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

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

\textsuperscript{1}National Taiwan University
\textsuperscript{2}National Chengchi University
\textsuperscript{3}University of Southern California
\textsuperscript{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 facilitate the discussion, we consider

\(i\) : index of test instances, and \(j\) : index of labels.

We further assume that for test instance \(i\)

\[
\begin{align*}
K_i : & \text{ true number of labels,} \\
\hat{K}_i : & \text{ predicted number of labels.}
\end{align*}
\]

Define

\[
\text{Micro-F1} = \frac{2 \times \text{TP sum}}{\text{TP sum} + \text{FP sum} + \text{TP sum} + \text{FN sum}},
\]

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

\[
\text{TP}_i + \text{FP}_i = \hat{K}_i \text{ and } \text{TP}_i + \text{FN}_i = K_i.
\]
From (11) and (12),
\[
\text{Micro-F1} = \frac{2 \times \text{TP sum}}{\sum_{i=1}^{l} (K_i + \hat{K}_i)} = \frac{2 \times \sum_{i=1}^{l} \text{TP}_i}{\sum_{i=1}^{l} (K_i + \hat{K}_i)}.
\] (13)

From (12), \(\text{TP}_i \leq \hat{K}_i\) and \(\text{TP}_i \leq K_i\), so the result in (7) follows from
\[
\text{TP}_i \leq \min(\hat{K}_i, K_i).
\] (14)

The second inequality in (7) easily follows from \(\min(\hat{K}_i, K_i) \leq \hat{K}_i \) and \(K_i\). Finally, when \(\hat{K}_i = K_i\),
\[
\frac{2 \times \sum_{i=1}^{l} \min(\hat{K}_i, K_i)}{\sum_{i=1}^{l} (K_i + \hat{K}_i)} = \frac{2 \times \sum_{i=1}^{l} K_i}{\sum_{i=1}^{l} (K_i + K_i)} = 1
\] (15)
achieves the maximum.

\[\square\]

1.2 Proof of Theorem 2

Proof. By the assumption on decision values, selecting the top ones leads to
\[
\text{TP}_i = \begin{cases} 
\hat{K}_i & \text{if } \hat{K}_i \leq K_i, \\
K_i & \text{otherwise}.
\end{cases}
\]

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

\[\square\]

1.3 Proof of Theorem 3

Proof. By properties of multi-class problems mentioned before the theorem statement,
\[
K_i = \hat{K}_i = 1.
\] (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).

\[\square\]
### Table 4: Dataset Statistics

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#nodes</th>
<th>#edges</th>
<th>#degrees per node</th>
<th>Edge Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td>10,312</td>
<td>333,983</td>
<td>64.78</td>
<td>undirected</td>
</tr>
<tr>
<td>Flickr</td>
<td>80,513</td>
<td>5,899,882</td>
<td>146.56</td>
<td>undirected</td>
</tr>
<tr>
<td>YouTube</td>
<td>1,138,499</td>
<td>2,990,443</td>
<td>5.25</td>
<td>undirected</td>
</tr>
<tr>
<td>PPI</td>
<td>56,944</td>
<td>409,358</td>
<td>14.38</td>
<td>undirected</td>
</tr>
</tbody>
</table>

1. [http://leitang.net/social_dimension.html](http://leitang.net/social_dimension.html)

#### 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.
<table>
<thead>
<tr>
<th></th>
<th>Dimension</th>
<th>Window Size</th>
<th>Walk Length</th>
<th>Walk Times</th>
<th>NS or HS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DeepWalk authors</strong></td>
<td>64</td>
<td>5</td>
<td>40</td>
<td>10</td>
<td>HS</td>
</tr>
<tr>
<td>Perozzi et al. (2014)</td>
<td>128</td>
<td>10</td>
<td>-</td>
<td>80</td>
<td>HS</td>
</tr>
<tr>
<td>Tang et al. (2015)</td>
<td>128</td>
<td>10</td>
<td>40</td>
<td>40</td>
<td>HS</td>
</tr>
<tr>
<td>Grover and Leskovec (2016)</td>
<td>128</td>
<td>10</td>
<td>80</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>Khosla et al. (2021)</td>
<td>128</td>
<td>10</td>
<td>40</td>
<td>80</td>
<td>-</td>
</tr>
<tr>
<td>Our Setting</td>
<td>128</td>
<td>10</td>
<td>40</td>
<td>80</td>
<td>HS</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Dimension</th>
<th>Window Size</th>
<th>Walk Length</th>
<th>Walk Times</th>
<th>((p, q))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Node2vec authors</strong></td>
<td>128</td>
<td>10</td>
<td>80</td>
<td>10</td>
<td>((1,1))</td>
</tr>
<tr>
<td>Grover and Leskovec (2016)</td>
<td>128</td>
<td>10</td>
<td>-</td>
<td>80</td>
<td>(p, q \in {0.25, 0.5, 1, 2, 4})</td>
</tr>
<tr>
<td>Khosla et al. (2021)</td>
<td>128</td>
<td>10</td>
<td>40</td>
<td>80</td>
<td>(p, q \in {0.25, 0.5, 1, 2, 4})</td>
</tr>
<tr>
<td>Our Setting</td>
<td>128</td>
<td>10</td>
<td>40</td>
<td>80</td>
<td>((4, 0.25)^2)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Dimension (1st + 2nd)</th>
<th>Sample Times</th>
<th>Negative Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LINE authors</strong></td>
<td>100 + 100</td>
<td>-</td>
<td>5</td>
</tr>
<tr>
<td>Tang et al. (2015)</td>
<td>128 + 128\dagger</td>
<td>10 Billion</td>
<td>5</td>
</tr>
<tr>
<td>Grover and Leskovec (2016)</td>
<td>64 + 64</td>
<td>10\times</td>
<td>|E</td>
</tr>
<tr>
<td>Khosla et al. (2021)</td>
<td>64 + 64</td>
<td>10 Billion</td>
<td>5</td>
</tr>
<tr>
<td>Our Setting</td>
<td>64 + 64</td>
<td>10 Billion</td>
<td>5</td>
</tr>
</tbody>
</table>

1 https://github.com/tangjianpku/LINE This is the source code provided by Tang et al. (2015).
\dagger 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.
\dagger |\|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

$$C < \frac{1}{l \max_i ||\mathbf{x}_i||^2}.$$ 

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

$$10 < \max_i ||\mathbf{x}_i||^2 < 500.$$ 

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

$$\max_i ||\mathbf{x}_i||^2 = 2.$$ 

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. We observe that in most cases there is a marginal improvement with an increase in 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 Table 14, 15 and 16, we compare the Macro-F1, Micro-F1 and Instance-F1 of different number of \(t\) values chosen equidistantly from the range \((0, 1]\) with \(C = 1\). We observe that the Macro-F1 is slightly better with more \(t\) values. However, Micro-F1 and Instance-F1 is usually better with less number of \(t\) values. We will choose to use seven \(t\) values based on a balance of Macro-F1 performance and run time. The justification for only considering Macro-F1 will be given next.

4.3 The Macro-F1 and Micro-F1 Tradeoff

Optimizing Macro-F1 and Micro-F1 is often a tradeoff. Due to the disproportionately large effect of rare labels on Macro-F1, Macro-F1 is optimized by more aggressively predicting positives while Micro-F1 is optimized by less aggressively predicting positives. When less number of \(t\) values is considered, Macro-F1 is effectively less optimized and Micro-F1 will generally be higher.

This can be explicitly observed with another cost-sensitive method optimizing Micro-F1 given by 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, 17, and 19, we compare the Macro-F1, Micro-F1 and Instance-F1 of cost-sensitive-simple and cost-sensitive-simple-micro. We observe that cost-sensitive-simple-micro is significantly better at optimizing Micro-F1, but significantly worse at optimizing Macro-F1. Furthermore, the situation does not change with other choices of number of \(t\) values for cost-sensitive-simple. Therefore, we will only consider the score being optimized for the choice of \(t\) values.

We also observe that Instance-F1 has the same trend as Micro-F1. This is because Instance-F1 behaves similarly as Micro-F1 with respect to the effect 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.

\(^1\)See the main paper about the range of \(t\) values used
<table>
<thead>
<tr>
<th>Training and prediction methods</th>
<th>DeepWalk</th>
<th>Macro-F1 BlogCatalog</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>unrealistic</td>
<td></td>
<td>0.276 ± 0.005</td>
<td>0.294 ± 0.004</td>
<td>0.239 ± 0.008</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.190 ± 0.010</td>
<td>0.203 ± 0.010</td>
<td>0.150 ± 0.003</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.208 ± 0.008</td>
<td>0.220 ± 0.008</td>
<td>0.195 ± 0.007</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td>0.241 ± 0.011</td>
<td>0.265 ± 0.007</td>
<td>0.204 ± 0.011</td>
<td></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.269 ± 0.006</td>
<td>0.283 ± 0.006</td>
<td>0.221 ± 0.006</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive</td>
<td><strong>0.270 ± 0.003</strong></td>
<td><strong>0.283 ± 0.006</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.268 ± 0.004</td>
<td>0.280 ± 0.007</td>
<td><strong>0.251 ± 0.007</strong></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.266 ± 0.005</td>
<td>0.275 ± 0.005</td>
<td>0.251 ± 0.007</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.261 ± 0.006</td>
<td>0.276 ± 0.005</td>
<td>0.251 ± 0.009</td>
<td></td>
</tr>
<tr>
<td>Flickr</td>
<td></td>
<td>0.304 ± 0.003</td>
<td>0.306 ± 0.001</td>
<td>0.258 ± 0.005</td>
</tr>
<tr>
<td>unrealistic</td>
<td></td>
<td>0.195 ± 0.003</td>
<td>0.191 ± 0.002</td>
<td>0.128 ± 0.004</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.209 ± 0.001</td>
<td>0.208 ± 0.002</td>
<td>0.188 ± 0.003</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.256 ± 0.001</td>
<td>0.259 ± 0.002</td>
<td>0.206 ± 0.004</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td></td>
<td><strong>0.299 ± 0.004</strong></td>
<td><strong>0.302 ± 0.002</strong></td>
<td>0.264 ± 0.002</td>
</tr>
<tr>
<td>thresholding</td>
<td>0.297 ± 0.000</td>
<td>0.301 ± 0.002</td>
<td>0.279 ± 0.003</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.298 ± 0.000</td>
<td>0.301 ± 0.002</td>
<td><strong>0.284 ± 0.006</strong></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.294 ± 0.001</td>
<td>0.296 ± 0.002</td>
<td>0.279 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.293 ± 0.001</td>
<td>0.295 ± 0.002</td>
<td>0.279 ± 0.003</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YouTube</td>
<td></td>
<td>0.397 ± 0.007</td>
<td>0.415 ± 0.009</td>
<td>0.412 ± 0.005</td>
</tr>
<tr>
<td>unrealistic</td>
<td></td>
<td>0.213 ± 0.003</td>
<td>0.240 ± 0.006</td>
<td>0.230 ± 0.004</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.217 ± 0.004</td>
<td>0.242 ± 0.006</td>
<td>0.246 ± 0.005</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.263 ± 0.003</td>
<td>0.284 ± 0.006</td>
<td>0.281 ± 0.004</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td></td>
<td>0.358 ± 0.005</td>
<td>0.374 ± 0.006</td>
<td><strong>0.377 ± 0.004</strong></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.360 ± 0.005</td>
<td><strong>0.375 ± 0.003</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive</td>
<td></td>
<td>0.359 ± 0.005</td>
<td>0.374 ± 0.003</td>
<td>0.375 ± 0.004</td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.349 ± 0.005</td>
<td>0.369 ± 0.005</td>
<td>0.371 ± 0.005</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.349 ± 0.005</td>
<td>0.368 ± 0.005</td>
<td>0.372 ± 0.004</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPI</td>
<td></td>
<td>0.483 ± 0.003</td>
<td>0.442 ± 0.003</td>
<td>0.504 ± 0.003</td>
</tr>
<tr>
<td>unrealistic</td>
<td></td>
<td>0.181 ± 0.001</td>
<td>0.148 ± 0.000</td>
<td>0.232 ± 0.002</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.183 ± 0.001</td>
<td>0.150 ± 0.000</td>
<td>0.243 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.181 ± 0.001</td>
<td>0.148 ± 0.000</td>
<td>0.232 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td></td>
<td><strong>0.482 ± 0.002</strong></td>
<td><strong>0.457 ± 0.002</strong></td>
<td><strong>0.498 ± 0.001</strong></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.482 ± 0.002</td>
<td><strong>0.461 ± 0.002</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.482 ± 0.002</td>
<td>0.461 ± 0.002</td>
<td>0.495 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.482 ± 0.002</td>
<td>0.461 ± 0.002</td>
<td>0.495 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.481 ± 0.002</td>
<td>0.460 ± 0.002</td>
<td>0.494 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.481 ± 0.002</td>
<td>0.460 ± 0.002</td>
<td>0.494 ± 0.002</td>
<td></td>
</tr>
</tbody>
</table>

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

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.
<table>
<thead>
<tr>
<th>Training and prediction methods</th>
<th>DeepWalk</th>
<th>Instance-F1</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BlogCatalog</td>
<td></td>
</tr>
<tr>
<td>unrealistic</td>
<td>0.422 ± 0.004</td>
<td>0.433 ± 0.006</td>
<td>0.408 ± 0.005</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.255 ± 0.004</td>
<td>0.265 ± 0.009</td>
<td>0.222 ± 0.004</td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.267 ± 0.003</td>
<td>0.275 ± 0.009</td>
<td>0.256 ± 0.003</td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td><strong>0.403 ± 0.003</strong></td>
<td><strong>0.416 ± 0.006</strong></td>
<td><strong>0.389 ± 0.006</strong></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.347 ± 0.006</td>
<td>0.354 ± 0.012</td>
<td>0.315 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.340 ± 0.006</td>
<td>0.348 ± 0.003</td>
<td>0.314 ± 0.005</td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.352 ± 0.011</td>
<td>0.363 ± 0.011</td>
<td>0.325 ± 0.015</td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.327 ± 0.004</td>
<td>0.335 ± 0.011</td>
<td>0.313 ± 0.006</td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.327 ± 0.003</td>
<td>0.334 ± 0.007</td>
<td>0.311 ± 0.004</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flickr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unrealistic</td>
<td>0.406 ± 0.002</td>
<td>0.410 ± 0.004</td>
<td>0.394 ± 0.004</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.211 ± 0.002</td>
<td>0.212 ± 0.002</td>
<td>0.185 ± 0.002</td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.217 ± 0.001</td>
<td>0.220 ± 0.002</td>
<td>0.207 ± 0.003</td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td><strong>0.386 ± 0.002</strong></td>
<td><strong>0.390 ± 0.004</strong></td>
<td><strong>0.376 ± 0.004</strong></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.352 ± 0.002</td>
<td>0.353 ± 0.005</td>
<td>0.325 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.342 ± 0.001</td>
<td>0.343 ± 0.003</td>
<td>0.326 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.345 ± 0.008</td>
<td>0.357 ± 0.008</td>
<td>0.349 ± 0.018</td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.344 ± 0.006</td>
<td>0.345 ± 0.005</td>
<td>0.329 ± 0.008</td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.346 ± 0.005</td>
<td>0.343 ± 0.007</td>
<td>0.328 ± 0.008</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>YouTube</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unrealistic</td>
<td>0.454 ± 0.007</td>
<td>0.467 ± 0.006</td>
<td>0.464 ± 0.003</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.245 ± 0.009</td>
<td>0.262 ± 0.008</td>
<td>0.254 ± 0.004</td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.247 ± 0.010</td>
<td>0.263 ± 0.008</td>
<td>0.263 ± 0.005</td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td><strong>0.412 ± 0.005</strong></td>
<td><strong>0.427 ± 0.005</strong></td>
<td><strong>0.428 ± 0.003</strong></td>
</tr>
<tr>
<td>thresholding</td>
<td>0.389 ± 0.006</td>
<td>0.400 ± 0.006</td>
<td>0.398 ± 0.006</td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.380 ± 0.011</td>
<td>0.399 ± 0.006</td>
<td>0.394 ± 0.004</td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.388 ± 0.010</td>
<td>0.414 ± 0.005</td>
<td>0.416 ± 0.006</td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.372 ± 0.009</td>
<td>0.388 ± 0.007</td>
<td>0.388 ± 0.004</td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.377 ± 0.012</td>
<td>0.388 ± 0.006</td>
<td>0.388 ± 0.006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PPI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>unrealistic</td>
<td>0.577 ± 0.002</td>
<td>0.565 ± 0.001</td>
<td>0.583 ± 0.002</td>
</tr>
<tr>
<td>one-vs-rest-basic</td>
<td>0.449 ± 0.001</td>
<td>0.438 ± 0.001</td>
<td>0.464 ± 0.001</td>
</tr>
<tr>
<td>one-vs-rest-basic-C</td>
<td>0.457 ± 0.001</td>
<td>0.445 ± 0.002</td>
<td>0.475 ± 0.001</td>
</tr>
<tr>
<td>one-vs-rest-no-empty</td>
<td>0.449 ± 0.001</td>
<td>0.438 ± 0.001</td>
<td>0.464 ± 0.001</td>
</tr>
<tr>
<td>thresholding</td>
<td><strong>0.503 ± 0.002</strong></td>
<td><strong>0.456 ± 0.002</strong></td>
<td><strong>0.517 ± 0.001</strong></td>
</tr>
<tr>
<td>cost-sensitive</td>
<td>0.501 ± 0.002</td>
<td>0.476 ± 0.002</td>
<td>0.513 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive-no-empty</td>
<td>0.501 ± 0.002</td>
<td><strong>0.467 ± 0.002</strong></td>
<td>0.513 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.496 ± 0.002</td>
<td>0.466 ± 0.002</td>
<td>0.512 ± 0.002</td>
</tr>
<tr>
<td>cost-sensitive-simple-rand</td>
<td>0.497 ± 0.002</td>
<td>0.466 ± 0.002</td>
<td>0.513 ± 0.002</td>
</tr>
</tbody>
</table>

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.
### 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.

<table>
<thead>
<tr>
<th>Choice of $C$ values</th>
<th>DeepWalk</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlogCatalog</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.266 ± 0.005</td>
<td>0.275 ± 0.005</td>
<td><strong>0.251 ± 0.007</strong></td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.266 ± 0.005</td>
<td>0.276 ± 0.006</td>
<td>0.249 ± 0.006</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td>0.266 ± 0.004</td>
<td><strong>0.282 ± 0.005</strong></td>
<td>0.250 ± 0.007</td>
</tr>
<tr>
<td><strong>Flickr</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.294 ± 0.001</td>
<td>0.296 ± 0.002</td>
<td>0.279 ± 0.002</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.293 ± 0.001</td>
<td>0.295 ± 0.003</td>
<td>0.278 ± 0.002</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.297 ± 0.002</strong></td>
<td><strong>0.299 ± 0.002</strong></td>
<td><strong>0.279 ± 0.002</strong></td>
</tr>
<tr>
<td><strong>YouTube</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.349 ± 0.005</td>
<td>0.369 ± 0.005</td>
<td>0.371 ± 0.005</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.348 ± 0.005</td>
<td>0.368 ± 0.005</td>
<td>0.371 ± 0.004</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.357 ± 0.004</strong></td>
<td><strong>0.374 ± 0.005</strong></td>
<td><strong>0.374 ± 0.004</strong></td>
</tr>
<tr>
<td><strong>PPI</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.481 ± 0.002</td>
<td>0.460 ± 0.002</td>
<td>0.494 ± 0.002</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.481 ± 0.002</td>
<td>0.460 ± 0.002</td>
<td><strong>0.494 ± 0.002</strong></td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.481 ± 0.002</strong></td>
<td><strong>0.460 ± 0.002</strong></td>
<td>0.494 ± 0.002</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Choice of $C$ values</th>
<th>DeepWalk</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BlogCatalog</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.351 ± 0.006</td>
<td>0.362 ± 0.017</td>
<td>0.337 ± 0.004</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td><strong>0.355 ± 0.006</strong></td>
<td>0.367 ± 0.017</td>
<td>0.336 ± 0.009</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td>0.353 ± 0.010</td>
<td><strong>0.371 ± 0.011</strong></td>
<td><strong>0.342 ± 0.007</strong></td>
</tr>
<tr>
<td><strong>Flickr</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.355 ± 0.003</td>
<td>0.360 ± 0.004</td>
<td>0.357 ± 0.004</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.355 ± 0.003</td>
<td><strong>0.361 ± 0.005</strong></td>
<td><strong>0.360 ± 0.005</strong></td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.358 ± 0.003</strong></td>
<td>0.361 ± 0.003</td>
<td>0.358 ± 0.005</td>
</tr>
<tr>
<td><strong>YouTube</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.365 ± 0.006</td>
<td>0.390 ± 0.007</td>
<td>0.400 ± 0.005</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.365 ± 0.006</td>
<td>0.391 ± 0.008</td>
<td><strong>0.401 ± 0.005</strong></td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.372 ± 0.007</strong></td>
<td><strong>0.395 ± 0.006</strong></td>
<td>0.400 ± 0.004</td>
</tr>
<tr>
<td><strong>PPI</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0.529 ± 0.002</td>
<td>0.495 ± 0.002</td>
<td>0.547 ± 0.002</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td><strong>0.529 ± 0.002</strong></td>
<td><strong>0.495 ± 0.002</strong></td>
<td><strong>0.547 ± 0.002</strong></td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td>0.529 ± 0.002</td>
<td><strong>0.495 ± 0.002</strong></td>
<td>0.547 ± 0.002</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Choice of $C$ values</th>
<th>DeepWalk</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BlogCatalog</td>
<td></td>
<td>Flickr</td>
</tr>
<tr>
<td>1</td>
<td>0.327 ± 0.004</td>
<td>0.335 ± 0.011</td>
<td><strong>0.313 ± 0.006</strong></td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.329 ± 0.003</td>
<td>0.340 ± 0.010</td>
<td>0.307 ± 0.007</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.333 ± 0.004</strong></td>
<td><strong>0.350 ± 0.006</strong></td>
<td>0.311 ± 0.004</td>
</tr>
<tr>
<td>1</td>
<td>0.344 ± 0.006</td>
<td>0.345 ± 0.005</td>
<td>0.329 ± 0.008</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.344 ± 0.005</td>
<td>0.345 ± 0.005</td>
<td>0.330 ± 0.008</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.344 ± 0.006</strong></td>
<td><strong>0.345 ± 0.005</strong></td>
<td><strong>0.331 ± 0.007</strong></td>
</tr>
<tr>
<td>1</td>
<td>0.372 ± 0.009</td>
<td>0.388 ± 0.007</td>
<td>0.388 ± 0.004</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.373 ± 0.010</td>
<td>0.389 ± 0.010</td>
<td><strong>0.388 ± 0.005</strong></td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.378 ± 0.009</strong></td>
<td><strong>0.392 ± 0.008</strong></td>
<td>0.384 ± 0.006</td>
</tr>
<tr>
<td>1</td>
<td>0.496 ± 0.002</td>
<td>0.466 ± 0.002</td>
<td>0.512 ± 0.002</td>
</tr>
<tr>
<td>1, 10, 100</td>
<td>0.496 ± 0.002</td>
<td>0.466 ± 0.002</td>
<td>0.513 ± 0.002</td>
</tr>
<tr>
<td>0.01, 0.1, 1, 10, 100</td>
<td><strong>0.497 ± 0.002</strong></td>
<td><strong>0.466 ± 0.002</strong></td>
<td>0.512 ± 0.002</td>
</tr>
</tbody>
</table>

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.”


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

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.

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

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.
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

<table>
<thead>
<tr>
<th># of ( t ) values</th>
<th>DeepWalk</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.338 ± 0.009</td>
<td>0.344 ± 0.003</td>
<td>0.319 ± 0.003</td>
</tr>
<tr>
<td>7</td>
<td>0.327 ± 0.004</td>
<td>0.335 ± 0.011</td>
<td>0.313 ± 0.006</td>
</tr>
<tr>
<td>10</td>
<td>0.321 ± 0.007</td>
<td>0.330 ± 0.006</td>
<td>0.304 ± 0.007</td>
</tr>
<tr>
<td>Flickr</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.353 ± 0.001</td>
<td>0.351 ± 0.002</td>
<td>0.337 ± 0.002</td>
</tr>
<tr>
<td>7</td>
<td>0.344 ± 0.006</td>
<td>0.345 ± 0.005</td>
<td>0.329 ± 0.008</td>
</tr>
<tr>
<td>10</td>
<td>0.341 ± 0.004</td>
<td>0.343 ± 0.001</td>
<td>0.329 ± 0.004</td>
</tr>
<tr>
<td>YouTube</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.393 ± 0.006</td>
<td>0.404 ± 0.005</td>
<td>0.399 ± 0.005</td>
</tr>
<tr>
<td>7</td>
<td>0.372 ± 0.009</td>
<td>0.388 ± 0.007</td>
<td>0.388 ± 0.004</td>
</tr>
<tr>
<td>10</td>
<td>0.373 ± 0.008</td>
<td>0.399 ± 0.006</td>
<td>0.395 ± 0.005</td>
</tr>
<tr>
<td>PPI</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.485 ± 0.002</td>
<td>0.462 ± 0.002</td>
<td>0.505 ± 0.002</td>
</tr>
<tr>
<td>7</td>
<td>0.496 ± 0.002</td>
<td>0.466 ± 0.002</td>
<td>0.512 ± 0.002</td>
</tr>
<tr>
<td>10</td>
<td>0.501 ± 0.002</td>
<td>0.467 ± 0.002</td>
<td>0.513 ± 0.003</td>
</tr>
</tbody>
</table>

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.
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.

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

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.

<table>
<thead>
<tr>
<th>Training and prediction methods</th>
<th>DeepWalk</th>
<th>Micro-F1</th>
<th>Node2vec</th>
<th>LINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlogCatalog</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.351 ± 0.006</td>
<td>0.362 ± 0.017</td>
<td>0.337 ± 0.004</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-micro</td>
<td><strong>0.398 ± 0.004</strong></td>
<td><strong>0.409 ± 0.008</strong></td>
<td><strong>0.383 ± 0.006</strong></td>
<td></td>
</tr>
<tr>
<td>Flickr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.355 ± 0.003</td>
<td>0.360 ± 0.004</td>
<td>0.357 ± 0.004</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-micro</td>
<td><strong>0.395 ± 0.003</strong></td>
<td><strong>0.398 ± 0.002</strong></td>
<td><strong>0.397 ± 0.002</strong></td>
<td></td>
</tr>
<tr>
<td>YouTube</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.365 ± 0.006</td>
<td>0.390 ± 0.007</td>
<td>0.400 ± 0.005</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-micro</td>
<td><strong>0.417 ± 0.005</strong></td>
<td><strong>0.432 ± 0.004</strong></td>
<td><strong>0.433 ± 0.002</strong></td>
<td></td>
</tr>
<tr>
<td>PPI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple</td>
<td>0.529 ± 0.002</td>
<td>0.495 ± 0.002</td>
<td>0.547 ± 0.002</td>
<td></td>
</tr>
<tr>
<td>cost-sensitive-simple-micro</td>
<td><strong>0.561 ± 0.001</strong></td>
<td><strong>0.545 ± 0.002</strong></td>
<td><strong>0.576 ± 0.002</strong></td>
<td></td>
</tr>
</tbody>
</table>
has been adopted for multi-label evaluation in social network datasets (Tang and Liu [2009b] Wang and Sukthankar [2011])."  
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.”

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

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.

---

2http://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. (2018e): 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. (2018d): The paper is at https://www.sciencedirect.com/science/article/abs/pii/S0950705118301868, “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. (2018e): 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 phase, 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 $\alpha_i$ predictions to the node $i$ as its predicted labels, where $\alpha_i$ 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 ($\rho$) and precision ($\pi$) of the results (as in Tang and Liu (2009a,b); Tang et al. (2010), $\rho$ and $\pi$ are computed on the ranking truncated to the true number of labels)”
31. Zhang et al. (2018a): The paper is at https://arxiv.org/pdf/1806.02623.pdf. “In the test phase, the one-vs-rest model yields a ranking of labels rather than an exact label assignment. To avoid the thresholding effect [Tang et al. (2009)], assume the number of labels for test data is given [Perozzi et al. (2014); Tang et al. (2009)]. We repeat each experiment for 10 times and report the average Micro-F1 and standard deviation. Analogous results hold for Macro-F1 or other accuracy metrics, and thus are not shown.”

32. 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.”

33. 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@1.”

34. 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

35. 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.

```c
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 \( k \).

```c
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.

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

```python
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)
```


```python
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)
```

Defining the code at [https://git.l3s.uni-hannover.de/khosla/nerd](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](https://github.com/HipGraph/Force2Vec). The following segment shows that the number of labels is used.

```python
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
```


```python
# 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)
```


```python
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 strategy is used in
```
DeepWalk and Node2Vec paper.
y_pred.append(y_pred_proba[inst, :].argsort()[:,-1][:,sum(y_test[inst, :])])

41. 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.

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

The following four papers [42, 43, 44, and 45] 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:


46. 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:

```python
# 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)
```

47. 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:

```python
# 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)
```


```python
# 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)
```

49. Cen et al. (2021): The paper is at https://arxiv.org/abs/2103.00959. The code is at https://github.com/tbudm/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.

```python
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)
```

50. 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)
```

51. Chanpuriya et al. (2021): The paper is at http://proceedings.mlr.press/v139/chanpuriya21a/chanpuriya21a.pdf. The code is at https://github.com/konsotirop/InvertEmbeddings. 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.

```python
y_score = clf.predict_proba(X_test)
y_pred = construct_indicator(y_score, y_test)
m1 = f1_score(y_test, y_pred, average="micro")
ma = f1_score(y_test, y_pred, average="macro")
```

52. 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.
```python
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
```


```python
# 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)
```

54. 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:

```matlab
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:

```matlab
% suppose we know the number of labels for ground truth
```

55. 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:

```python
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)
```

56. 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:

```python
# 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:])

57. 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:

```python
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)
    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_)
```

58. 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:

```python
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:

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

59. 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.

60. 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:

```python
# 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:])

61. 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]
       pred = clf.predict(X_test, top_k_list)
   else:
       pred = clf.predict(X_test)

62. 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]
   pred = clf.predict(X_test, top_k_list)

63. 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))
           test_proba = list(true_prop(test_proba))
       train_proba = np.array(train_proba).T
       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])[:np.count_nonzero(labels)]] = 1
       for i, labels in enumerate(node_labels_test):
           test_pred[i][np.argsort(-test_proba[i])[:np.count_nonzero(labels)]] = 1

64. 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)
       y_sort = np.flip(y, axis=1, dtype=np.int)

   25
# 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)
m_i = f1_score(y_test, y_pred, average="micro")
ma = f1_score(y_test, y_pred, average="macro")

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)

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

67. 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)

68. Gu et al. (2020): The paper is at https://proceedings.neurips.cc/paper/2020/file/8b5c8441a8ff8e151b191c53e1842a38-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.

```python
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
```

69. 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.

```python
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_labels
```

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

```python
def get_classifier_performace(classifier, X_test, y_test, multi_label_binarizer):
    top_k_list_test = [len(l) for l in y_test]
    y_test_pred = predict_top_k(classifier, X_test, top_k_list_test)
```

70. Mitra et al. (2020). 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:

```python
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.flip(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:

```python
# 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.

71. 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)

72. 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.

73. 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.”

74. 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.”

75. 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.

76. **Hou et al. (2019):** The paper is at [https://dl.acm.org/doi/abs/10.1145/3292500.3330948](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).

77. **Xin et al. (2019):** The paper is at [https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8850094](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).

78. **Zhang et al. (2021):** The paper is at [https://arxiv.org/abs/2106.05476](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).

79. **Zhu et al. (2018):** The paper is at [https://dl.acm.org/doi/10.1145/3219819.3220052](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.

80. **García-Durán and Niepert (2017):** The paper is at [https://dl.acm.org/doi/10.5555/3295222.3295265](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).
81. [Perozzi et al. (2017)](https://dl.acm.org/doi/10.1145/3110025.3110086) “We evaluate our method using the same experimental procedure outlined in [Perozzi et al. (2014)](https://dl.acm.org/doi/10.1145/3132847.3132925).” 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).

82. [Cavallari et al. (2017)](https://dl.acm.org/doi/pdf/10.1145/3132847.3132925) “We follow [Perozzi et al. (2014)](https://dl.acm.org/doi/10.1145/3110025.3110086) 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.

83. [Fang et al. (2021)](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 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).

84. [Zhu and Koniusz (2021)](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.

85. [Yang et al. (2018)](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.

86. [Zhang et al. (2018b)](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)](https://ai foss.com/ai-ferrara-university-of-milan-2018), which used the number of labels during evaluation in code.

87. [Fahrbach et al. (2020)](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.

88. [Zhang and Xu (2020)](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.

89. Zhang et al. (2018f): 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 Deepwalk 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.

90. 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.

91. 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.

92. 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 cross validation 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.

93. 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.
94. 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.

95. 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).

96. 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.

97. 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 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.

98. 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.

99. 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.

100. 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 number of labels into account during the prediction stage.

101. [Yao et al. (2021)](https://arxiv.org/pdf/2010.04895.pdf): The paper is at [https://arxiv.org/pdf/2010.04895.pdf](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 number of labels into account during the prediction stage.
References


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.


Jie Zhang, Yan Wang, Jie Tang, and Ming Ding. Spectral network embedding: A fast and scalable method via sparsity, 2018a.


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


