Visualizing relationships between python packages

Robert Kozikowski

[2016-07-10 Sun 21:24]

Contents

1 Introduction

Using the github data on BigQuery and new force layout in d3 via the Verlet numerical integration I implemented graph visualization of relationships between 3500 most popular python packages.

Center cluster of "standard library" indeed looks like "a bag of wool that my cat would want to play with", but there are relevant clusters on the outskirts, like scientific python or django.

You can see the app at `http://clustering.kozikow.com?center=numpy`. You can:

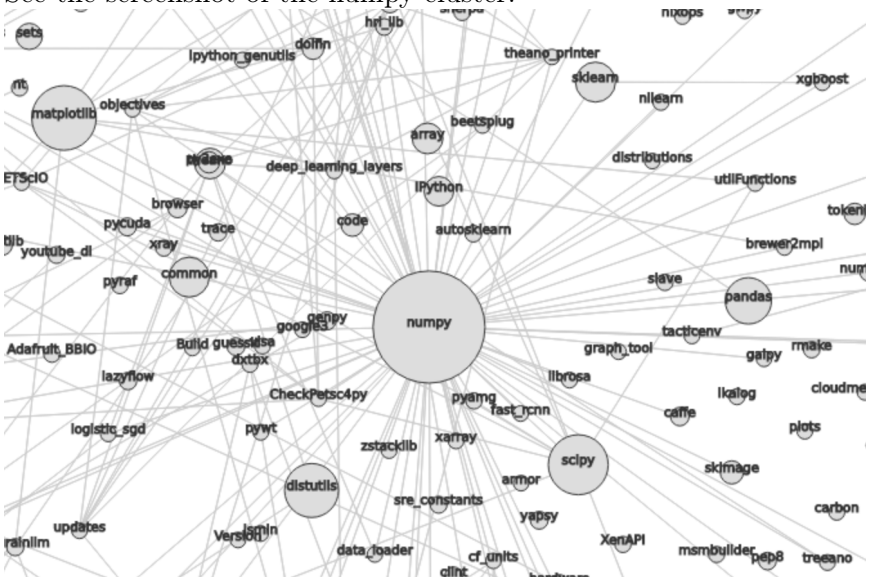
- Pass different package names as a query argument in the URL.
- Scroll the page horizontally and vertically.
- Click a node to open the pypi. Note that not all packages are on pypi.
- I will migrate to the graphistry soon. It will let you mouse over the package or zoom in and out.

Graph properties:

- Each node is each python package found on github. Radius is calculated in DataFrame with nodes section.
- For two packages A and B, weight of an edge is $(\frac{|A \cap B|}{min(|A|,|B|)})^2$, where $|A \cap B|$ is number of occurrences of packages A and B within the same file. I will migrate it to the normalized pointwise mutual information soon.

- Edges with weight smaller than 0.1 are removed.
- Algorithm searches for minimal energy state by the Verlet Integration according to simulation parameters.

See the screenshot of the numpy cluster:



Graph visualizations often lack actionable insights except looking cool. Types of insights you can use this for:

- I have been exploring packages in the scientific python cluster a lot, and I found a few things I plan to use.
- Use this to evaluate which web framework to use. You may think that django became too heavyweight or some other framework have dying community.
- Find some interesting python use cases, like robotics cluster.

Revision history of this post is on github.

2 Analysis of specific clusters

2.1 Scientific python land

Unsurprisingly, it is centered on numpy. Many of scientific packages are in close proximity, like scipy, pandas, sklearn or matplotlib.

I have found package networkx and graph_tool in the nearby proximity, that I plan to use for analyzing data from this post.

You can see even division between statistics and machine learning, by sklearn being surrounded by packages like xgboost or theano.

2.2 Robotics land

Nearby to scientific python land there is a robotics land centered at rospy. Some computational geometry packages like shapely are nearby.

2.3 Web frameworks

Web frameworks are interesting:

- It could be said that sqlalchemy is a center of web frameworks land.
- Found nearby, there's a massive and monolithic cluster for django.
- Smaller nearby clusters for flask and pyramid.
- pylons, lacking a cluster of its own, in between django and sqlalchemy.
- Small cluster for zope, also nearby sqlalchemy
- tornado got swallowed by the big cluster of standard library in the middle, but is still close to other web frameworks.
- Some smaller web frameworks like gluon (web2py) or turbo gears ended up close to django, but barely visible and without clusters of their own.

2.4 Open stack land

Cloud computing, open stack and low level networking land centered at nova contain packages like oslo, nova, webob, neutron, ironic or netaddr.

2.5 Testing land

Centered on unittest. Some packages in the cluster are testfixtures or atomic-reactor.

2.6 Gaming

Cluster for some gaming related libraries, is centered around pygame.

3 Potentials for further analysis

3.1 Other programming languages

Majority of the code is not specific to python. Only the first step, create a table with packages, is specific to python.

I had to do a lot of work on fitting the parameters in Simulation parameters to make the graph look good enough. I suspect that I would have to do similar fitting to each language, as each language graph would have different properties.

Probably majority of languages would have "heavy weight" center cluster that makes it hard to fit the parameters, so maybe removing the cluster like in described in Reduce an effect of a heavy weight center cluster could make algorithm more easily generalizeable to other languages.

3.2 Reduce an effect of a heavy weight center cluster

"Standard library" cluster in the center is very heavyweight and includes many packages. It is also the least interesting, as everyone knows those packages, so there is little insight to be gained.

Removing standard library could improve the quality of visualization. Removing just standard library is not easily generalizeable to other programming languages.

Removing the biggest cluster as detected by clustering algorithm from sklearn or networkx could work well. Alternatively, I could cluster nodes prior to visualization and let users hide some clusters from the browser.

3.3 Reduce an effect of heavy weight packages

In current visualization, big central packages like django, numpy, os and sys dominate the graph. I believe that they dominate some of the smaller, more relevant relationships between smaller packages.

I thought about replacing edge weight $(\frac{|A \cap B|}{min(|A|,|B|)})^2$ by $(\frac{|A \cap B|}{|A|*|B|})^2$, but that could end up clustering packages by size rather than by common usage.

3.4 Search for "Alternatives to package X", e.g. seaborn vs bokeh

For example, it would be interesting to cluster together all python data visualization packages.

Intuitively, such packages would be used in similar context, but would be rarely used together. They would have high correlation of their neighbor weights, but low direct edge. This would work in many situations, but there are some others it wouldn't handle well. Example case it wouldn't handle well:

- sqlalchemy is an alternative to django built-in ORM.
- django ORM is only used in django.
- django ORM is not well usable in other web frameworks like flask.
- other web frameworks make heavy use of flask ORM, but not django built-in ORM.

Therefore, django ORM and sqlalchemy wouldn't have their neighbor weights correlated. I might got some ORM details wrong, as I don't do much web dev.

3.5 networkx and graph_tool packages

Thanks to this visualization I have found about networkx and graph_tool packages. It have some niceties for analyzing graphs. I plan to take a look at package dependency data using those packages.

3.6 Within repository relationship

Currently, I am only looking at imports within the same file. It could be interesting to look at the same graph built using "within same repository" relationship.

3.7 Highlight when overlay

- Highlight a node and edges when overlaying with mouse.
- List all edges in a popup
- Potentially let user reduce number of packages

4 Data

- Post-processed JSON data used by d3
- Publicly available BigQuery tables with all the data. See Reproduce section to see how each table was generated.

5 Steps to reproduce

5.1 Extract data from BigQuery

5.1.1 Create a table with packages

Save to wide-silo-135723:github_clustering.packages_in_file_py:

```
SELECT
  id,
  NEST(UNIQUE(COALESCE(
      REGEXP_EXTRACT(line, r"^from ([a-zA-Z0-9_-]+).*import"),
      REGEXP_EXTRACT(line, r"^import ([a-zA-Z0-9_-]+)")))) AS package
FROM (
  SELECT
    id AS id,
    LTRIM(SPLIT(content, "\n")) AS line,
  FROM
    [fh-bigquery:github_extracts.contents_py]
  HAVING
    line CONTAINS "import")
GROUP BY id
HAVING LENGTH(package) > 0;
```

Table will have two fields - id representing the file and repeated field with packages in the single file. Repeated fields are like arrays - the best description of repeated fields I found.

This is the only step that is specific for python.

5.1.2 Verify the packages_in_file_py table

Check that imports have been correctly parsed out from some random file.

```
SELECT
    GROUP_CONCAT(package, ", ") AS packages,
    COUNT(package) AS count
FROM [wide-silo-135723:github_clustering.packages_in_file_py]
WHERE id == "009e3877f01393ae7a4e495015c0e73b5aa48ea7"
```

packages	count
distutils, itertools, numpy, decimal, pandas, csv, warnings, future, IPython, math, locale, sys	12

5.1.3 Filter out not popular packages

```
SELECT
  COUNT(DISTINCT(package))
FROM (SELECT
  package,
  count(id) AS count
FROM [wide-silo-135723:github_clustering.packages_in_file_py]
GROUP BY 1)
WHERE count > 200;
```

There are 3501 packages with at least 200 occurrences and it seems like a fine cut off point. Create a filtered table, wide-silo-135723:github_clustering.packages_in_file_top_py:

```
SELECT
    id,
    NEST(package) AS package
FROM (SELECT
        package,
        count(id) AS count,
        NEST(id) AS id
    FROM [wide-silo-135723:github_clustering.packages_in_file_py]
    GROUP BY 1)
WHERE count > 200
GROUP BY id;
```

Results are in [wide-silo-135723:github_clustering.packages_in_file_top_py].

```
SELECT
    COUNT(DISTINCT(package))
FROM [wide-silo-135723:github_clustering.packages_in_file_top_py];
```

```
3501
```

5.1.4 Generate graph edges

I will generate edges and save it to table wide-silo-135723:github_clustering.packages_in_file_edges_py

```
SELECT
  p1.package AS package1,
```

```
  p2.package AS package2,
  COUNT(*) AS count
FROM (SELECT
  id,
  package
FROM FLATTEN([wide-silo-135723:github_clustering.packages_in_file_top_py], package)) AS
JOIN
(SELECT
  id,
  package
FROM [wide-silo-135723:github_clustering.packages_in_file_top_py]) AS p2
ON (p1.id == p2.id)
GROUP BY 1,2
ORDER BY count DESC;
```

Top 10 edges:

```
SELECT
    package1,
    package2,
    count AS count
FROM [wide-silo-135723:github_clustering.packages_in_file_edges_py]
WHERE package1 < package2
ORDER BY count DESC
LIMIT 10;
```

package1	package2	count
os	sys	393311
os	re	156765
os	time	156320
logging	os	134478
sys	time	133396
re	sys	122375
__future__	django	119335
__future__	os	109319
os	subprocess	106862
datetime	django	94111

5.1.5 Filter out irrelevant edges

Quantiles of the edge weight:

```
SELECT
    GROUP_CONCAT(STRING(QUANTILES(count, 11)), ", ")
FROM [wide-silo-135723:github_clustering.packages_in_file_edges_py];
```

```
1, 1, 1, 2, 3, 4, 7, 12, 24, 70, 1005020
```

In my first implementation I filtered edges out based on the total count. It was not a good approach, as a small relationship between two big packages was more likely to stay than strong relationship between too small packages.

Create wide-silo-135723:github_clustering.packages_in_file_nodes_py:

```
SELECT
  package AS package,
  COUNT(id) AS count
FROM [github_clustering.packages_in_file_top_py]
GROUP BY 1;
```

package	count
os	1005020
sys	784379
django	618941
__future__	445335
time	359073
re	349309

Create the table packages_in_file_edges_top_py:

```
SELECT
    edges.package1 AS package1,
    edges.package2 AS package2,
    # Wordpress gets confused by less than sign after nodes1.count
    edges.count / IF(nodes1.count nodes2.count,
        nodes1.count,
        nodes2.count) AS strength,
    edges.count AS count
FROM [wide-silo-135723:github_clustering.packages_in_file_edges_py] AS edges
JOIN [wide-silo-135723:github_clustering.packages_in_file_nodes_py] AS nodes1
    ON edges.package1 == nodes1.package
JOIN [wide-silo-135723:github_clustering.packages_in_file_nodes_py] AS nodes2
    ON edges.package2 == nodes2.package
HAVING strength > 0.33
AND package1 <= package2;
```

Full results in google docs.

5.2 Process data with Pandas to json

5.2.1 Load csv and verify edges with pandas

```
import pandas as pd
import math

df = pd.read_csv("edges.csv")
pd_df = df[( df.package1 == "pandas" ) | ( df.package2 == "pandas" )]
pd_df.loc[pd_df.package1 == "pandas","other_package"] = pd_df[pd_df.package1 == "panda
pd_df.loc[pd_df.package2 == "pandas","other_package"] = pd_df[pd_df.package2 == "panda

df_to_org(pd_df.loc[:,["other_package", "count"]])

print "\n", len(pd_df), "total edges with pandas"
```

other_package	count
pandas	33846
numpy	21813
statsmodels	1355
seaborn	1164
zipline	684
11 more rows	

16 total edges with pandas

5.2.2 DataFrame with nodes

```
nodes_df = df[df.package1 == df.package2].reset_index().loc[:, ["package1", "count"]].
nodes_df["label"] = nodes_df.package1
nodes_df["id"] = nodes_df.index
nodes_df["r"] = (nodes_df["count"] / nodes_df["count"].min()).apply(math.sqrt) + 5
nodes_df["count"].apply(lambda s: str(s) + " total usages\n")
df_to_org(nodes_df)
```

package1	count	label	id	r
os	1005020	os	0	75.711381704
sys	784379	sys	1	67.4690570169
django	618941	django	2	60.4915169887
__future__	445335	__future__	3	52.0701286903
time	359073	time	4	47.2662138808
3460 more rows				

5.2.3 Create map of node name -> id

```
id_map = nodes_df.reset_index().set_index("package1").to_dict()["index"]

print pd.Series(id_map).sort_values()[:5]
```

```
os            0
sys           1
django        2
__future__    3
time          4
dtype: int64
```

5.2.4 Create edges data frame

```
edges_df = df.copy()
edges_df["source"] = edges_df.package1.apply(lambda p: id_map[p])
edges_df["target"] = edges_df.package2.apply(lambda p: id_map[p])
edges_df = edges_df.merge(nodes_df[["id", "count"]], left_on="source", right_on="id", h
edges_df = edges_df.merge(nodes_df[["id", "count"]], left_on="target", right_on="id", h
df_to_org(edges_df)

print "\ndf and edges_df should be the same length: ", len(df), len(edges_df)
```

package1	package2	strength	count_x	source	target	id_x	count_y	id
os	os	1.0	1005020	0	0	0	1005020	
sys	sys	1.0	784379	1	1	1	784379	
django	django	1.0	618941	2	2	2	618941	
__future__	__future__	1.0	445335	3	3	3	445335	
os	sys	0.501429793505	393311	0	1	0	1005020	
11117 more rows								

df and edges_df should be the same length: 11122 11122

5.2.5 Add reversed edge

```
edges_rev_df = edges_df.copy()
edges_rev_df.loc[:,["source", "target"]] = edges_rev_df.loc[:,["target", "source"]].va
edges_df = edges_df.append(edges_rev_df)
df_to_org(edges_df)
```

package1	package2	strength	count_x	source	target	id_x	count_y	id
os	os	1.0	1005020	0	0	0	1005020	
sys	sys	1.0	784379	1	1	1	784379	
django	django	1.0	618941	2	2	2	618941	
__future__	__future__	1.0	445335	3	3	3	445335	
os	sys	0.501429793505	393311	0	1	0	1005020	
22239 more rows								

5.2.6 Truncate edges DataFrame

```
edges_df = edges_df[["source", "target", "strength"]]
df_to_org(edges_df)
```

source	target	strength
0.0	0.0	1.0
1.0	1.0	1.0
2.0	2.0	1.0
3.0	3.0	1.0
0.0	1.0	0.501429793505
22239 more rows		

5.2.7 After running simulation in the browser, get saved positions

The whole simulation takes a minute to stabilize. I could just download an image, but there are extra features like pressing the node opens pypi.

Download all positions after the simulation from the javascript console:

```
var positions = nodes.map(function bar (n) { return [n.id, n.x, n.y]; })
JSON.stringify()
```

Join the positions x and y with edges dataframe, so they will get picked up by the d3.

```
pos_df = pd.read_json("fixed-positions.json")
pos_df.columns = ["id", "x", "y"]
nodes_df = nodes_df.merge(pos_df, on="id")
```

5.2.8 Truncate nodes DataFrame

```
# c will be collision strength. Prevent labels from overlaping.
nodes_df["c"] = pd.DataFrame([nodes_df.label.str.len() * 1.8, nodes_df.r]).max() + 5
nodes_df = nodes_df[["id", "r", "label", "c", "x", "y"]]
df_to_org(nodes_df)
```

	id	r	label	c	x	y
	0	75.711381704	os	80.711381704	158.70817237	396.074393369
	1	67.4690570169	sys	72.4690570169	362.371142521	-292.138913114
	2	60.4915169887	django	65.4915169887	526.471326062	1607.83507287
	3	52.0701286903	__future__	57.0701286903	1354.91212894	680.325432179
	4	47.2662138808	time	52.2662138808	419.407448663	439.872927665

3460 more rows

5.2.9 Save files to json

```
# Truncate columns
with open("graph.js", "w") as f:
    f.write("var nodes = {}\n\n".format(nodes_df.to_dict(orient="records")))
    f.write("var nodeIds = {}\n".format(id_map))
    f.write("var links = {}\n\n".format(edges_df.to_dict(orient="records")))
```

5.3 Draw a graph using the new d3 Verlet Integration algorithm

The physical simulation Simulation uses the new velocity Verlet integration force graph in d3 v 4.0. Simulation takes about one minute to stabilize, so for viewing purposes I hard-coded the position of node after running simulation on my machine.

The core component of the simulation is:

```
var simulation = d3.forceSimulation(nodes)
    .force("charge", d3.forceManyBody().strength(-400))
    .force("link", d3.forceLink(links).distance(30).strength(function (d) {
        return d.strength * d.strength;
    }))
    .force("collide", d3.forceCollide().radius(function(d) {
        return d.c;
    }).strength(5))
    .force("x", d3.forceX().strength(0.1))
```

```
.force("y", d3.forceY().strength(0.1))
.on("tick", ticked);
```

To re-run the simulation you can:

- Remove fixed positions added in one of pandas processing steps.
- Uncomment the "forces" in the javascript file.

5.3.1 Simulation parameters

I have been tweaking simulation parameters for a while. Very dense "center" of the graph is in conflict with clusters on the edge of the graph.

As you may see in the current graph, nodes in the center sometimes overlap, while distance between nodes on the edge of a graph is big.

I got as much as I could from the collision parameter and increasing it further wasn't helpful. Potentially I could increase gravity towards the center, but then some of the valuable "clusters" from edges of the graph got lumped into the big "kernel" in the center. The most promising approach at this point would be to try some ways of Reduce an effect of a heavy weight center cluster.

1. Attraction forces
   - Weight of edge between packages A and B: $(\frac{|A \cap B|}{min(|A|,|B|)})^2$, with distance 30
   - Gravity towards center: 0.1
2. Repulsion forces
   - Repulsion between nodes: -400
   - Strength of nodes collision: 5