Learning One Cross-Domain Graph Representation with PyPI Data

Anonymous Authors[1]

Abstract

To understand a new dataset a practitioner may turn to sums, buckets, and averages to quantity different properties in the data. Traditionally, modern machine learning does not help in this situation - only in matters when one wants to predict. In this case study we present a careful analysis of a data space so large and specialized that traditional statics cannot properly characterize it's properties. We apply both traditional and more recent, advanced techniques to develop an embedding space for python packages hosted on PyPi. To evaluate these results we develop a task consisting of finding "package neighbors" determined by python developers. With this evaluation task, the multiple methods are compared and we recommend (make recommendation here), along with the

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 and the Python ecosystem. Motivated by this we build multiple embedding spaces of python packages based on the documentation and/or graphical structure of direct dependencies.

By using the documentation of python packages this problem is framed as a natural language processing task. We take advantage of this and apply traditional NLP techniques as well as more advanced recent techniques like doc2vec (Le et al. 2014) and lda2vec (Moody 2016).

However building an embedding space for a data such as python packages introduces a problem: How does one evaluate such a space when there are no experts that can evaluate a potential solution due to the extreme nichety of the space. E.g. With word embeddings any proficient speaker of the language can reasonably evaluate if the nearest-neighbors of an embedding are relevant. With an extremely niche space like python packages expert python developers may be able to evaluate a subset of the space (e.g. python web developers could indicate which packages are similar to a package like flask). To mitigate this problem we plan to sample these subsets of Python experts to get list of potential package neighbors that would be able to indicate the correctness of the resulting embedding space.

[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.

2. The Dataset

2.1. Data Collection

3. Learning Python Package Representations

One of Deep Learning's most relevant 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 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 domains have been researched independently. Document representations have been a long researched area within NLP, graph representations are becoming a hotter 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 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.

Within the PyPI dataset we interpret each package as a graph consisting of a subgraph for each domain. As shown in [ADD A FIGURE AND REFERENCE HERE] a package

graph consists of a language graph, a dependency graph, and a code graph. The following sections detail how we construct each subgraph and the current SOTA methods in representation learning for that domain.

3.1. The Language Graph

The language graph consists of either the package's GitHub 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. The Cross Domain Graph

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

$$\arg\max_{\theta} \prod_{i}^{D} \prod_{t}^{V} P(v_t^{(i)} | v_{-w:w}^{(i)}, \mathbf{d}; \theta) \tag{1}$$

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

4. Representation Evaluation

4.1. 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. Github allows for users to label their projects under various tags - as a vast majority of PyPI packages link to a github we were able to extract a subset of python packages that are labeled by the creator into these categories. Following the example of Perozzi et al. (maybe add others?) we use a logistic regression and a shallow neural network to classify the learned representations.

This task empirically shows that no single domain is sufficient to capture a complete representation of a python package.

4.2. Link Prediction

4.3. Topic Generation

5. Conclusion

References

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.

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.