Python Dependency Network Analysis

Network Science - Project 1

Duarte Alves 89439, Guilherme Ribeiro 89454, Nikoletta Matsur 89513
Group 08

October 2020

1 Introduction

The Python ecosystem has been growing in popularity in the past years and in this report, we aim to analyse the enthralling network of the language package dependencies.

In this report, we perform an analysis on the Python dependency network based on a data set [1], gathered from PyPI (Python Package Index) in late 2015.

We start by modeling the network. The packages and their dependencies are represented by nodes and edges in a directed graph. For example, the package *networkX* requires the package *numpy* so there is an edge going from the node that represents *networkX* to the node that represents *numpy*.

2 Network Properties

In the analysed data, every record is defined by a package name, a package version and the required package name. However, the requirement version is not supplied so there is no way to know it. To solve this problem, we chose to ignore the package version. This is possible since there are 72253 dependencies and 58774 packages, from which 10 have multiple versions, meaning we can safely discard the versions without any loss of accuracy.

Throughout this report, we will be focusing on the 'In degree' because it provides us with more relevant information, since it translates to the number of packages that depend on a given package.

The degree distribution is one of the most fundamental network properties. As depicted in the figure below, we can see that the network 'In degree' distribution follows a power-law with an average of 1.229964 neighbours. Nevertheless, this average does not give us the real picture as the 'In variance' is very high (645.975085) meaning that there are many nodes with a really high degree and others with the degree very low.

Another important step in the study of a network is the clustering analysis. Here, we resort to the network clustering coefficient [2], defined as:

$$C = \frac{1}{n} \sum_{i=1}^{n} C_i$$

where n is the number of nodes in the network and C_i is the local clustering coefficient for node i.

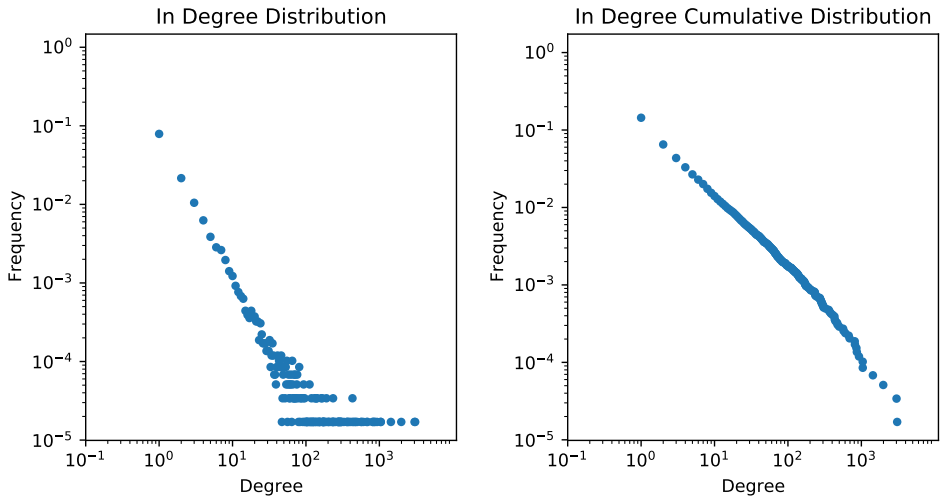

Figure 1: Network In/Out Degree Distributions

The obtained value is approximately 0.01543, which could be explained by the large amount of packages without any connections (more than 50%). However, putting these packages aside, the clustering coefficient keeps turning out low, of approximately 0.03455.

These values suggest an absence of locality in the network meaning that, for any two requirements of a package, there is a low probability that one of them also depends on the other.

In addition to that, we can also evaluate how close the nodes are to each other. As such, we can compute the average path length [2] for the network:

$$APL = \frac{1}{n \cdot (n-1)} \cdot \sum_{i \neq j} d(v_i, v_j)$$

where n denotes the number of vertices on the network, and $d(v_i, v_j)$ the shortest distance from vertex i to j.

However, a close inspection of the network reveals that the network is not connected with the existence of a extremely large number of strongly connected components (totaling 58696 SCCs, almost as the number of vertices). This makes the creation of a giant component impossible, since the largest SCC has only 17 vertices, which means the APL is not defined.

3 Centrality Measures

In this section, we will assess the importance of a given package within the network. In order to achieve such goal, we will be considering the 'In degree' centrality measure which, for each node, is defined as the vertex 'In degree'. Therefore, we consider a package with a higher degree as one of higher importance, as many others depend on it.

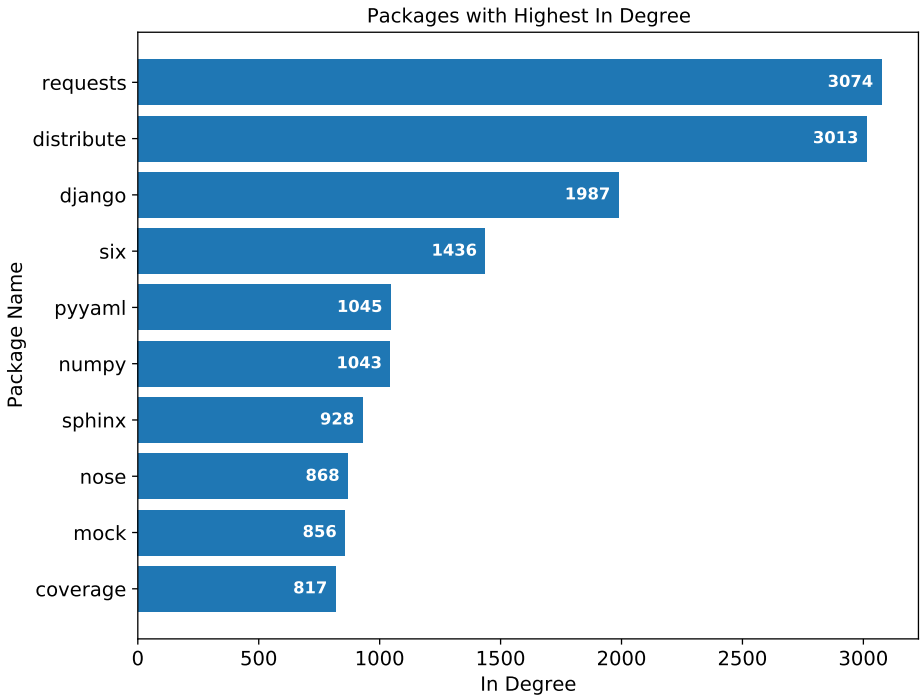

Figure 2: Histogram with In degree for Python packages

4 Network Topology

After an evaluation of the network properties, we are ready to analyse why the network has grown to take this shape. One important aspect was just mentioned and it's the continuous growth of the network, seeing that in the years leading to the data set creation, the number of Python packages grew exponentially [3].

In addition to the network growth, we have preferential attachment, where known packages tend to be more required, mostly because they are verified to work and to solve common problems.

Unlike many large scale networks, this one has a rather small clustering coefficient as well as a large number of SCCs. We can explain this as a consequence of the specific domain of package dependency networks. Regarding the clustering coefficient, we can think that just because a package requires another two packages, it doesn't imply the two packages are related. About the SCCs, we can see that there are many isolated packages, as well as a small number of cycles, since circular dependencies should be avoided in software development.

5 Conclusion

This practical analysis studied the Python dependency network. We started by exploring some of the network properties. After which, we evaluated the importance of a node 'In degree', using degree centrality to discover the most important nodes in the network. Finally, we explored the reasons behind the creation of the topology for the Python dependency network.

References

[1] *Python Dependencies Data Set,*
`http://kgullikson88.github.io/Downloads/PypiDependencies/requirements.csv`

[2] M. E. J. Newmann. *Networks: An Introduction.* 2010 Oxford University Press, ISBN 978-0-19-920665-0

[3] A.Duncan, T. Mens, M. Claes. *On the topology of package dependency networks: a comparison of three programming language ecosystems*
`https://dl.acm.org/doi/abs/10.1145/2993412.3003382`