CrossWalk: Learning Cross-Domain Subgraph Representations

Anonymous Authors[1]

Abstract

Graphs often provide a wonderful way to interpret the complex relationships of data. However, this data may be spread across multiple domains, e.g. a python package can be characterized by a language graph, a python code graph, and a dependency graph. This same problem emerges across a wide variety of data-hungry industries where no single data domain is sufficient to represent the entity being sought after i.e. a python package, a user in social media network, or a web page. In this paper we help address this problem with a two-fold contribution: (1) We introduce the PyPI Graph Dataset that includes three graph data domains. (2) We benchmark this dataset with a novel method that extends random walk methods into cross domains that we coin CrossWalk.

1. Introduction

The Python Package Index (PyPI) is home to over one-hundred and twenty-thousand python packages (Bommarito et al. 2019). Finding similar packages in terms of use-cases, features, and performance can be very difficult without specific domain knowledge of a task or the python ecosystem.

One of deep learning's most fundamental advances is the ability to learn meaningful representations from raw data. The PyPI dataset presented in this is unique due to the multi-domain nature of its data which is all unified under the idea of a python package. That is to say, every python package may have a textual description, a place in a large dependency graph, and the python code itself. Each of these data domains have been researched independently. Document representations have been a long researched area within NLP, graph representations are a hot area of research, code representations are a newer but largely unexplored area of representation learning. This dataset not only provides a new context to further explore these domains, but allows for researchers to explore the interaction and generalization of methods that may apply across all of these domains.

One can imagine that a sentence can be thought of a random walk through a language graph thus, the strong performance of word2vec (Mikolov et al., 2013) and other neural probabilistic language models (Bengio et al., 2003) can be applied in an effective manner on arbitrary graphs to learn node representations (Perozzi et al., 2014). We argue that this analogy can and should be pushed further in the pursuit of building general cross-domain methods.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Table 1. Summary of PyPI Graph Dataset

DOMAIN	# NODES	# EDGES
LANGUAGE	166,987	
PYTHON CODE		
DEPENDENCIES	95,951	1,509,378
TOTAL	199,234	

2. Related Work

To the best of our knowledge there is only one other paper that directly addresses the issue of cross-domain representations in the space of graph neural networks (GNNs). (Ouyang et al., 2019) introduces Deep Multi-Graph Embedding (DMGE) which relies on a multiple-gradient descent optimizer to balance learning across domains.

Not only do we learn cross-domain representations we also learn a representation for subgraph. Sub2Vec as detailed in (Adhikari et al., 2017) builds on the formalization in (Perozzi et al., 2014) which itself extends the skip-gram model (Mikolov et al., 2013) to arbitrary graphs.

3. The Dataset

Within the PyPI dataset we interpret each package as a graph consisting of a subgraph for each domain. As shown in Figure 1 a package graph consists of a language graph, a dependency graph, and a python code graph. The following sections detail how we interpret and construct each subgraph.

3.1. The Language Graph

The language graph consists of either the package's ReadMe file or its PyPI description (whichever is longer). If we were to learn a python package representation from just this graph we would end up with a reasonable measure of package similarity as a package creator wants to explain what the package features. However, we reiterate that this one domain is a part to the whole concept that is a python package. [NEED TRANSITION]. We discuss the following methods that learn document representations: TF-IDF and Doc2Vec (Le & Mikolov, 2014).

TF-IDF builds vectors of length of a keyword dictionary. At the corresponding keyword's index there is a value in the range [0, 1] indicating the relative importance of the term to that python package's documentation. The main advantage of this approach is also the interpretability of the calculated vectors, but also encodes more information than the Binary Keyword approach. These vectors were calculated with sklearn's TFIDFVectorizer class.

Doc2Vec (Le & Mikolov, 2014) learns a dense representation of documents alongside a vocabulary. This extends the word2vec (Mikolov et al., 2013) approach to documents. This approach has a much better memory requirement as we don't need each vector to be the length of a keyword dictionary. These word2vec type approaches are well known for encoding the semantic relationships of language.

3.2. The Direct Dependency Graph

3.3. The Code Graph

3.4. Data Collection

3.5. Applications

4. CrossWalk

It has been empirically showed that the skip-gram model (Mikolov et al., 2013) can be applied to learn node representations from the social structure inherent to any arbitrary graph (Perozzi et al., 2014). By taking random walks through the graph, beginning at a vertex of interest, DeepWalk maximizes the likelihood of observing a context vertex v_c conditioned on a target vertex v_t w.r.t the parameters θ. The objective can then be characterized by the following:

$$\arg\max_{\theta} \prod_{(v_c,v_t)\in W} p(v_c|v_t;\theta) \tag{1}$$

Where W is the set of all context target vertex pairs from random walks. However, this objective cannot the representation of a subgraph. Sub2Vec (Ouyang et al., 2019) generalizes doc2vec with the following objective:

$$\arg\max_{\theta} \prod_{(v_c,v_t)\in W} p(v_c|v_t,s;\theta) \tag{2}$$

Where s is a vector that characterizes a specific subgraph. Intuitively, this is the same objective that governs the Doc2Vec method just generalized to arbitrary graphs.

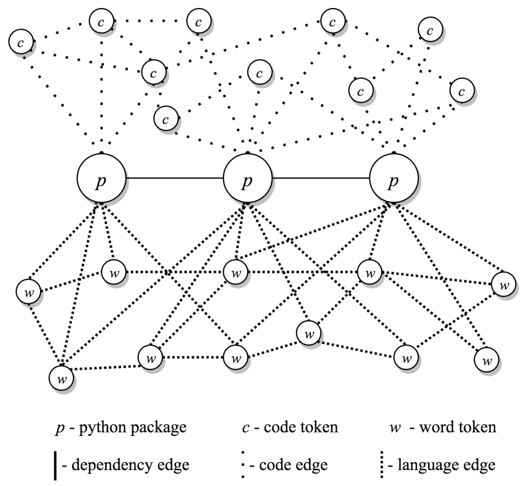

Figure 1. CrossWalk's Graphical Interpretation of the PyPI dataset. Notice the three distinct graphs, i.e domains, and how each relates to a python package.

Our maximum likelihood w.r.t the parameters of our model across all domains present becomes:

$$\arg\max_{\theta} \prod_{i}^{D} \prod_{t}^{V^{(i)}} p(v_t^{(i)}|v_{-w:w}^{(i)}, \mathbf{d};\theta) \tag{3}$$

Where D is the set of all domains, $v_t^{(i)}$ is the tth vertex in the ith domain, and $\mathbf{d}$ is the document vector.

Equation 3. can be further extended to include domain-specific embeddings and cross-domain embeddings. For example, with the PyPI dataset we could learn an embedding that crosses the language and dependency graph domains but remains independent of the code graph. In fact, the total possible number of embeddings that can be learned with CrossWalk over all domains D is:

$$N_e = \sum_{k=1}^{D} \binom{D}{k} \tag{4}$$

However, the authors highly encourage practitioners to not waste resources on learning all possible embeddings but to rather use this property of CrossWalk to determine which intersection of domains is best for a downstream task.

5. Reproducibility

There is a much needed and growing movement in the field of machine learning that encourages the researchers to ensure all results are easily reproduced. In the context of this paper and our results we make this priority.

Table 2. Topic classification results on PyPi Graph Dataset

% Labeled	10%	50%	90%
TF-IDF	30.55%	36.84%	38.40%
Doc2Vec	31.82%	**41.03**%	**43.33**%
DeepWalk	**34.6**%	39.11 %	40.53%

We want to help create a standard that moves beyond providing a script or notebook that can be run that output the same numbers found in the paper. Instead, we want to build a tool that will allow other researchers to not just match our numbers but build on any aspect of our codebase that may benefit future research.

To accomplish this goal we highlight the following features in our codebase that will be open to the public upon the publication of this paper. (1) We provide an API that makes it easy for users to switch out datasets beyond those used in this paper. (2) We provide tooling to explore the embedding spaces learned via CrossWalk in the form of a web API. (3) We prioritize the use open source libraries and frameworks so no software used is hidden behind a proprietary wall.

6. Representation Evaluation

6.1. Topic Classification

In order to evaluate the learned python package embeddings we use an extrinsic evaluation task. We define this task as the classification of python packages into different categories. GithHub allows for users to label their projects under various tags - as a vast majority of PyPI packages link to a GitHub repository we were able to extract a subset of python packages that are labeled by the creator into these topics. GitHub encourages users to classify their repositories "to help other people find and contribute to your project, you can add topics to your repository related to your project's intended purpose, subject area, affinity groups, or other important qualities." [1] This further justifies our interpretation of topics as classification classes.

Following the example of (Perozzi et al., 2014) (maybe add others?) we use a one vs. rest logistic regression model and a shallow neural network to classify the learned representations. These model choices reflect our desire to intentionally cripple the transformation ability of the classifier so that the natural discriminatory ability of the learned embeddings is what is being evaluated.

This task empirically shows that no single domain is sufficient to capture a complete representation of a python package. [Perform Ablation Study!]

[1]https://help.github.com/en/github/administering-a-repository/classifying-your-repository-with-topics

6.2. Link Prediction

6.3. Topic Generation

7. Conclusion

References

Adhikari, B., Zhang, Y., Ramakrishnan, N., and Prakash, B. A. Distributed representations of subgraphs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 111–117. IEEE, 2017.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Le, Q. V. and Mikolov, T. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. URL http://arxiv.org/abs/1405.4053.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Ouyang, Y., Guo, B., Tang, X., He, X., Xiong, J., and Yu, Z. Learning cross-domain representation with multi-graph neural network. *arXiv preprint arXiv:1905.10095*, 2019.

Perozzi, B., Al-Rfou, R., and Skiena, S. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 701–710, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2956-9. doi: 10.1145/2623330.2623732. URL http://doi.acm.org/10.1145/2623330.2623732.