005	**0.369** ± 0.005	**0.371** ± 0.005
cost-sensitive-simple-micro	0.324 ± 0.003	0.350 ± 0.005	0.351 ± 0.003
		PPI	
cost-sensitive-simple	**0.481** ± 0.002	**0.460** ± 0.002	**0.494** ± 0.002
cost-sensitive-simple-micro	0.408 ± 0.001	0.358 ± 0.002	0.446 ± 0.002

Table 17: Macro-F1 for cost-sensitive-simple and cost-sensitive-simple-micro on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

Training and prediction methods	Micro-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
cost-sensitive-simple	0.351 ± 0.006	0.362 ± 0.017	0.337 ± 0.004
cost-sensitive-simple-micro	**0.398** ± 0.004	**0.409** ± 0.008	**0.383** ± 0.006
		Flickr	
cost-sensitive-simple	0.355 ± 0.003	0.360 ± 0.004	0.357 ± 0.004
cost-sensitive-simple-micro	**0.395** ± 0.003	**0.398** ± 0.002	**0.397** ± 0.002
		YouTube	
cost-sensitive-simple	0.365 ± 0.006	0.390 ± 0.007	0.400 ± 0.005
cost-sensitive-simple-micro	**0.417** ± 0.005	**0.432** ± 0.004	**0.433** ± 0.002
		PPI	
cost-sensitive-simple	0.529 ± 0.002	0.495 ± 0.002	0.547 ± 0.002
cost-sensitive-simple-micro	**0.561** ± 0.001	**0.545** ± 0.002	**0.576** ± 0.002

Table 18: Micro-F1 for cost-sensitive-simple and cost-sensitive-simple-micro on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

Training and prediction methods	Instance-F1 DeepWalk	Node2vec	LINE
		BlogCatalog	
cost-sensitive-simple	0.327 ± 0.004	0.335 ± 0.011	0.313 ± 0.006
cost-sensitive-simple-micro	**0.343** ± 0.016	**0.347** ± 0.006	**0.342** ± 0.004
		Flickr	
cost-sensitive-simple	0.344 ± 0.006	0.345 ± 0.005	0.329 ± 0.008
cost-sensitive-simple-micro	**0.357** ± 0.002	**0.360** ± 0.003	**0.343** ± 0.002
		YouTube	
cost-sensitive-simple	0.372 ± 0.009	0.388 ± 0.007	0.388 ± 0.004
cost-sensitive-simple-micro	**0.417** ± 0.006	**0.427** ± 0.006	**0.418** ± 0.003
		PPI	
cost-sensitive-simple	0.496 ± 0.002	0.466 ± 0.002	0.512 ± 0.002
cost-sensitive-simple-micro	**0.529** ± 0.001	**0.514** ± 0.002	**0.537** ± 0.002

Table 19: Micro-F1 for cost-sensitive-simple and cost-sensitive-simple-micro on embedding vectors generated by some representation learning methods. Each entry is the average and standard deviation of five 80/20 training/testing splits. The score of the best training/prediction method is bold-faced.

has been adopted for multi-label evaluation in social network datasets (Tang and Liu, 2009b; Wang and Sukthankar, 2011)."

9. Wang et al. (2013): The paper is at https://link.springer.com/article/10.1007/s10115-012-0555-0. "We follow the same evaluation procedure as in Tang and Liu (2009a,b).[2]"

10. Li et al. (2016b): The paper is at https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0152857. "To evaluate the performance of different methods, dataset will be divided into training set and testing set randomly. The labels of nodes in the training set are known, and the labels of nodes in the testing set are unknown. All the classification methods need to utilize the training set to predict the labels of all nodes in the testing set. For handling multi-label classification task, we apply the typically used strategy, which assumes the number of labels of unlabeled nodes is already known and check the match of the top-ranking labels with the truth (Wang and Sukthankar, 2013; Tang and Liu, 2011, 2009b)."

11. Li et al. (2016a): The paper is at https://aclanthology.org/P16-1095.pdf. As the datasets are not only multi-class but also multi-label, we usually need a thresholding method to test the results. But literature gives a negative opinion of arbitrarily choosing thresholding methods because of the considerably different performances. To avoid this, we assume that the number of the labels is already known in all the test processes.

12. Nandanwar and Murty (2016): The paper is at https://dl.acm.org/doi/abs/10.1145/2939672.2939782. "For each node in the test set, decision values are obtained from the respective class models. We assign s most probable classes to the node using these decision values, where s is equal to the number of labels assigned to the node originally."

[2]http://leitang.net/social_dimension.html

13. Rizos et al. (2017): The paper is at https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0173347. "Following Tang and Liu (2009a); Devooght et al. (2014), we assume the true number of labels for each vertex to be known."

14. Qiu et al. (2018): The paper is at https://dl.acm.org/doi/abs/10.1145/3159652.3159706. "To avoid the thresholding effect (Tang et al., 2009), we assume that the number of labels for test data is given (Perozzi et al., 2014; Tang et al., 2009). We repeat the prediction procedure 10 times and evaluate the performance in terms of average Micro-F1 and average Macro-F1 (Tsoumakas et al., 2010)."

15. Zhang et al. (2018d): The paper is at https://www.sciencedirect.com/science/article/pii/S0925231218302467. "For evaluation purpose, we assume that the number of labels for the unlabeled nodes is already known and we assign the labels according to the top-ranking set of class membership probabilities. Such a scheme has been adopted for multi-label evaluation in social network datasets (Wang and Sukthankar, 2013; Tang and Liu, 2009a)."

16. Zhang et al. (2018c): The paper is at https://www.sciencedirect.com/science/article/abs/pii/S0950705118303186. "For evaluation purpose, we assume that the number of labels for the unlabeled nodes is already known and assign the labels according to the top-ranking set of class-membership probabilities. Such a scheme has been adopted for multi-label evaluation in social network datasets (Tang et al., 2012)"

17. Zhang et al. (2018b): The paper is at https://link.springer.com/chapter/10.1007/978-3-030-02922-7_25. "if the probability of a test instance belonging to a label is more than 0.5, MLKNN assigns this class label to this test instance. We made a minor change. For fair comparison, in all methods, we assume that the number of labels for all unlabeled nodes are already known and we assign the labels based on the classes with highest probability. Such a scheme has been adopted for multilabel evaluation in social network data sets (Tang and Liu, 2009a; Wang and Sukthankar, 2013) and improve the performance of MLKNN."

18. Goyal and Ferrara (2018): The paper is at https://www.sciencedirect.com/science/article/abs/pii/S0950705118301540. "For data sets with multiple labels per node, we assume that we know how many labels to predict."

19. Ye et al. (2018): The paper is at https://link.springer.com/article/10.1007/s41019-018-0062-8. "In multi-label prediction phrase, the goal is to find the most probable classes for each unlabeled node. Since most methods yield a ranking of labels rather than an exact assignment, a threshold is often required. To avoid the affection of introducing a threshold, we assign s most probable classes to a node, where s is the number of labels assigned to the node originally."

20. Nandanwar and Murty (2018): The paper is at https://ojs.aaai.org/index.php/AAAI/article/view/11787. "In our multilabel setting, each node has a different number of labels assigned to it. For an unlabeled node, we predict a set of labels using predicted class membership probabilities. The cardinality of the predicted label set is equal to that of its original label set."

21. Berberidis et al. (2018): The paper is at https://ieeexplore.ieee.org/abstract/document/8622130. "Similar to Grover and Leskovec (2016) and Perozzi et al. (2014), during evaluation the number of labels per sampled node is known, and check how many of them are in the top predictions."

22. Yin and Wei (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3292500.3330860. "To avoid the thresholding effect (Tang et al., 2009), we assume that the number of labels for test data is given (Perozzi

et al., 2014). The performance of each method is evaluated in terms of average Micro-F1 and average Macro-F1 (Tsoumakas et al., 2010), and we only report Micro-F1 as we experience similar behaviors with Macro-F1. Table 7 and Table 8 show the node classification results on BlogCatalog and Flickr. Surprisingly, DeepWalk outperforms all successors other than STRAP on both datasets."

23. Schlötterer et al. (2019): The paper is at https://ieeexplore.ieee.org/abstract/document/8816899. "Since BlogCatalog is multi-label, we first obtain the number of actual labels to predict for each sample from the test set. Then we predict the k most probable classes, where k is the number of labels to predict. This is a common choice in the evaluation setup of the reproduced methods. All methods report the Macro-F1 score, except for Walklets, reporting Micro-F1, which we follow in the reproduction."

24. Qiu et al. (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3308558.3313446. "To avoid the thresholding effect, we take the assumption that was made in DeepWalk, LINE, and node2vec, that is, the number of labels for vertices in the test data is given Grover and Leskovec (2016); Perozzi et al. (2014); Tang et al. (2009)."

25. Berberidis et al. (2019): The paper is at https://ieeexplore.ieee.org/abstract/document/8590776. "Similar to Grover and Leskovec (2016) and Perozzi et al. (2014), during evaluation of accuracy the number of labels per sampled node is known, and check how many of them are in the top predictions."

26. Berberidis and Giannakis (2019): The paper is at http://www.mlgworkshop.org/2019/papers/MLG2019_paper_40.pdf. "In the testing phase, we sorted the predicted class probabilities for each node in decreasing order, and extracted the top-k_i ranking labels, were k_i is the true number of labels of node v_i. We then computed the Micro- and Macro-averaged F1 scores of the predicted labels."

27. Berberidis and Giannakis (2021): The paper is at https://ieeexplore.ieee.org/abstract/document/8778744. "In the testing phase, we sorted the predicted class probabilities for each node in decreasing order, and extracted the top-k_i ranking labels, were k_i is the true number of labels of node v_i. We then computed the Micro- and Macro-averaged F1 scores of the predicted labels."

28. Postăvaru et al. (2020): The paper is at https://arxiv.org/abs/2010.06992. "The other datasets have multiple labels per node, and we are using a One-vs-The-Rest ensemble. When evaluating, we assume the number of correct labels, K, is known and select the top K probabilities from the ensemble."

29. Yue et al. (2020): The paper is at https://academic.oup.com/bioinformatics/article/36/4/1241/5581350?login=true. "We assign top α_i predictions to the node i as its predicted labels, where α_i is the number of golden labels of the node i in the testing set. Accuracy, Macro-F1 and Micro-F1 are used to evaluate the performance of different embedding methods on the testing set."

30. Devooght et al. (2014): The paper is at https://dl.acm.org/doi/abs/10.1145/2566486.2567986. "Each element of the BlogCatalog and Flickr datasets can have multiple labels, and the presented algorithms produce for each element a ranking of the most probable labels. This ranking is compared to the ground-truth using the micro and macro-average of the F-measure, respectively noted microF1 an macroF1 [20, 26], a pair of metrics well-known in information retrieval. The F-measure is the harmonic mean of precision and recall. The micro- and macro-average are two ways of computing the F-measure when dealing with multiple labels. The microF1 is defined using directly the global recall (ρ) and precision (π) of the results (as in Tang and Liu (2009a,b); Tang et al. (2010), ρ and π are computed on the ranking truncated to the true number of labels)"

31. Krohn-Grimberghe et al. (2012): The paper is at https://dl.acm.org/doi/10.1145/2124295.2124317. "As suggested in Tang and Liu (2009b), we remove the dependency on the length of the top-N list for performance evaluation w.r.t. Micro-F1 and Macro-F1 by assuming that the number of true instances is known at prediction time."

32. Santos et al. (2018): The paper is at https://dl.acm.org/doi/pdf/10.1145/3201603. "We have considered two different evaluation measures: Precision at 1 (P@1) measures the percentage of nodes for which the category with the highest predicted score is among the observed labels for this node. For monolabel classification, this should be the target label, while for multilabel classification, this could be any of the target labels. Precision at k (P@k) is the proportion of correct labels in the set of k labels with the highest predicted scores. For the monolabel dataset DBLP, we only make use of Precision at 1 (P@1). For the multilabels dataset, P@k will denote an average over all the node types, with k set to the number of categories a node belongs to. We optimized the different models with regard to microaverage, but we report both microaverage and macroaverage precision P@•"

33. Sun et al. (2021): The paper is at https://link.springer.com/chapter/10.1007/978-981-15-8411-4_219. "The prediction step is the same as the experimental procedure in NetMF (Qiu et al., 2018). We divide the datasets randomly into training and testing parts. The training ratio ranges from 10 to 90%. The rest data is the testing part. The multi-label classification task is implemented by one-vs-rest logistic regression model. The result of the testing step is a list of labels' ranking, instead of a fixed label. The number of labels is given in the experiment. The experiments repeat ten times and compute the average of macro-F1 and micro-F1 score, respectively as evaluation."

5.2 Papers Using Number of Labels in Their Code

34. Tang et al. (2015): The paper is at https://dl.acm.org/doi/abs/10.1145/2736277.2741093. The code is at https://github.com/tangjianpku/LINE. Predictions were done at linux/evaluate/program/score.cpp. The following segment shows that the number of labels is used.

```
FILE *fi = fopen(candidate_file, "rb");
while (fscanf(fi, "%d", &len) == 1)
{
  v_nlabels.push_back(len);
  for (int k = 0; k != len; k++)
  {
    fscanf(fi, "%d", &lb);
    if (lb2id[lb] == 0) lb2id[lb] = ++id_size;
    id = lb2id[lb];
    truth[id].insert(test_size);
  }
  test_size++;
}
fclose(fi);
```

From the above we can see `v_nlabels[k]` represents the number of true labels for instance `k`.

```
for (int k = 0; k != test_size; k++)
{
  fscanf(fi, "%d", &tmp);
  for (int i = 0; i != label_size; i++)
  {
    fscanf(fi, "%lf", &prob);
```

```
      id = pst2id[i];
      ranked_list[i].id = id;
      ranked_list[i].value = prob;
    }
    sort(ranked_list, ranked_list + label_size);
    int n = v_nlabels[k];
    for (int i = 0; i != n; i++)
    {
      id = ranked_list[i].id;
      predict[id].insert(k);
    }
  }
  fclose(fi);
```

After ranking the probability values, they then predict the labels according to the number of true labels, which used ground-truth information. Finally, they used this prediction to evaluate Macro- and Micro-F1 scores.

35. Perozzi et al. (2014): The paper is at https://dl.acm.org/doi/abs/10.1145/2623330.2623732. The code is at https://github.com/phanein/deepwalk. Predictions were done at example_graphs/scoring.py. The following code shows that the number of labels is used.

```
clf = TopKRanker(LogisticRegression())
clf.fit(X_train, y_train_)

# find out how many labels should be predicted
top_k_list = [len(l) for l in y_test]
preds = clf.predict(X_test, top_k_list)

results = {}
averages = ["micro", "macro"]
for average in averages:
  results[average] = f1_score(mlb.fit_transform(y_test),
        mlb.fit_transform(preds),
        average=average)
```

36. Dalmia et al. (2018): The paper is at https://dl.acm.org/doi/abs/10.1145/3184558.3191523. The code is at https://github.com/ganeshjawahar/interpretNode. Predictions were done at downstream/nodeclass/scoring.py, which used the same way as Perozzi et al. (2014).

```
clf = TopKRanker(LogisticRegression())
clf.fit(X_train, y_train)

# find out how many labels should be predicted
top_k_list = [len(l) for l in y_test]
preds = clf.predict(X_test, top_k_list)
```

37. Khosla et al. (2020): The paper is at https://link.springer.com/chapter/10.1007/978-3-030-46150-8_24. The code is at https://git.l3s.uni-hannover.de/khosla/nerd. For evaluation, their predictions used the number of true labels, which can be found in evaluation/functions.py. The following segment shows that the number of labels is used.

```
def __get_f1(predictions, y, number_of_labels):
  # find the indices (labels) with the highest probabilities (ascending order)
  pred_sorted = numpy.argsort(predictions, axis=1)
```

```
  # the true number of labels for each node
  num_labels = numpy.sum(y, axis=1)

  # we take the best k label predictions for all nodes, where k is the true number of
      labels
  pred_reshaped = []
  for pr, num in zip(pred_sorted, num_labels):
    pred_reshaped.append(pr[-num:].tolist())

  # convert back to binary vectors
  pred_transformed = MultiLabelBinarizer(range(number_of_labels)).fit_transform(
     pred_reshaped)
  f1_micro = f1_score(y, pred_transformed, average='micro')
  f1_macro = f1_score(y, pred_transformed, average='macro')
  return f1_micro, f1_macro
```

38. Rahman et al. (2020): The paper is at https://ieeexplore.ieee.org/abstract/document/9338414. The code is at https://github.com/HipGraph/Force2Vec. When computing node classification scores, a similar OneVsRestClassifier class is defined as in Perozzi et al. (2014), which takes in the number of labels for each entry and subsequently uses it for prediction.

```
#this class is defined similar to deepwalk paper.
class MyClass(OneVsRestClassifier):
  def prediction(self, X, nclasses):
    ps = np.asarray(super(MyClass, self).predict_proba(X))
    #print(ps)
    predlabels = []
    for i, k in enumerate(nclasses):
      ps_ = ps[i, :]
      labels = self.classes_[ps_.argsort()[-k:]].tolist()
      predlabels.append(labels)
    return predlabels
modelLR = MyClass(LogisticRegression(random_state=0)).fit(trainX, trainY)
ncs = [len(x) for x in testY]
predictedY = modelLR.prediction(testX, ncs)
```

39. Liang et al. (2021): The paper is at https://ojs.aaai.org/index.php/ICWSM/article/view/18067. The code is at https://github.com/jiongqian/MILE. In eval_embed.py, the multilabel classifier function uses the same strategy as Perozzi et al. (2014).

```
clf = OneVsRestClassifier(LogisticRegression(), n_jobs=12) # for multilabel scenario.
    #penalty='l2'
clf.fit(X_train, y_train)
y_pred_proba = clf.predict_proba(X_test)
y_pred = []
for inst in range(len(X_test)):
  # assume it has the same number of labels as the truth. Same strtegy is used in
     DeepWalk and Node2Vec paper.
  y_pred.append(y_pred_proba[inst, :].argsort()[::-1][:sum(y_test[inst, :])])
```

40. Cui et al. (2020): The paper is at https://ieeexplore.ieee.org/abstract/document/9377928. The code is at https://github.com/7733com/MLANE. Again, as seen in src/utils.py, a OneVsRestClassifier is used to make predictions taking into account a `top_k_list` containing the number of labels for each entry.

```
top_k_list = [len(l) for l in Y]
Y_ = self.predict(X, top_k_list)
```

The following four papers (41, 42, 43, and 44) all contain the exact two lines of code as the above in their respective scoring or evaluation functions, only differing in the variable names used:

41. Zhuo et al. (2019): The paper is at https://www.hindawi.com/journals/cin/2019/8106073/. The code is at https://github.com/JhuoW/CAHNE and evaluation done in classify.py.

42. Zhang et al. (2019a): The paper is at https://www.ijcai.org/proceedings/2019/594. The code is at https://github.com/THUDM/ProNE and evaluation done in classifier.py.

43. Yang et al. (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3292500.3330951. The code is at https://github.com/eXascaleInfolab/NodeSketch and evaluation done in scoring_kernel_hamming.py.

44. Akbas and Aktas (2019): The paper is at https://ieeexplore.ieee.org/abstract/document/9006142. The code is at https://github.com/esraabil/NECL and evaluation done in classify.py.

45. Wang et al. (2020b): The paper is at https://dl.acm.org/doi/abs/10.1145/3340531.3412041. The code is at https://github.com/SoftWiser-group/RankNE/. scoring.py's evaluation function contains the following, which follows the trend from the above papers:
```
# find out how many labels should be predicted
top_k_list = [np.sum(y_test_[i]) for i in range(y_test_.shape[0])]
# print('top_k_list', top_k_list[:10])
preds = clf.predict(X_test, top_k_list)
```

46. Li et al. (2021): The paper is at https://www.jair.org/index.php/jair/article/view/12567. The code is at https://github.com/RingBDStack/RWNE/. code/eval_classify.py's _classify_thread_body function contains the following, almost identical to the above:
```
# find out how many labels should be predicted
top_k_list = [np.sum(Y_test[i]) for i in range(np.size(Y_test,axis=0))]
clf = TopKRanker(LogisticRegression())
clf.fit(X_train, Y_train)
preds = clf.predict(X_test, top_k_list)
```

47. Xiao et al. (2020): The paper is at https://epubs.siam.org/doi/abs/10.1137/1.9781611976236.67. The code is at https://github.com/HKUST-KnowComp/vertex-reinforced-random-walk. Evaluation is done inside cogdl/tasks/unsupervised_node_classification.py's _evaluate function:
```
# find out how many labels should be predicted
top_k_list = list(map(int, y_test.sum(axis=1).T.tolist()[0]))
preds = clf.predict(X_test, top_k_list)
```

48. Cen et al. (2021): The paper is at https://arxiv.org/abs/2103.00959. The code is at https://github.com/thudm/cogdl. Predictions were done at cogdl/tasks/unsupervised_node_classification.py. The following code shows that the number of labels is used in prediction function.
```
class TopKRanker(OneVsRestClassifier):
    def predict(self, X, top_k_list):
        assert X.shape[0] == len(top_k_list)
        probs = np.asarray(super(TopKRanker, self).predict_proba(X))
        all_labels = sp.lil_matrix(probs.shape)
```

```
        for i, k in enumerate(top_k_list):
            probs_ = probs[i, :]
            labels = self.classes_[probs_.argsort()[-k:]].tolist()
            for label in labels:
                all_labels[i, label] = 1
        return all_labels
```

From the code below we can see the number of labels is passed into the prediction function above.

```
clf = TopKRanker(LogisticRegression(solver="liblinear"))
clf.fit(X_train, y_train)

# find out how many labels should be predicted
top_k_list = list(map(int, y_test.sum(axis=1).T.tolist()[0]))
preds = clf.predict(X_test, top_k_list)
result = f1_score(y_test, preds, average="micro")
all_results[train_percent].append(result)
```

49. Liu et al. (2020b): The paper is at https://dl.acm.org/doi/abs/10.1145/3340531.3411910. The code is at https://github.com/smufang/meta-tail2vec/. The following can be found in multilabel_task.py:

```
# find out how many labels should be predicted, same as deepwalk
top_k_list = [len(l) for l in y_test__]
preds = clf.predict(X_test, top_k_list)
```

50. Chanpuriya et al. (2021): The paper is at http://proceedings.mlr.press/v139/chanpuriya21a/chanpuriya21a.pdf. The code is at https://github.com/konsotirop/Invert_Embeddings. In predict.py, the function construct_indicator utilizes the number of labels to generate predictions:

```
def construct_indicator(y_score, y):
    # rank the labels by the scores directly
    num_label = np.sum(y, axis=1, dtype=np.int)
    y_sort = np.fliplr(np.argsort(y_score, axis=1))
    y_pred = np.zeros_like(y, dtype=np.int)
    for i in range(y.shape[0]):
#         print(type(i), num_label.shape, num_label[i])
        for j in range(num_label[i]):
            y_pred[i, y_sort[i, j]] = 1
    return y_pred
```

y_pred returned by the above is then used to calculated Micro- and Macro-f1 scores in the main predict_cv function.

```
y_score = clf.predict_proba(X_test)
y_pred = construct_indicator(y_score, y_test)
mi = f1_score(y_test, y_pred, average="micro")
ma = f1_score(y_test, y_pred, average="macro")
```

51. Lutov et al. (2019): The paper is at https://ieeexplore.ieee.org/abstract/document/9006038. The code is at https://github.com/eXascaleInfolab/GraphEmbEval. This evaluation framework similarly uses a scoring_classif.py that includes a TopKRanker class predicting labels according to known numbers.

```
class TopKRanker(OneVsRestClassifier):
    def predict(self, gram_test, top_k_list):
        assert gram_test.shape[0] == len(top_k_list)
        probs = super(TopKRanker, self).predict_proba(gram_test)
        if not isinstance(probs, np.ndarray):
            probs = probs.toarray()
```

```
        all_labels = []
        for i, k in enumerate(top_k_list):
            probs_ = probs[i]
            # Fetch test labels
            labels = self.classes_[probs_.argsort()[-k:]].tolist()
            all_labels.append(labels)
        return all_labels
```

52. Chen et al. (2018): The paper is at https://ojs.aaai.org/index.php/AAAI/article/view/11849. The code is at https://github.com/GTmac/HARP. The scoring function inside src/scoring.py contains the following:
```
# find out how many labels should be predicted
top_k_list = [len(l) for l in y_test]
preds = clf.predict(X_test, top_k_list)
```

53. Yang et al. (2017): The paper is at https://www.ijcai.org/proceedings/2017/544. The code is at https://github.com/thunlp/NEU. The main program neu.m calls the evaluate function and by default generates prediction through evaluation by ranking via instance, shown below:
```
function [pred] = evaluate_by_ranking_via_instance(pred, Y)
  % rank the labels by the scores directly
  [n, k] = size(Y);
  [val, index] = sort(full(pred), 2, 'descend');

  numlabel = sum(Y,2); % the number of labels for each instance
  clear val;

  pred = construct_indicator(index, numlabel);
  pred = sparse(pred);
```
In addition, the main evaluation function contains the following comment:
```
% suppose we know the number of labels for  ground truth
```

54. Tsitsulin et al. (2021): The paper is at https://dl.acm.org/doi/abs/10.14778/3447689.3447713. The code is at https://github.com/xgfs/FREDE. DeepWalk's performance in the paper is evaluated using the known number of labels, as seen inside src/classification.py's evaluate_deepwalk function:
```
clf = TopKRanker(LogisticRegression(solver='liblinear'))
clf.fit(X_train, Y_train)
top_k_list = [l.nnz for l in Y_test]
preds = clf.predict(X_test, top_k_list)
```

55. Liu et al. (2021): The paper is at https://ieeexplore.ieee.org/abstract/document/9380483. The code is at https://github.com/Qidong-Liu/TriATNE. Inside eval/functions.py, the __get_f1 function uses the true number of labels:
```
# the true number of labels for each node
num_labels = numpy.sum(y, axis=1)

# we take the best k label predictions for all nodes, where k is the true number of
    labels
pred_reshaped = []
pred_set = set()
for pr, num in zip(pred_sorted, num_labels):
    pred_reshaped.append(pr[-num:].tolist())
    pred_set.update(pr[-num:])
```

56. Sheikh et al. (2019): The paper is at https://link.springer.com/article/10.1007/s00607-018-0622-9. The code is at https://github.com/snash4/GAT2VEC. Inside src/GAT2VEC/evaluation/classification.py, the following function explicitly uses label information:

```
def fit_and_predict_multilabel(self, clf, X_train, X_test, y_train, y_test):
    """ predicts and returns the top k labels for multi-label classification
    k depends on the number of labels in y_test."""
    clf.fit(X_train, y_train)
    y_pred_probs = clf.predict_proba(X_test)

    pred_labels = []
    nclasses = y_test.shape[1]
    top_k_labels = [np.nonzero(label)[0].tolist() for label in y_test]
    for i in range(len(y_test)):
        k = len(top_k_labels[i])
        probs_ = y_pred_probs[i, :]
        labels_ = tuple(np.argsort(probs_).tolist()[-k:])
        pred_labels.append(labels_)
```

57. Mitra et al. (2020): The paper is at https://epubs.siam.org/doi/abs/10.1137/1.9781611976236.55. The code is at https://github.com/sonaidgr8/USS_NMF. Inside main_algo.py, the construct_indicator function is used to generate predictions by ranking labels' scores:

```
def construct_indicator(y_score, y):
    # rank the labels by the scores directly
    num_label = np.sum(y, axis=1, dtype=np.int)
    y_sort = np.fliplr(np.argsort(y_score, axis=1))
    y_pred = np.zeros_like(y, dtype=np.int)
    for i in range(y.shape[0]):
        for j in range(num_label[i]):
            y_pred[i, y_sort[i, j]] = 1
    return y_pred
```

The above is used in the main function as follows:

```
y_pred = construct_indicator(predictions, labels[pred_ids, :])
```

58. Zhu et al. (2021): The paper is at https://epubs.siam.org/doi/abs/10.1137/1.9781611976700.19. The code is at https://github.com/GemsLab/PhUSION. Inside src/eval/predict.py, the exact same evaluation structure is followed as in the previous paper, differing only in variable names used.

59. Liu et al. (2020a): The paper is at https://ieeexplore.ieee.org/abstract/document/9121704. The code is at https://github.com/Qidong-Liu/ANE. In eval/functions.py, the __get_f1 function explicitly uses the true number of labels in prediction:

```
# the true number of labels for each node
num_labels = numpy.sum(y, axis=1)

# we take the best k label predictions for all nodes, where k is the true number of
   labels
pred_reshaped = []
pred_set = set()
for pr, num in zip(pred_sorted, num_labels):
    pred_reshaped.append(pr[-num:].tolist())
    pred_set.update(pr[-num:])
```

60. Chen et al. (2017): The paper is at https://arxiv.org/abs/1702.05764. The code is downloaded from https://users.ece.cmu.edu/~sihengc/code.zip. In `scoring.py`, the same strategy is again used as in Perozzi et al. (2014), with `top_k` set as true in all evaluations done in `run.sh`.

```
if topk:
    # find out how many labels should be predicted
    top_k_list = [len(l) for l in y_test]
    preds = clf.predict(X_test, top_k_list)
    preds = label2onehot(preds, labels_matrix.toarray().shape[1])
else:
    preds = clf.predict(X_test)
```

61. Zhang and Xu (2019): The paper is at https://arxiv.org/abs/1912.00303. The code is at https://github.com/hanzh015/MANELA. To score node classification, the scoring.py program is modified from the original DeepWalk paper, using the number of labels in prediction:

```
# find out how many labels should be predicted
top_k_list = [len(l) for l in y_test]
preds = clf.predict(X_test, top_k_list)
```

62. Schumacher et al. (2020): The paper is at https://arxiv.org/abs/2005.10039. The code is at https://github.com/SGDE2020/embedding_stability. Inside downstream_classification/classify.py, regardless of the classifier used, multilabel predictions for both training and testing sets are done by first ranking probabilities in descending order and then selecting the top k, where k is equal to the number of true labels. The following is from the get_predictions function, where k is np.count_nonzero(labels).

```
if classification_type == ClassificationType.MULTILABEL:
    if isinstance(train_proba, list):
        train_proba = list(true_prop(train_proba))
        train_proba = np.array(train_proba).T
        test_proba = list(true_prop(test_proba))
        test_proba = np.array(test_proba).T

    target_classes = node_labels_train.shape[-1]
    train_pred = np.zeros((node_labels_train.shape[0], target_classes))
    test_pred = np.zeros((node_labels_test.shape[0], target_classes))

    for i, labels in enumerate(node_labels_train):
        train_pred[i][np.argsort(-train_proba[i])[0:np.count_nonzero(labels)]] = 1
    for i, labels in enumerate(node_labels_test):
        test_pred[i][np.argsort(-test_proba[i])[0:np.count_nonzero(labels)]] = 1
```

63. Qiu et al. (2021): The paper is at https://dl.acm.org/doi/abs/10.1145/3448016.3457329. The code is at https://github.com/xptree/LightNE. In LightNE/predict.py, the function below shows that the number of labels is used to construct prediction results.

```
def construct_indicator(y_score, y):
    # rank the labels by the scores directly
    num_label = y.sum(axis=1, dtype=np.int32)
    # num_label = np.sum(y, axis=1, dtype=np.int)
    y_sort = np.fliplr(np.argsort(y_score, axis=1))
    #y_pred = np.zeros_like(y_score, dtype=np.int32)
    row, col = [], []
    for i in range(y_score.shape[0]):
        row += [i] * num_label[i, 0]
        col += y_sort[i, :num_label[i, 0]].tolist()
```

```
        #for j in range(num_label[i, 0]):
        #    y_pred[i, y_sort[i, j]] = 1
    y_pred = sp.csr_matrix(
            ([1] * len(row), (row, col)),
            shape=y.shape, dtype=np.bool_)
    return y_pred
```

The code below shows that the above function is used when evaluating multi-label classification results.

```
y_score = clf.predict_proba(X_test)
y_pred = construct_indicator(y_score, y_test)
mi = f1_score(y_test, y_pred, average="micro")
ma = f1_score(y_test, y_pred, average="macro")
```

64. Cheng et al. (2019): The paper is at https://link.springer.com/article/10.1007/s11390-019-1934-8. The code is at https://github.com/daweicheng/BHONEM. In method/classify.py, the function below shows that the classifier takes number of labels as argument.

```
class TopKRanker(OneVsRestClassifier):
    def predict(self, X, top_k_list):
        probs = numpy.asarray(super(TopKRanker, self).predict_proba(X))
        all_labels = []
        for i, k in enumerate(top_k_list):
            probs_ = probs[i, :]
            labels = self.classes_[probs_.argsort()[-k:]].tolist()
            probs_[:] = 0
            probs_[labels] = 1
            all_labels.append(probs_)
        return numpy.asarray(all_labels)
```

The code below shows that prediction function takes number of labels as input.

```
        top_k_list = [len(l) for l in Y]
        Y_ = self.predict(X, top_k_list)
```

65. Liu et al. (2018): The paper is at http://www.mlgworkshop.org/2018/papers/MLG2018_paper_5.pdf. The code is at https://github.com/uestcnlp/GEN. The code below shows that the number of labels is used to construct new prediction results.

```
predlist = softmax[i][0]
top_k = len(y_test[h])
top_k_list = heapq.nlargest(top_k, range(len(predlist)), predlist.__getitem__)
top_k_list = [j for j in top_k_list]
preds[h] = top_k_list
```

66. He et al. (2019): The paper is at https://dl.acm.org/doi/pdf/10.1145/3357384.3358061. The code is at https://github.com/HKUST-KnowComp/HeteSpaceyWalk. The code below shows that the number of labels is used to reconstruct prediction results. Inside eval_classify.py, the TopKRanker class used the number of labels as arguments and returned the same number of prediction result for each entry.

```
# find out how many labels should be predicted
top_k_list = [np.sum(Y_test[i]) for i in range(np.size(Y_test,axis=0))]
clf = TopKRanker(LogisticRegression())
clf.fit(X_train, Y_train)
preds = clf.predict(X_test, top_k_list)
```

67. Gu et al. (2020): The paper is at https://proceedings.neurips.cc/paper/2020/file/8b5c8441a8ff8e151b191c53c1842a38-Paper.pdf. The code is at https://github.com/SwiftieH/IGNN/tree/main/nodeclassification.

From the Evaluation function inside utils.py, the following code shows that the number of labels is used to select top k prediction results as the final prediction to evaluate, and calculates Micro-F1 and Macro-F1 scores under this setting.

```
for i in range(preds.shape[0]):
    k = labels[i].sum().astype('int')
    topk_idx = preds[i].argsort()[-k:]
    binary_pred[i][topk_idx] = 1
    for pos in list(labels[i].nonzero()[0]):
        if labels[i][pos] and labels[i][pos] == binary_pred[i][pos]:
            num_correct += 1
```

68. Gurukar et al. (2019): The paper is at https://arxiv.org/pdf/1905.00987.pdf. The code is at https://github.com/PriyeshV/NRL_Benchmark. The code below shows that the number of labels is used to select top k prediction results as the final prediction to evaluate.

```
def predict_top_k(classifier, X, top_k_list):
    assert X.shape[0] == len(top_k_list)
    probs = numpy.asarray(classifier.predict_proba(X))
    all_labels = []
    for i, k in enumerate(top_k_list):
        probs_ = probs[i, :]
        try:
            labels = classifier.classes_[probs_.argsort()[-k:]].tolist()
        except AttributeError:  # for eigenpro
            labels = probs_.argsort()[-k:].tolist()
        all_labels.append(labels)
    return all_label
```

The code below shows that the number of labels is passed into the function above.

```
def get_classifer_performace(classifer, X_test, y_test, multi_label_binarizer):
    top_k_list_test = [len(l) for l in y_test]
    y_test_pred = predict_top_k(classifer, X_test, top_k_list_test)
```

69. Mitra et al.: The paper is at https://dl.acm.org/doi/10.1145/3447548.3467443. The code is at https://github.com/anasuamitra/ssdcm. Inside evaluation/evaluate_ml.py, the function construct_indicator shows that the prediction were done by utilizing the number of ground truth labels:

```
def construct_indicator(y_score, y):
    # rank the labels by the scores directly
    num_label = np.sum(y, axis=1, dtype=np.int)
    y_sort = np.fliplr(np.argsort(y_score, axis=1))
    y_pred = np.zeros_like(y, dtype=np.int)
    for i in range(y.shape[0]):
        for j in range(num_label[i]):
            y_pred[i, y_sort[i, j]] = 1
    return y_pred
```

This prediction result then be used for calculating Macro- and Micro-F1 scores:

```
# test
logits = sup[idx_test] #log(test_embs)
preds = construct_indicator(logits.detach().cpu().numpy(), test_lbls)

test_acc = np.sum(preds == test_lbls) / test_lbls.shape[0]
test_f1_macro = f1_score(test_lbls, preds, average='macro')
test_f1_micro = f1_score(test_lbls, preds, average='micro')
```

5.3 Open-Source Libraries Using Number of Labels in Their Code

The following network embedding libraries have no corresponding papers but similarly use the number of labels in prediction.

70. The github is at https://github.com/thunlp/OpenNE and contains network embedding models and evaluations for the following: DeepWalk, Node2vec, LINE, GraRep, TADW, GCN, GraphFactorization, SDNE. Except for GCN, all of the above models use the following code inside src/openne/classify.py for evaluation.
```
def evaluate(self, X, Y):
    top_k_list = [len(l) for l in Y]
    Y_ = self.predict(X, top_k_list)
```

71. The github is at https://github.com/shenweichen/GraphEmbedding and contains network embedding models and evaluations for the following: DeepWalk, LINE, Node2vec, SDNE, Struc2Vec. ge/classify.py contains the same evaluation code as in OpenNE.

5.4 Papers Citing OpenNE in Evaluation

The following papers neither explicitly state the use of unrealistic information during prediction in their papers nor have publicly available code, but implicitly admit to following this practice by citing OpenNE from section 5.3 in their experimental procedures. Papers that do not mention specifically how evaluations were done are included in this section as well - we only suspect they may follow the same practice since the OpenNE framework combines training and evaluation into one command. If possible, we also take into consideration the experimental values and how these values compare to our results for the same method and data set under the unrealistic prediction.

72. Lozano et al. (2021): The paper is at https://www.sciencedirect.com/science/article/pii/S0167865521000027. "These experiments are performed by training a logistic regression classifier with the embedding vectors corresponding to 50% of the nodes, and tested with the remaining vectors, using the OpenNE framework."

73. Chandra et al. (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3347146.3359104. "We evaluate our proposed models against the baseline models on classification and clustering data mining tasks. We use OpenNE toolkit to implement DeepWalk, LINE, GraRep, and node2vec models...We evaluate the performances of the models by calculating Macro-F1 and Micro-F1 scores."

74. Li et al. (2019a): The paper is at https://dl.acm.org/doi/pdf/10.1145/3341161.3342864. "The baseline methods of GraRep, Deepwalk, and LINE are implemented in the OpenNE toolkit, and we follow their default parameter settings." Their results also seem close to unrealistic i.e. BlogCatalog's DeepWalk at 80% training ratio has Micro-F1: 0.4051 and Macro-F1: 0.2701 (compared to Micro-F1: 0.417 and Macro-F1: 0.276 in ours).

5.5 Papers with Abnormally High Macro-F1 and/or Micro-F1 Results

We list papers reporting Macro-F1 and/or Micro-F1 scores that are abnormally high or comparable to our results under unrealistic predictions. For papers with results that are slightly lower than ours, we suspect them on the basis of citing papers in section 5.2 for their experimental procedures or describing a similar procedure with results that are higher than what would likely be achieved without using the number of labels.

75. Hou et al. (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3292500.3330948. For PPI data, this paper "use about 80% of the nodes (i.e., those in 22 subgraphs) for training and nodes in the remaining 2 subgraphs for validation and test". On Node2vec method, their Micro-F1 is 0.61, which is consistent with our unrealistic prediction value (i.e., 0.62), while their Macro-F1 is 0.48 which even better than ours (i.e., 0.44). Though on DeepWalk method, their Micro-F1 and Macro-F1 are respectively 0.60 and 0.45, which are slightly lower than our DeepWalk unrealistic prediction (i.e., 0.64 and 0.48). Again on DeepWalk method, if we consider our highest downstream method thresholding, then for Macro-F1, their value is slightly lower than ours (i.e., 0.48). However, for Micro-F1, their value is much higher than ours (i.e., 0.53).

76. Xin et al. (2019): The paper is at https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8850094. "We evaluated our method using the same experimental procedure as other research (Perozzi et al., 2014)...We trained a one-vs-rest logistic regression model with L2 regularization on the network embeddings for prediction." Their results seem close to unrealistic i.e. BlogCatalog's DeepWalk at 80% training ratio has Macro-F1: 0.253 (compared to Macro-F1: 0.276 in ours).

77. Zhang et al. (2021): The paper is at https://arxiv.org/abs/2106.05476. "Following previous work (Grover and Leskovec, 2016; Perozzi et al., 2014), we employ a one-vs-rest logistic regression classifier implemented by LIBLINEAR [15] with default parameters for all methods. Micro-F1 score is used as the evaluation metric for the classification task." Their results seem close to unrealistic i.e. BlogCatalog's DeepWalk at 70% training ratio has Micro-F1 0.399 and at 90% training ratio has Micro-F1 0.413 (compared to Micro-F1: 0.417 in ours at 80% training ratio).

78. Zhu et al. (2018): The paper is at https://dl.acm.org/doi/10.1145/3219819.3220052. "Then, following (Perozzi et al., 2014), we randomly sample a portion of the labeled nodes as the training data and the rest as the test. For BlogCatalog, we randomly sample 10% to 90% of the nodes as the training samples and use the left nodes to test the performance." Their results are shown graphically and seem slightly lower than unrealistic but still comparable i.e. BlogCatalog's DeepWalk at 80% training ratio has Micro-F1 around 0.375 and Macro-F1 around 0.250.

79. García-Durán and Niepert (2017): The paper is at https://dl.acm.org/doi/10.5555/3295222.3295265. "For the graphs without attributes (BlogCatalog, PPI and POS) we follow the exact same experimental procedure as in previous work (Tang and Liu, 2009a; Perozzi et al., 2014; Grover and Leskovec, 2016). First, the node embeddings were computed in an unsupervised fashion. Second, we sampled a fraction Tr of nodes uniformly at random and used their embeddings and class labels as training data for a logistic regression classifier. " Their Micro-F1 results for both DeepWalk and Node2vec seem close to unrealistic but Macro-F1 is slightly lower i.e. for BlogCatalog, DeepWalk at 90% training ratio has Micro-F1: 0.383 ± 0.018 and Macro-F1: 0.229 ± 0.010; Node2vec at 90% training ratio has Micro-F1: 0.400 ± 0.012 and Macro-F1: 0.248 ± 0.010 (For Node2vec, our results at 80% training ratio were Micro-F1: 0.426 ± 0.005 and Macro-F1: 0.294 ± 0.004).

80. Perozzi et al. (2017): The paper is at https://dl.acm.org/doi/10.1145/3110025.3110086. "We evaluate our method using the same experimental procedure outlined in Perozzi et al. (2014)." Their DeepWalk results seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 90% training ratio has Micro-F1: 0.423 (their LINE results are not comparable as they use the LINE 1st method whereas we use LINE 1st+2nd).

81. Cavallari et al. (2017): The paper is at https://dl.acm.org/doi/pdf/10.1145/3132847.3132925. "We follow (Perozzi et al., 2014) to first train graph embedding on the whole graph, then randomly split 10%

(BlogCatalog, Wikipedia and DBLP) and 90% (Flickr) of nodes as test data, respectively." Their DeepWalk Micro-F1 results seem comparable to ours although Macro-F1 is considerably lower i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1: 0.383 and Macro-F1: 0.215.

82. Fang et al. (2021): The paper is at https://ranger.uta.edu/~jiang/publication/Conferences/2021/HuGE_ICDE_publish.pdf. "To perform this task, we use embedding vector and a one-vs-rest logistic regression classifier with L2 regularization (using the LIBLINEAR library), and evaluate accuracy by micro-averaged F1 (Micro-F1) and micro-averaged F1 (Macro-F1)." Their results are shown graphically and Micro-F1 results seem comparable to ours although Macro-F1 is considerably lower i.e. for Flickr, DeepWalk at 80% training ratio has Micro-F1 around 38 and Macro-F1 around 22; For Node2vec at 80% training ratio, Micro-F1 is around 0.390 and Macro-F1 around 0.240 (for Node2vec on Flickr, our results at 80% training ratio were Micro-F1: 0.420 and Macro-F1: 0.306).

83. Zhu and Koniusz (2021): The paper is at https://arxiv.org/pdf/2108.10703.pdf. By using 70% BlogCatalog data set for training and 30% for testing, the paper indicated that DeepWalk had a 0.425 on Micro-F1 score and 0.285 on Macro-F1 score, which is similar or even higher to our DeepWalk results which had a 0.417 on Micro-F1 score and 0.279 on Macro-F1 score under 80% of training data with unrealistic prediction setting.

84. Yang et al. (2018): The paper is at https://crad.ict.ac.cn/EN/10.7544/issn1000-1239.2018.20180248. The paper did not clearly show their evaluation process; however, due to experimental results similar to our unrealistic evaluation, we suspected that the number of labels was used. By using 80% of training data in BlogCatalog, the paper indicated that DeepWalk had a nearly 0.400 on Micro-F1 score and nearly 0.300 on Macro-F1, which are similar to our results, having a 0.417 on Micro-F1 score and 0.279 on Macro-F1 under 80% of training data.

85. Zhang et al. (2018a): The paper is at https://dl.acm.org/doi/abs/10.1145/3269206.3269320. On the BlogCatalog task, the paper scored 0.392, 0.403, 0.370 on Micro-F1 and 0.254, 0.264, 0.219 on Macro-F1 by using DeepWalk, Node2vec, and LINE respectively, while our result under unrealistic setting scores 0.417, 0.426, 0.406 on micro F1 and 0.276, 0.294, 0.239 on macro F1. Although the scores are slightly lower than ours as well as no clear evidence showing the number of labels is used, the paper revealed that they followed the same experimental setting as Goyal and Ferrara (2018), which used the number of labels during evaluation in code.

86. Fahrbach et al. (2020): The paper is at http://proceedings.mlr.press/v119/fahrbach20a/fahrbach20a.pdf. Using 80% of BlogCatalog data set for training, the paper indicated that LINE scored nearly 0.400 on Micro-F1 and nearly 0.200 on Macro-F1, which is similar to our unrealistic evaluation setting on LINE results which were 0.406 on Micro-F1 score and 0.239 on Macro-F1 trained by 80% of the data.

87. Zhang and Xu (2020): The paper is at https://arxiv.org/pdf/2003.02689.pdf. Using 80% of BlogCatalog data set for training, the paper showed the performance of DeepWalk, Node2vec, and LINE are 0.427, 0.413, 0.433 on Micro-F1 score, and 0.287, 0.277, 0.292 on Macro-F1 score, while our results are 0.417, 0.426, 0.406 on Micro-F1 and 0.276, 0.294, 0.239 on Macro-F1 under unrealistic evaluation setting. Since the scores are similar or even higher than our result, we suspect the paper introduced the number of labels in the evaluation process.

88. Zhang et al. (2018e): The paper is at https://ieeexplore.ieee.org/abstract/document/8594903. "We use two measurements, Macro-F1 and Micro-F1 Perozzi et al. (2014), to evaluate the performance. The average

results of 5 runs are reported." Their results are shown graphically and BlogCatalog is the only comparable data set due to the experimental setting. By applying Deewalk and Node2vec, the Micro-F1 scores are above 0.400 and the Macro-F1 scores are about 0.300, while our result under unrealistic setting scores are 0.417, 0.426 on Micro-F1 and 0.276, 0.294 on Macro-F1. Not only the experiments are similar to our unrealistic result, but they also seemed to follow the experiment setting from DeepWalk, thus we suspect the paper used the number of labels during evaluation.

89. Zhang et al. (2019b): The paper is at https://ieeexplore.ieee.org/abstract/document/8923519. "All node embeddings are used to train a one-vs-rest logistic regression classifier with L2 regularization" With the experiment results the paper provided, we can see that by using 80% of BlogCatalog for training, the Macro-F1 score of DeepWalk, Node2vec, and LINE are 0.395, 0.401, 0.360 respectively. On the other hand, our results showed 0.417, 0.426, 0.406 on the Micro-F1 score under the unrealistic setting, which is similar but slightly higher than the paper's results. Since the results introduced by the paper are still significantly higher than the one-vs-rest-basic setting, we suspect that the paper used the number of labels in the evaluation process.

90. Li et al. (2019b): The paper is at https://www.sciencedirect.com/science/article/abs/pii/S0925231219310835. The paper showed experiment results of BlogCatalog and Flickr. By using DeepWalk, Node2vec and LINE and trained with 80% of the BlogCatalog data, the Micro-F1 scores are 0.409, 0.426, 0.434 and Macro-F1 scores are 0.275, 0.303, 0.291 respectively. The results are similar to our experiment under unrealistic setting, which are 0.417, 0.426, 0.406 under Micro-F1 score and 0.276, 0.294, 0.239 under Macro-F1 score. However, the result in Flickr didn't match with our paper; therefore, we are not sure if the paper used the number of labels or not, but we suspect that they used it while evaluating at least BlogCatalog task.

91. Rashed et al. (2019): The paper is at https://arxiv.org/pdf/1902.09294.pdf. "We followed the same experimental protocol in (Krohn-Grimberghe et al., 2012; Perozzi et al., 2014; Grover and Leskovec, 2016). We used 10-fold crossvalidation experiments on each target relation. These experiments were applied using different percentages of labeled nodes ranging from 10% to 90%." Their DeepWalk and Node2vec results are almost identical to unrealistic except for training ratio i.e. for BlogCatalog, DeepWalk at 90% training ratio has Micro-F1: 0.411 and Macro-F1: 0.278. Node2vec at 90% training ratio has Micro-F1: 0.422 and Macro-F1: 0.292.

92. Zheng et al. (2021): The paper is at https://arxiv.org/pdf/2103.07295.pdf. "Following the experiment settings in Grover and Leskovec (2016) for multi-label node classification, we report the performance of each method by increasing the number of nodes labeled for training from 10% to 80% of the total number of nodes...and the other nodes are divided into two sets for verification and testing." Their results are shown graphically and seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1 around 0.40 and Macro-F1 around 0.24.

93. Ma et al. (2019): The paper is at http://proceedings.mlr.press/v97/ma19a/ma19a.pdf. "We follow node2vec Grover and Leskovec (2016) and report the performance of each method while varying the number of nodes labeled for training from $10\%|V|$ to $90\%|V|$,where $|V|$ is the total number of nodes. The rest of the nodes are split equally to form a validation set and a test set." Their results are shown graphically and seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1 around 0.39 and Macro-F1 around 0.24.

94. Xu et al. (2021): The paper is at https://www.mdpi.com/2227-7390/9/15/1767. "We utilize a one-vs-rest logistic regression (Perozzi et al., 2014) to evaluate the learned representational quality of all methods. We randomly sample a certain fraction of the nodes during the training phase as the training set, and the remaining nodes are used for testing." Their results are again shown graphically and seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1 around 0.42 and Macro-F1 around 0.24 (Micro-F1 is even higher than our unrealistic result: 0.417).

95. Wang et al. (2020a): The paper is at https://www.frontiersin.org/articles/10.3389/fnins.2019.01387/full. "In this experiment, Logistic Regression (LR) is used as a classifier. A portion of the labeled nodes are randomly selected as the training dataset, and thus the remaining nodes without labels are adopted to test the performance." Their DeepWalk and Node2vec results seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1: 0.408 and Macro-F1: 0.264. Node2vec at 80% training ratio has Micro-F1: 0.403 and Macro-F1: 0.271.

96. Wu et al. (2021): The paper is at https://arxiv.org/pdf/2101.06471.pdf. "We follow node2vec Grover and Leskovec (2016) and report the performance of each method while varying the number of nodes labeled for training from 10% $|V|$ to 90% $|V|$, where $|V|$ is the total number of nodes. The rest of nodes are split equally to form a validation set and a test set. Then with the best hyperparameters on the validation sets, we report the averaged performance of 30 runs on each multi-label test set." Their results are shown graphically and seemc close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1 around 0.39 and Macro-F1 around 0.24.

97. Cavallari et al. (2019): The paper is at https://ieeexplore.ieee.org/document/8764640?denied=. "We follow Perozzi et al. (2014) to first train graph embedding on the whole graph. Then, we randomly split 80% of the nodes as a training set and the remaining 20% is used for testing. Finally, an SVM classifier [49] is used to infer the node labels. So far, we report the results with the SVM classifier parameter c=1, which was suggested by Perozzi et al. (2014), for all the methods. " Their results seem close to unrealistic i.e. for BlogCatalog, DeepWalk at 80% training ratio has Micro-F1: 0.383 and Macro-F1: 0.222.

98. Chen et al. (2019): The paper is at https://dl.acm.org/doi/abs/10.1145/3357384.3357879. By training with 80% of BlogCatalog data set, the paper indicated that Deepwalk, Node2vec and LINE scored about 0.280, 0.290, 0.280 on Macro-F1 respectively, which is similar to our unrealistic setting on BlogCatalog data set, scoring 0.276, 0.294, 0.239 on Macro-F1 under the same proportion of training data.

99. Zhang et al. (2019c): The paper is at https://www.sciencedirect.com/science/article/abs/pii/S0167739X19300378. By training with 80% of PPI data set, the paper indicated that Deepwalk, Node2vec and LINE scored between 0.580 to 0.620 on Micro-F1, and scored between 0.400 to 0.470 on Macro-F1. The result is similar to our unrealistic evaluation setting on PPI data set, with Micro-F1 ranging from 0.626 to 0.647, and Macro-F1 ranging from 0.442 to 0.504. We can see there's still a small gap between the paper's result and our result; however, any other method without using number of labels will greatly reduce the performance; therefore, we suspect that the paper took the true number of labels into account during the prediction stage.

100. Yao et al. (2021): The paper is at https://arxiv.org/pdf/2010.04895.pdf. By training with 80% of BlogCatalog data set, the paper indicated that Deepwalk and Node2vec scored 0.400, 0.414 on Micro-F1, and scored 0.258, 0.270 on Macro-F1. The result is similar to our unrealistic evaluation setting on BlogCatalog data set, which scored 0.417 to 0.426 on Micro-F1, and scored 0.276 to 0.294 on Macro-F1. Therefore, we suspect that the paper took the true number of labels into account during the prediction stage.

References

Esra Akbas and Mehmet Emin Aktas. Network embedding: on compression and learning. In *2019 IEEE International Conference on Big Data*, pages 4763–4772, 2019.

Dimitris Berberidis and Georgios B. Giannakis. Node embedding via adaptive similarities. In *Proceedings of the 15th International Workshop on Mining and Learning with Graphs*, 2019.

Dimitris Berberidis and Georgios B. Giannakis. Node embedding with adaptive similarities for scalable learning over graphs. *IEEE Transactions on Knowledge and Data Engineering*, 33:637–650, 2021.

Dimitris Berberidis, Athanasios N. Nikolakopoulos, and Georgios B. Giannakis. AdaDIF: Adaptive diffusions for efficient semi-supervised learning over graphs. In *Proceedings of IEEE International Conference on Big Data*, pages 92–99, 2018.

Dimitris Berberidis, Athanasios N. Nikolakopoulos, and Georgios B. Giannakis. Adaptive diffusions for scalable learning over graphs. *IEEE Transactions on Signal Processing*, 67:1307–1321, 2019.

Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen-Chuan Chang, and Erik Cambria. *Learning Community Embedding with Community Detection and Node Embedding on Graphs*, page 377–386. 2017.

Sandro Cavallari, Erik Cambria, Hongyun Cai, Kevin Chen-Chuan Chang, and Vincent W. Zheng. Embedding both finite and infinite communities on graphs. *IEEE Computational Intelligence Magazine*, 14:39–50, 2019.

Yukuo Cen, Zhenyu Hou, Yan Wang, Qibin Chen, Yizhen Luo, Xingcheng Yao, Aohan Zeng, Shiguang Guo, Yang Yang, Peng Zhang, Guohao Dai, Yu Wang, Chang Zhou, Hongxia Yang, and Jie Tang. CogDL: Toolkit for deep learning on graphs, 2021.

Dakshak Keerthi Chandra, Pengyang Wang, Jennifer Leopold, and Yanjie Fu. Collective representation learning on spatiotemporal heterogeneous information networks. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 319–328, 2019.

Sudhanshu Chanpuriya, Cameron Musco, Konstantinos Sotiropoulos, and Charalampos Tsourakakis. Deepwalking backwards: From embeddings back to graphs. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, pages 1473–1483, 2021.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: Hierarchical representation learning for networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Haochen Chen, Syed Fahad Sultan, Yingtao Tian, Muhao Chen, and Steven Skiena. Fast and accurate network embeddings via very sparse random projection. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 399–408, 2019.

Siheng Chen, Sufeng Niu, Leman Akoglu, Jelena Kovačević, and Christos Faloutsos. Fast, warped graph embedding: Unifying framework and one-click algorithm. *arXiv preprint arXiv:1702.05764*, 2017.

Da-Wei Cheng, Yi Tu, Zhen-Wei Ma, Zhi-Bin Niu, and Li-Qing Zhang. BHONEM: Binary high-order network embedding methods for networked-guarantee loans. *Journal of Computer Science and Technology*, 34:657–669, 2019.

Bo-Yu Chu, Chia-Hua Ho, Cheng-Hao Tsai, Chieh-Yen Lin, and Chih-Jen Lin. Warm start for parameter selection of linear classifiers. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2015. URL http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/warm-start/warm-start.pdf.

Chen Cui, Ning Yang, and Philip S. Yu. MLANE: Meta-learning based adaptive network embedding. In *2020 IEEE International Conference on Big Data*, pages 904–909, 2020.

Ayushi Dalmia, Ganesh Jawahar, and Manish Gupta. Towards interpretation of node embeddings. In *Proceedings of the The Web Conference*, page 945–952, 2018. doi: 10.1145/3184558.3191523.

Robin Devooght, Amin Mantrach, Ilkka Kivimäki, Hugues Bersini, Alejandro Jaimes, and Marco Saerens. Random walks based modularity: Application to semi-supervised learning. In *Proceedings of the 23rd International Conference on World Wide Web (WWW)*, pages 213–224, 2014.

Matthew Fahrbach, Gramoz Goranci, Richard Peng, Sushant Sachdeva, and Chi Wang. Faster graph embeddings via coarsening. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pages 2953–2963, 2020.

Rong-En Fan and Chih-Jen Lin. A study on threshold selection for multi-label classification. Technical report, Department of Computer Science, National Taiwan University, 2007.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: a library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. URL http://www.csie.ntu.edu.tw/~cjlin/papers/liblinear.pdf.

Peng Fang, Fang Wang, Zhan Shi, Hong Jiang, Dan Feng, and Lei Yang. HuGE: An entropy-driven approach to efficient and scalable graph embeddings. In *Proceedings of the 37th IEEE International Conference on Data Engineering (ICDE)*, pages 2045–2050, 2021.

Alberto García-Durán and Mathias Niepert. Learning graph representations with embedding propagation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*, pages 5125–5136, 2017.

Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 855–864, 2016. doi: 10.1145/2939672.2939754.

Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. Implicit graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 11984–11995, 2020.

Saket Gurukar, Priyesh Vijayan, Aakash Srinivasan, Goonmeet Bajaj, Chen Cai, Moniba Keymanesh, Saravana Kumar, Pranav Maneriker, Anasua Mitra, Vedang Patel, Balaraman Ravindran, and Srinivasan Parthasarathy. Network representation learning: Consolidation and renewed bearing. *arXiv preprint arXiv:1905.00987*, 2019.

Yu He, Yangqiu Song, Jianxin Li, Cheng Ji, Jian Peng, and Hao Peng. HeteSpaceyWalk: A heterogeneous spacey random walk for heterogeneous information network embedding. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 639–648, 2019.

Yifan Hou, Hongzhi Chen, Changji Li, James Cheng, and Ming-Chang Yang. A representation learning framework for property graphs. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 65–73, 2019.

Megha Khosla, Jurek Leonhardt, Wolfgang Nejdl, and Avishek Anand. Node representation learning for directed graphs. In *Proceedings of the joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD)*, pages 395–411, 2020.

Megha Khosla, Vinay Setty, and Avishek Anand. A comparative study for unsupervised network representation learning. *IEEE Transactions on Knowledge and Data Engineering*, 33(5):1807–1818, 2021. doi: 10.1109/TKDE.2019.2951398.

Artus Krohn-Grimberghe, Lucas Drumond, Christoph Freudenthaler, and Lars Schmidt-Thieme. Multi-relational matrix factorization using bayesian personalized ranking for social network data. In *Proceedings of the fifth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 173–182, 2012.

Jianxin Li, Cheng Ji, Hao Peng, Yu He, Yangqiu Song, Xinmiao Zhang, and Fanzhang Peng. Rwne: A scalable random-walk based network embedding framework with personalized higher-order proximity preserved. *Journal of Artificial Intelligence Research*, 71:237–263, 2021.

Jundong Li, Liang Wu, Ruocheng Guo, Chenghao Liu, and Huan Liu. Multi-level network embedding with boosted low-rank matrix approximation. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 49–56, 2019a.

Juzheng Li, Jun Zhu, and Bo Zhang. Discriminative deep random walk for network classification. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 1004–1013, 2016a. doi: 10.18653/v1/P16-1095.

Le Li, Junyi Xu, Weidong Xiao, and Bin Ge. Behavior based social dimensions extraction for multi-label classification. *PLOS ONE*, 11:1–16, 2016b.

Qi Li, Zehong Cao, Jiang Zhong, and Qing Li. Graph representation learning with encoding edges. *Neurocomputing*, 361:29–39, 2019b.

Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *Proceedings of the International AAAI Conference on Web and Social Media (ICWSM)*, pages 361–372, 2021.

Qiao Liu, Rui Wan, Xiaohui Yang, Yifu Zeng, and Haibin Zhang. Generalized embedding models for knowledge graph mining. In *Proceedings of the 14th International Workshop on Mining and Learning with Graphs (MLG)*, 2018.

Qidong Liu, Xin Zhou, Cheng Long, Jie Zhang, and Mingliang Xu. Learning network representations with different order structural information. *IEEE Transactions on Computational Social Systems*, 7:907–914, 2020a.

Qidong Liu, Cheng Long, Jie Zhang, Mingliang Xu, and Pei Lv. TriATNE: Tripartite adversarial training for network embeddings. *IEEE Transactions on Cybernetics*, 2021.

Yi Liu, Rong Jin, and Liu Yang. Semi-supervised multi-label learning by constrained non-negative matrix factorization. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI)*, pages 421–426, 2006.

Zemin Liu, Wentao Zhang, Yuan Fang, Xinming Zhang, and Steven C.H. Hoi. *Towards Locality-Aware Meta-Learning of Tail Node Embeddings on Networks*, pages 975–984. 2020b.

Miguel Angel Lozano, Francisco Escolano, Manuel Curado, and Edwin R. Hancock. Network embedding from the line graph: Random walkers and boosted classification. *Pattern Recognition Letters*, 143:36–42, 2021.

Artem Lutov, Dingqi Yang, and Philippe Cudré-Mauroux. Bridging the gap between community and node representations: Graph embedding via community detection. In *Proceedings of the 2019 IEEE International Conference on Big Data*, pages 2681–2690, 2019.

Jianxin Ma, Peng Cui, Kun Kuang, Xin Wang, and Wenwu Zhu. Disentangled graph convolutional networks. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 4212–4221, 2019.

Aditya Krishna Menon and Charles Elkan. Predicting labels for dyadic data. *Data Mining and Knowledge Discovery*, 21:327–343, 2010.

Anasua Mitra, Priyesh Vijayan, Ranbir Sanasam, Diganta Goswami, Srinivasan Parthasarathy, and Balaraman Ravindran. Semi-supervised deep learning for multiplex networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1234–1244.

Anasua Mitra, Priyesh Vijayan, Srinivasan Parthasarathy, and Balaraman Ravindran. A unified non-negative matrix factorization framework for semi supervised learning on graphs. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*, pages 487–495, 2020.

Sharad Nandanwar and M. Narasimha Murty. Structural neighborhood based classification of nodes in a network. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1085–1094, 2016.

Sharad Nandanwar and M. Narasimha Murty. Overlap-robust decision boundary learning for within-network classification. *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

Shameem A. Puthiya Parambath, Nicolas Usunier, and Yves Grandvalet. Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems*, volume 27, 2014.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 701–710, 2014. doi: 10.1145/2623330.2623732.

Bryan Perozzi, Vivek Kulkarni, Haochen Chen, and Steven Skiena. Don't walk, skip! online learning of multi-scale network embeddings. In *Proceedings of the IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 258–265, 2017.

Ştefan Postăvaru, Anton Tsitsulin, Filipe Miguel Gonçalves de Almeida, Yingtao Tian, Silvio Lattanzi, and Bryan Perozzi. InstantEmbedding: Efficient local node representations. *arXiv preprint arXiv:2010.06992*, 2020.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and Node2vec. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM)*, page 459–467, 2018. doi: 10.1145/3159652.3159706.

Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Chi Wang, Kuansan Wang, and Jie Tang. NetSMF: Large-scale network embedding as sparse matrix factorization. In *Proceeding of the World Wide Web Conference (WWW)*, 2019.

Jiezhong Qiu, Laxman Dhulipala, Jie Tang, Richard Peng, and Chi Wang. LightNE: A lightweight graph processing system for network embedding. In *Proceedings of the International Conference on Management of Data*, pages 2281–2289, 2021.

Md. Khaledur Rahman, Majedul Haque Sujon, and Ariful Azad. Force2vec: Parallel force-directed graph embedding. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, pages 442–451, 2020.

Ahmed Rashed, Josif Grabocka, and Lars Schmidt-Thieme. Multi-label network classification via weighted personalized factorizations, 2019.

Georgios Rizos, Symeon Papadopoulos, and Yiannis Kompatsiaris. Multilabel user classification using the community structure of online networks. *PLOS ONE*, 12:1–34, 2017.

Ludovic Dos Santos, Benjamin Piwowarski, Ludovic Denoyer, and Patrick Gallinari. Representation learning for classification in heterogeneous graphs with application to social networks. *ACM Transactions on Knowledge Discovery from Data*, 12, 2018.

Jörg Schlötterer, Martin Wehking, Fatemeh Salehi Rizi, and M. Granitzer. Investigating extensions to random walk based graph embedding. In *Proceedings of IEEE International Conference on Cognitive Computing*, pages 81–89, 2019.

Tobias Schumacher, Hinrikus Wolf, Martin Ritzert, Florian Lemmerich, Jan Bachmann, Florian Frantzen, Max Klabunde, Martin Grohe, and Markus Strohmaier. The effects of randomness on the stability of node embeddings. *arXiv preprint arXiv:2005.10039*, 2020.

Nasrullah Sheikh, Zekarias Kefato, and Alberto Montresor. GAT2VEC: representation learning for attributed graphs. *Computing*, 101:187–209, 2019.

Yuchen Sun, Liangtian Wan, Lu Sun, and Xianpeng Wang. Z-netmf: A biased embedding method based on matrix factorization. In *Proceedings of the 9th International Conference on Communications, Signal Processing, and Systems*, pages 1669–1676, 2021.

Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international Conference on World Wide Web (WWW)*, pages 1067–1077, 2015.

Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 817–826, 2009a.

Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM conference on Information and Knowledge Management (CIKM)*, pages 1107–1116, 2009b.

Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23:447–478, 2011.

Lei Tang, Suju Rajan, and Vijay K. Narayanan. Large scale multi-label classification via metalabeler. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 211–220, 2009.

Lei Tang, Xufei Wang, Huan Liu, and Lei Wang. A multi-resolution approach to learning with overlapping communities. In *Proceedings of the First Workshop on Social Media Analytics*, pages 14–22, 2010.

Lei Tang, Xufei Wang, and Huan Liu. Scalable learning of collective behavior. *IEEE Transactions on Knowledge and Data Engineering*, 24:1080–1091, 2012.

Anton Tsitsulin, Marina Munkhoeva, Davide Mottin, Panagiotis Karras, Ivan Oseledets, and Emmanuel Müller. FREDE: Anytime graph embeddings. *Proceedings of the VLDB Endowment*, 14:1102–1110, 2021.

Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Mining multi-label data. *Data Mining and Knowledge Discovery Handbook*, pages 667–685, 2010.

Shanfeng Wang, Qixiang Wang, and Maoguo Gong. Multi-task learning based network embedding. *Frontiers in Neuroscience*, 13:1387, 2020a.

Xi Wang and Gita Sukthankar. Extracting social dimensions using fiedler embedding. In *Proceedings of IEEE Third International Conference on Privacy, Security, Risk and Trust, and IEEE Third International Conference on Social Computing*, pages 824–829, 2011.

Xi Wang and Gita Sukthankar. Multi-label relational neighbor classification using social context features. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 464–472, 2013.

Xufei Wang, Lei Tang, Huan Liu, and Lei Wang. Learning with multi-resolution overlapping communities. *Knowledge and Information Systems*, 36:517–535, 2013.

Yaojing Wang, Guosheng Pan, Yuan Yao, Hanghang Tong, Hongxia Yang, Feng Xu, and Jian Lu. Bringing order to network embedding: A relative ranking based approach. In *Proceedings of the 29th ACM International Conference on Informationand Knowledge Management (CIKM)*, pages 1585–1594, 2020b.

Likang Wu, Zhi Li, Hongke Zhao, Qi Liu, Jun Wang, Mengdi Zhang, and Enhong Chen. Learning the implicit semantic representation on graph-structured data. In *Proceedings of 26th International Conference on Database Systems for Advanced Applications*, 2021.

Wenyi Xiao, Huan Zhao, Vincent Wenchen Zheng, and Yangqiu Song. Vertex-reinforced random walk for network embedding. In *Proceedings of the 2020 SIAM International Conference on Data Mining (SDM)*, pages 595–603, 2020.

Zhenghua Xin, Jie Chen, Guolong Chen, and Shu Zhao. Marc: multi-granular representation learning for networks based on the 3-clique. *IEEE Access*, 7:141715–141727, 2019.

Xin Xu, Yang Lu, Yupeng Zhou, Zhiguo Fu, Yanjie Fu, and Minghao Yin. An information-explainable random walk based unsupervised network representation learning framework on node classification tasks. *Mathematics*, 9(15):1767, 2021.

Cheng Yang, Maosong Sun, Zhiyuan Liu, and Cunchao Tu. Fast network embedding enhancement via high order proximity approximation. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3894–3900, 2017.

Dingqi Yang, Paolo Rosso, Bin Li, and Philippe Cudre-Mauroux. NodeSketch: Highly-efficient graph embeddings via recursive sketching. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1162–1172, 2019.

Xiaohui Yang, Rui Wan, Haibin Zhang, Yifu Zeng, and Qiao Liu. Semantical symbol mapping embedding learning algorithm for knowledge graph. *Journal of Computer Research and Development*, 55:1773–1784, 2018.

Xingyu Yao, Yingxia Shao, Bin Cui, and Lei Chen. Uninet: Scalable network representation learning with metropolis-hastings sampling. In *Proceedings of the 37th IEEE International Conference on Data Engineering (ICDE)*, pages 516–527, 2021.

Qi Ye, Changlei Zhu, Gang Li, Zhimin Liu, and Feng Wang. Using node identifiers and community prior for graph-based classification. *Data Science and Engineering*, 3:68–83, 2018.

Yuan Yin and Zhewei Wei. Scalable graph embeddings via sparse transpose proximities. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, page 1429–1437, 2019. doi: 10.1145/3292500.3330860.

Xiang Yue, Zhen Wang, Jingong Huang, Srinivasan Parthasarathy, Soheil Moosavinasab, Yungui Huang, Simon M Lin, Wen Zhang, Ping Zhang, and Huan Sun. Graph embedding on biomedical networks: methods, applications and evaluations. *Bioinformatics*, 36:1241–1251, 2020.

Han Zhang and Hong Xu. MANELA: A multi-agent algorithm for learning network embeddings. *arXiv preprint arXiv:1912.00303*, 2019.

Jie Zhang, Yuxiao Dong, Yan Wang, Jie Tang, and Ming Ding. ProNE: Fast and scalable network representation learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, pages 4278–4284, 2019a.

Jie Zhang, Yan Wang, and Jie Tang. A fast network embedding approach with preserving hierarchical proximities. In *Proceedings of IEEE Fourth International Conference on Data Science in Cyberspace (DSC)*, pages 142–149, 2019b.

Luoyi Zhang and Ming Xu. EPINE: Enhanced proximity information network embedding. *arXiv preprint arXiv:2003.02689*, 2020.

Xingyi Zhang, Kun Xie, Sibo Wang, and Zengfeng Huang. Learning based proximity matrix factorization for node embedding. *arXiv preprint arXiv:2106.05476*, 2021.

Yi Zhang, Jianguo Lu, and Ofer Shai. Improve network embeddings with regularization. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1643–1646, 2018a.

Yunyi Zhang, Zhan Shi, Dan Feng, and Xiu-Xiu Zhan. Degree-biased random walk for large-scale network embedding. *Future Generation Computer Systems*, 100:198–209, 2019c.

Zan Zhang, Jiuyong Li, Hao Wang, Lin Liu, and Jixue Liu. Which type of classifier to use for networked data, connectivity based or feature based? In *Proceedings of the Web Information Systems Engineering*, pages 364–380, 2018b.

Zan Zhang, Lin Liu, Hao Wang, Jiuyong Li, Daning Hu, Jiaqi Yan, Rene Algesheimer, and Markus Meierer. Collective behavior learning by differentiating personal preference from peer influence. *Knowledge-Based Systems*, 159:233–243, 2018c.

Zan Zhang, Hao Wang, Lin Liu, and Jiuyong Li. Multi-label relational classification via node and label correlation. *Neurocomputing*, 292:72–81, 2018d.

Ziwei Zhang, Peng Cui, Haoyang Li, Xiao Wang, and Wenwu Zhu. Billion-scale network embedding with iterative random projection. In *Proceddings of IEEE International Conference on Data Mining (ICDM)*, pages 787–796, 2018e.

Shuai Zheng, Zhenfeng Zhu, Zhizhe Liu, Shuiwang Ji, Jian Cheng, and Yao Zhao. Adversarial graph disentanglement, 2021.

Dingyuan Zhu, Peng Cui, Daixin Wang, and Wenwu Zhu. Deep variational network embedding in wasserstein space. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 2827–2836, 2018.

Hao Zhu and Piotr Koniusz. Refine: random range finder for network embedding. In *Proceedings of the 30th ACM International Conference on Conference on Information and Knowledge Management (CIKM)*, 2021.

Jing Zhu, Xingyu Lu, Mark Heimann, and Danai Koutra. *Node Proximity Is All You Need: Unified Structural and Positional Node and Graph Embedding*, pages 163–171. 2021.

Wei Zhuo, Qianyi Zhan, Yuan Liu, Zhenping Xie, and Jing Lu. Context attention heterogeneous network embedding. *Computational Intelligence and Neuroscience*, 2019. doi: 10.1155/2019/8106073.