INFO 623 Social Network Analytics

Stack-Overflow Tag Network Analysis

Anju Tyagi
Sandeep Kumar Singh

Introduction:

StackOverflow is a platform where students and professionals post queries and answer questions about programming. It is a platform to showcase their knowledge. It is owned by the StackExchange Network. The answers are upvoted based on its usefulness to the community. Users can also use StackOverflow to advance their careers. It is a community of more than seven million programmers.

Along with questions, students and professionals also creates their Developer Stories in Stack Overflow. They are designed to give a full picture of a students/professionals based on what they know and built. Stories gives insight into their proficiency in different technologies, open source contributions, blogs, applications and websites they've built, Stack Overflow activity, and much more. Technology tags is a part of their profile which highlight's their preferred tools and languages. Each tag represents a specific technology that the candidate has used, likes or dislikes. Thus technology tags on Developer Stories is the great way to analyze the relation between various technologies.

Objective:

Being in the IT company, we spend a lot of time and energy thinking about tech ecosystems and how technologies are related to each other. One way to get at this idea of relationships between technologies is tag correlations, how often technology tags at Stack Overflow appear together relative to how often they appear separately. One place we see developers using tags at Stack Overflow is on their Developer Stories, or professional profiles/CVs/resumes. It will be interesting to know how technologies are connected and how they are used together, developers' own descriptions of their work and careers is a great place to get that.

This analysis can help the individuals from various background. For experienced individual can explore the technologies which are related or attached to their current field or area of expertise so they can develop new skills, even if someone wants to change their domain in IT suppose from frontend to backend or full-stack, so by analysing which popular technologies, developers are using currently and from fresh graduate perspective or someone who is new to IT world, this analysis can help them to know the fastest growing technologies and can upskill themselves accordingly.

Here, we carry out an analysis of the Stack Overflow tags viewed as a network, or a graph. Specifically, we aim to get some insight about the user communities by representing tags and their cooccurrences as a graph, where the graph nodes are the tags themselves, and an edge between two nodes exists if the corresponding tags are found together in the same Stack Overflow question. The resulting network edges are weighted, i.e., the more questions exist with two tags co-occurring, the higher the weight of the relevant edge between them will be.

In graph terminology, the resulting graph is a weighted undirected one.

Problem Statements:

1. How technologies relate to each other in the network.
2. How often technology appears together.
3. Analysing the busiest node and try to identify the reason behind it.
4. Which technology has the greatest number of centrality and which has least number of centrality.
5. Analysing different clusters for each programming language and family pattern like android with java or embedded systems with C and C++.

Data Description:

The dataset include two files only a subset of tags used on Developer Stories, tags that were used by at least 0.5% of users and were correlated with another tag with a correlation coefficient above 0.1. This means that very sparsely used tags and tags that are not used with other tags were filtered out.

Nodes and edges can have metadata associated with them.

Node metadata :
node-size : Proportional to how many developers have that tag in their developer story profile.
group : which group that node belongs to (calculated via a cluster walk-trap).

Edge metadata :
value : proportional to how correlated those two tags are (correlation coefficient * 100).

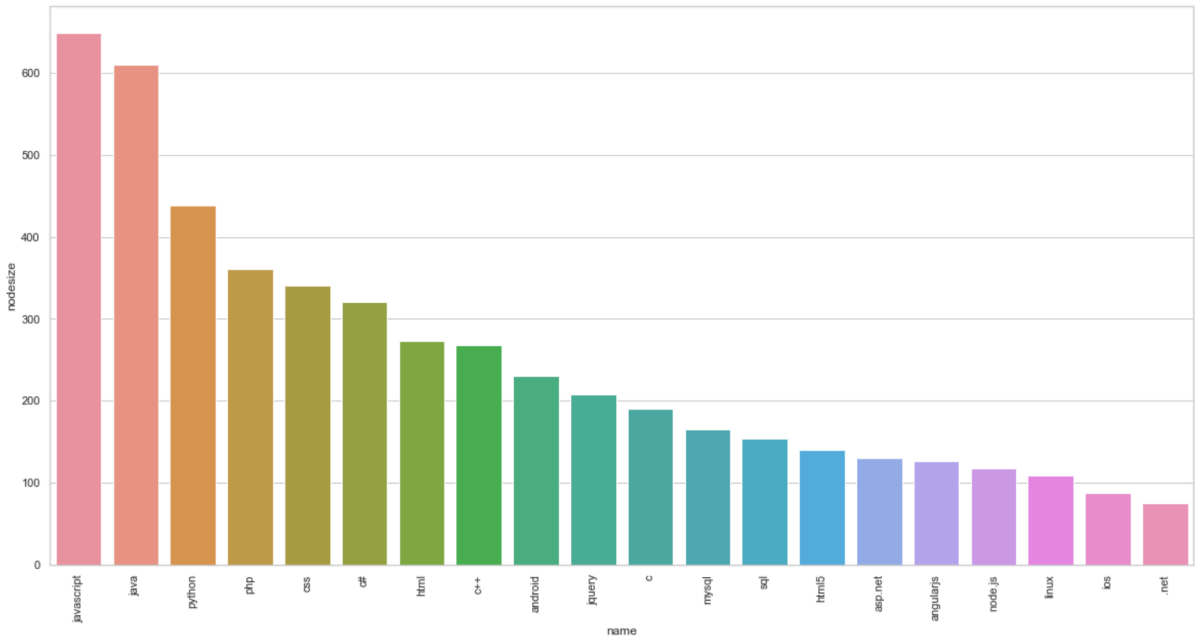
	name	group	nodesize
0	html	6	272.45
1	css	6	341.17
2	hibernate	8	29.83
3	spring	8	52.84
4	ruby	3	70.14
5	ruby-on-rails	3	55.31
6	ios	4	87.46
7	swift	4	63.62
8	html5	6	140.18
9	c	1	189.83

Nodes (stack_network_nodes)

	source	target	value
0	azure	.net	20.933192
1	sql-server	.net	32.322524
2	asp.net	.net	48.407030
3	entity-framework	.net	24.370903
4	wpf	.net	32.350925
5	linq	.net	20.501744
6	wcf	.net	28.074400
7	c#	.net	62.167895
8	tdd	agile	37.146590
9	codeigniter	ajax	23.191900

Edges (stack_network_links)

figure 1: Tag frequency distribution for the first 20 tags



Top 10 most frequent tags:

	name	group	nodesize
14	javascript	6	649.16
42	java	8	610.65
46	python	1	438.67
18	php	6	361.22
1	css	6	341.17
12	c#	2	321.13
0	html	6	272.45
10	c++	1	268.11
41	android	4	229.86
15	jquery	6	208.29

Network Visualisation

We can see the different connected components in the graph, often consisting of 2-3 edges. E.g (excel, excel-vba) and (testing, selenium) which probably refers to the business analyst and Quality assurance developers.

Also we can see by graph that the graph is not connected graph and we have many connected components in the network.

In this interactive network visualisation the size of each circle represents how often that tag is used; tags with larger circles are used more often. The circles are coloured based on their subgroup membership within the network as a whole, which is calculated via many random walks (a cluster walk-trap). This network includes tags that are used more than 800 times on Developer Stories and have correlations greater than 0.1 with other tags.

We see here, for example, more evidence about how developers are using Python both for data science along with R (another language used for data science), Pandas, and NumPy, as well as for web development with Django and Flask.

There is so much we can see by exploring this network! One thing we can notice is subgroups within the network that show us tech ecosystems, some of them densely interconnected. We see some groups made up of:

1. Front-end web development technologies from HTML to JavaScript to Bootstrap.
2. Microsoft-related technologies including C#, .NET, and SQL Server.
3. DevOps technologies like AWS and Docker.
4. Mobile technologies including Android and Objective-C.

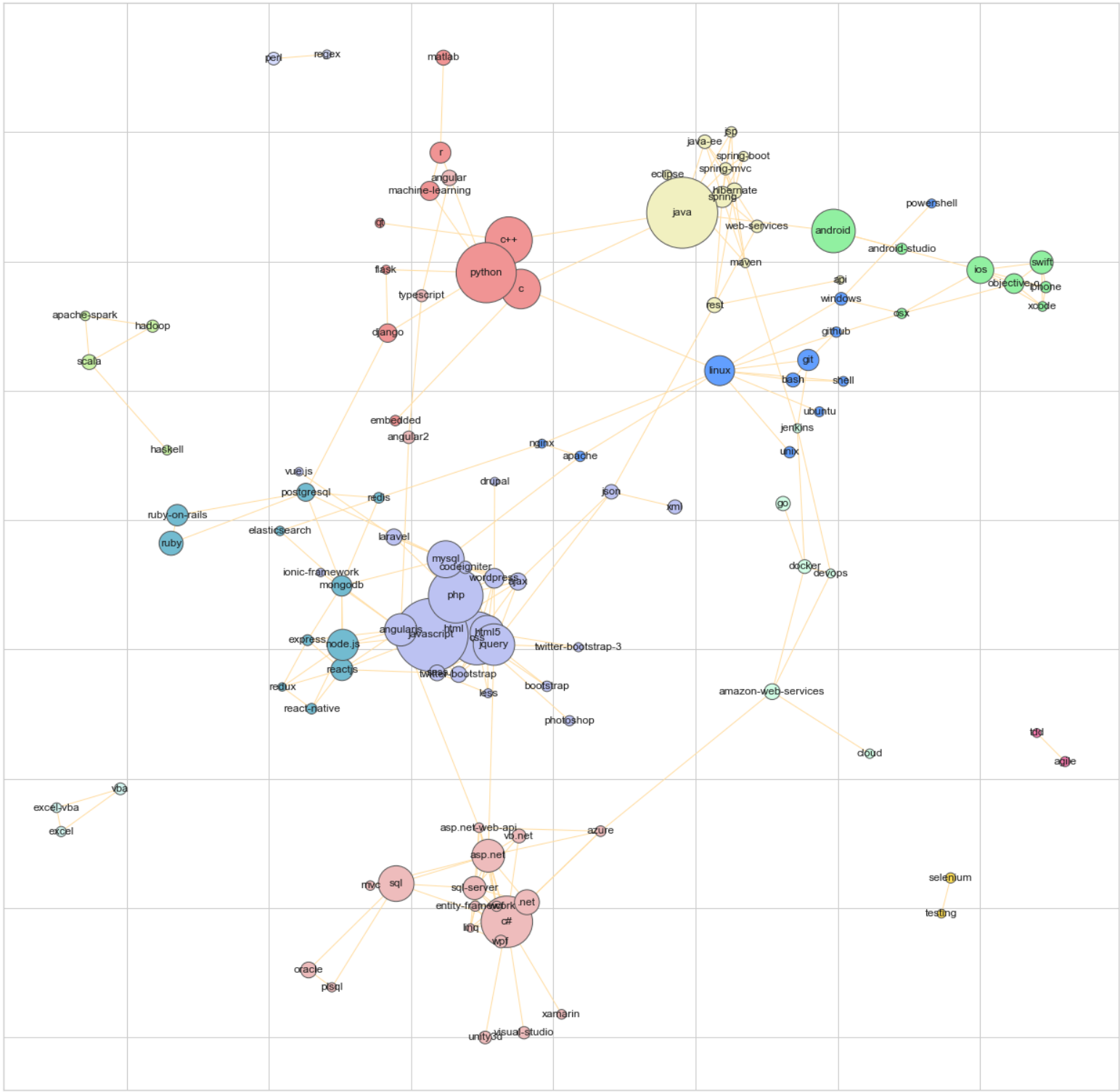


Figure 2: Tag Network Graph

Network Data Statistics

Metric	Number
Number of nodes:	115
Number of edges	245
Average Degree	4.2609

After building the network the first thing to check is the number of nodes and edges it consists of.
Degree of a node in an undirected graph shows the number of nodes it's connected to. Average degree is the average of all node's degree.
Stack-overflow tags have an average degree of 4.26 which indicates that on average tags are connected to four other tags.
115 nodes mean there are 115 unique tags which are connected among themselves with 245 edges.

Social Network Analysis Matrix

Connectivity

A network is connected if there is a path between every pair of vertices. But this tag network is not connected, which means there are some isolated nodes or isolated subgraphs. A connected component is the maximal connected subgraph of a graph. In the tag network we have 6 unique connected components.

```
Number of components in Tag Network: 6
```

Clustering Coefficient

The local clustering coefficient of a node is the fraction of its direct neighbours that are themselves direct neighbours (i.e., directly connected); in a social network framework, for example, it would be the fraction of one’s friends that are also friends themselves. The average clustering coefficient of the whole graph is simply the average of the nodes’ local clustering coefficients. By definition, both coefficients lie between 0 and 1.

```
Tag network clustering coefficient: 0.46024448980970717
```

Network density

This is simply the ratio of actual edges in the network to all possible edges in the network. In an undirected network like this one, there could be a single edge between any two nodes, but as you saw in the visualisation, only a few of those possible edges are actually present. Network density gives you a quick sense of how closely knit your network is.

Network density: 0.03737604881769641

The density of our Tag network is approximately 0.03737. On a scale of 0 to 1, not a very dense network, which comports with what you can see in the visualisation. A 0 would mean that there are no connections

at all, and a 1 would indicate that all possible edges are present (a perfectly connected network): this tag network is on the lower end of that scale, but still far from 0.

Shortest Path

A shortest path measurement calculates the shortest possible series of nodes and edges that stand between any two nodes, something hard to see in large network visualizations.

```
Shortest path between sql and python:
['sql',   'asp.net',   'jquery',   'mysql',   'postgresql',   'django',
'python']
Length of that path: 6
```

Shortest path will be a list of the nodes that includes the path belween “source” (sql), the “target” (python), and the nodes between them. In this case, we can see that jquery is on the shortest path between them. Since jQuery and asp.net is also a hub (see degree centrality, below) with many connections, we might suppose that several shortest paths run through him as a mediator.

```
Shortest path between c and sql:
['mongodb', 'postgresql', 'django', 'python', 'machine-learning']
Length of that path: 4
```

Here we are observing shortest path between mongodb and machine learning, so result also make sense as framework Django (python based framework) is used to develop machine learning application which uses mongodb as database.

Diameter

There are many network metrics derived from shortest path lengths. One such measure is diameter, which is the longest of all shortest paths. After calculating all shortest paths between every possible pair of nodes in the network, diameter is the length of the path between the two nodes that are furthest apart. The measure is designed to give you a sense of the network’s overall size, the distance from one end of the network to another.

But as we know the Tag Network is not connected and have 6 unique connected components, Because there are some nodes that have no path at all to others, it is impossible to find all of the shortest paths. Since there is no shortest path between nodes of one component and nodes of another, finding the largest component and then calculating diameter on that component alone.

```
Network diameter of largest component: 10
Average shortest path length: 4.507862550960978
```

This component has a diameter of 10 — meaning this is the “widest” number of connections between any two tags. The average path length is just around four. This means that, on average, any two languages/ technologies tags are separated by four edges. The above figures give a measure of the “size” of the network.

Network Visualisation of largest component in the Tag network¶

Since we took the largest component, we can assume there is no larger diameter for the other components. Therefore this figure is a good stand in for the diameter of the whole Graph. The network diameter of this network's largest component is 10: there is a path length of 10 between the two farthest-apart nodes in the network.

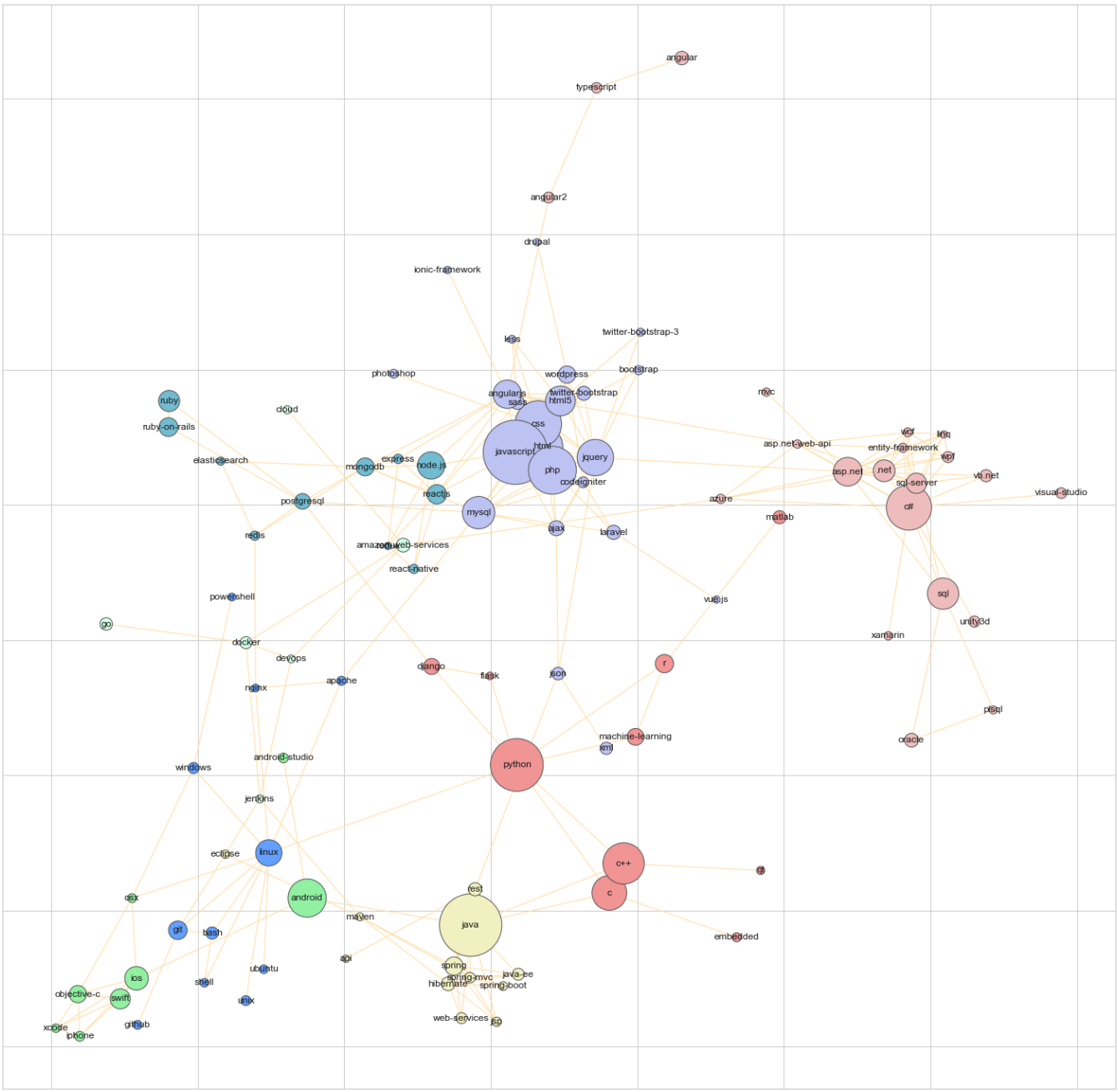


Figure 3: Tag Network of Largest Component

Visualising Programming Language Network

Since it's possible to draw the subgraph of a graph, a subgraph containing the nodes for the programming languages only can also be visualised. In the visualisation it's possible to see the different clusters for each programming language and familiar patterns like android with java or embedded systems with C and C++. These are tags that are more likely to be used by a developer on her or his Developer stories, We can see that what are the other tags which are linked to below popular technologies/language.

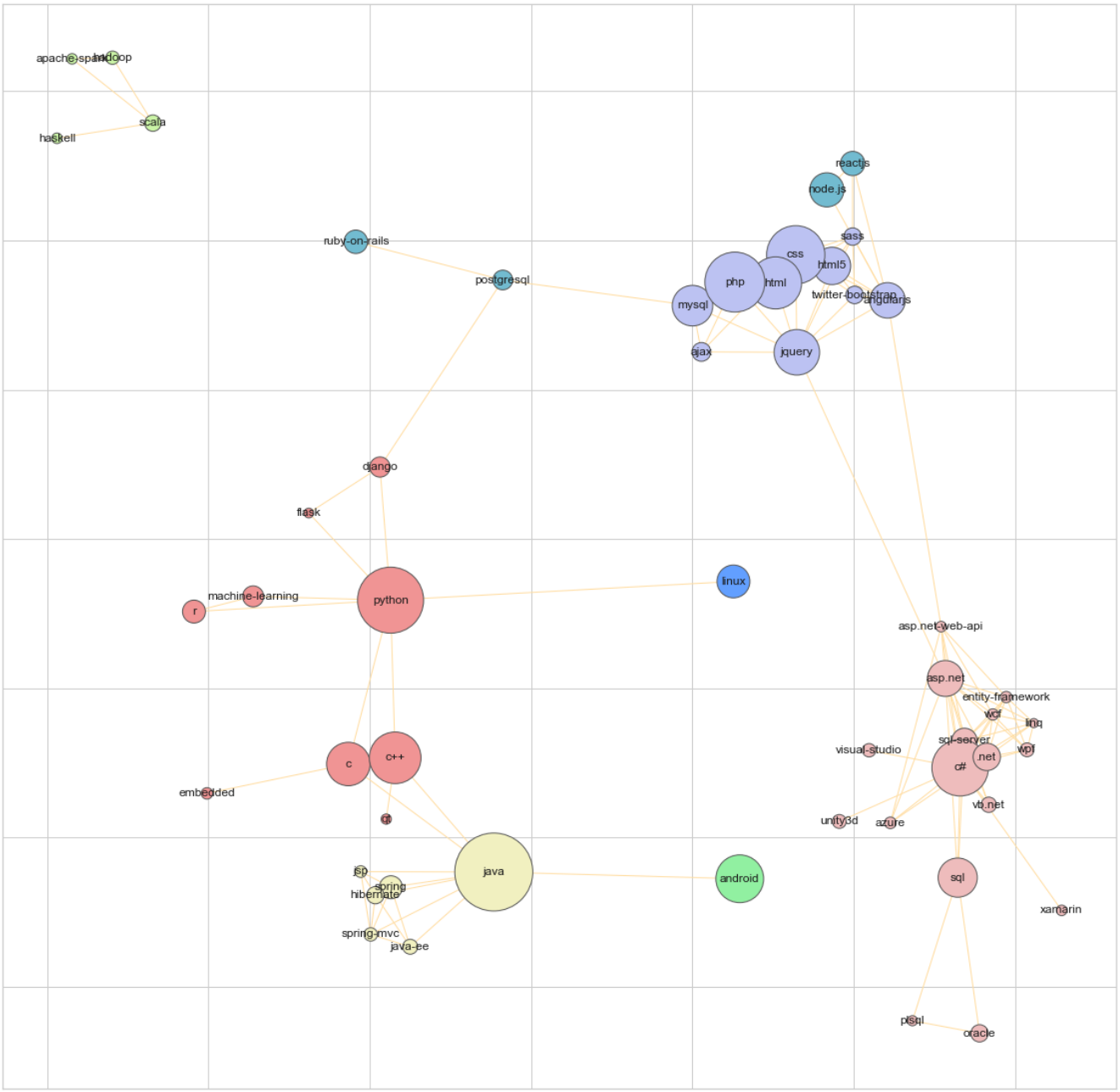


Figure 4: Programming Language Network

Node Importance: Centrality Measures

After getting some basic measures of the entire network structure, a good next step is to find which nodes are the most important ones in our network. In network analysis, measures of the importance of nodes are referred to as centrality measures. Because there are many ways of approaching the question “Which nodes are the most important?” there are many different ways of calculating centrality. Here we have measured four of the most common centrality measures: degree, betweenness centrality, closeness centrality and eigenvector centrality.

Degree Centrality

Degree centrality is one of the easiest to calculate. The degree centrality of a node is simply its degree—the number of edges it has. The higher the degree, the more central the node is. The nodes with the highest degree in a social network are the people who know the most people. These nodes are often referred to as hubs, and calculating degree is the quickest way of identifying hubs. This can be an effective measure, since many nodes with high degrees also have high centrality by other measures.

Top 10 node-tags with highest degree centrality:

Node-Tags	Degree Centrality
jquery	0.1403
css	0.1228
c#	0.1228
asp.net	0.1140
angularjs	0.1140
javascript	0.1052
mysql	0.0964
html5	0.0877
php	0.0877
linux	0.0877

Comparing their Degree Centrality ranking to the frequency ranking of the corresponding tags, we can see that, 'Java', 'JavaScript' tag have highest frequency as compare to others, but 'Java' is not even in the list of top 10 nodes with high degree value and even 'JavaScript' stands in 6th place. Hence this help us to understand that frequency of anything in the network doesn't mean it has high degree or it is connected to many other nodes. It might be chances that particular tag occurs multiple times alone in the developer stories due to which it has very high frequency in the network. Same is true for tags like 'java' & 'javascript'.

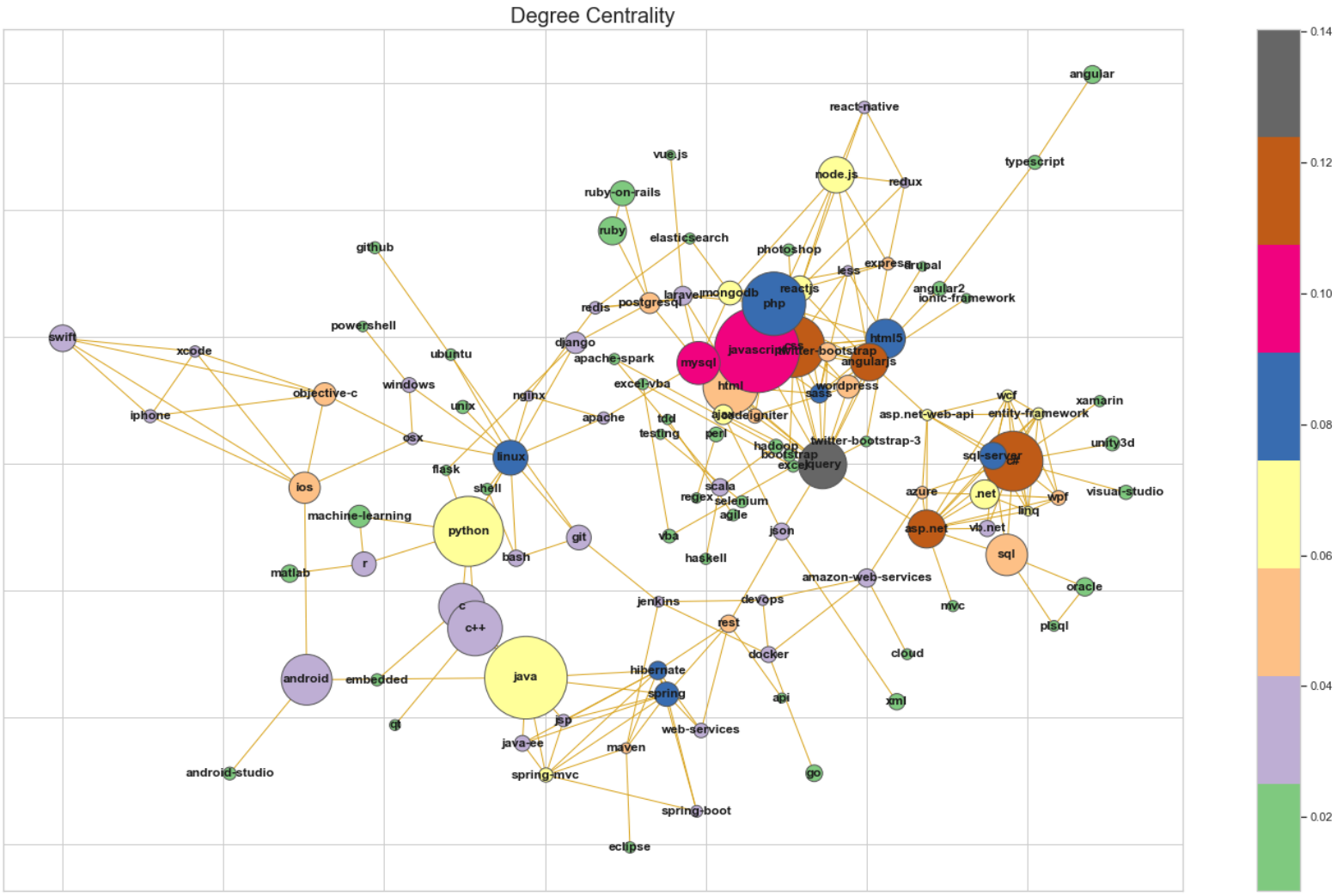


Figure 5: Degree Centrality Tag Network

Closeness Centrality

Closeness centrality is a way of detecting nodes that are able to spread information very efficiently through a graph.The closeness centrality of a node measures its average farness (inverse distance) to all other nodes. Nodes with a high closeness score have the shortest distances to all other nodes.

Closeness centrality can help find good 'broadcasters', but in a highly-connected network, you will often find all nodes have a similar score. What may be more useful is using Closeness to find influencers in a single cluster. In our network jquery have the highest closeness centrality. This tells us that jquery is directly connected to some of the important tags in our network. And if will see the network that jquery is close to asp.net that means it is directly connected to asp.net on one side and html, javascript on other side.

Top 10 node-tags with highest closeness centrality:

Node-Tags	Closeness Centrality
jquery	0.2895
mysql	0.2778
ajax	0.2586
css	0.2578
javascript	0.2571
angularjs	0.2571
apache	0.2549
php	0.2513
html	0.2471
asp.net	0.2465

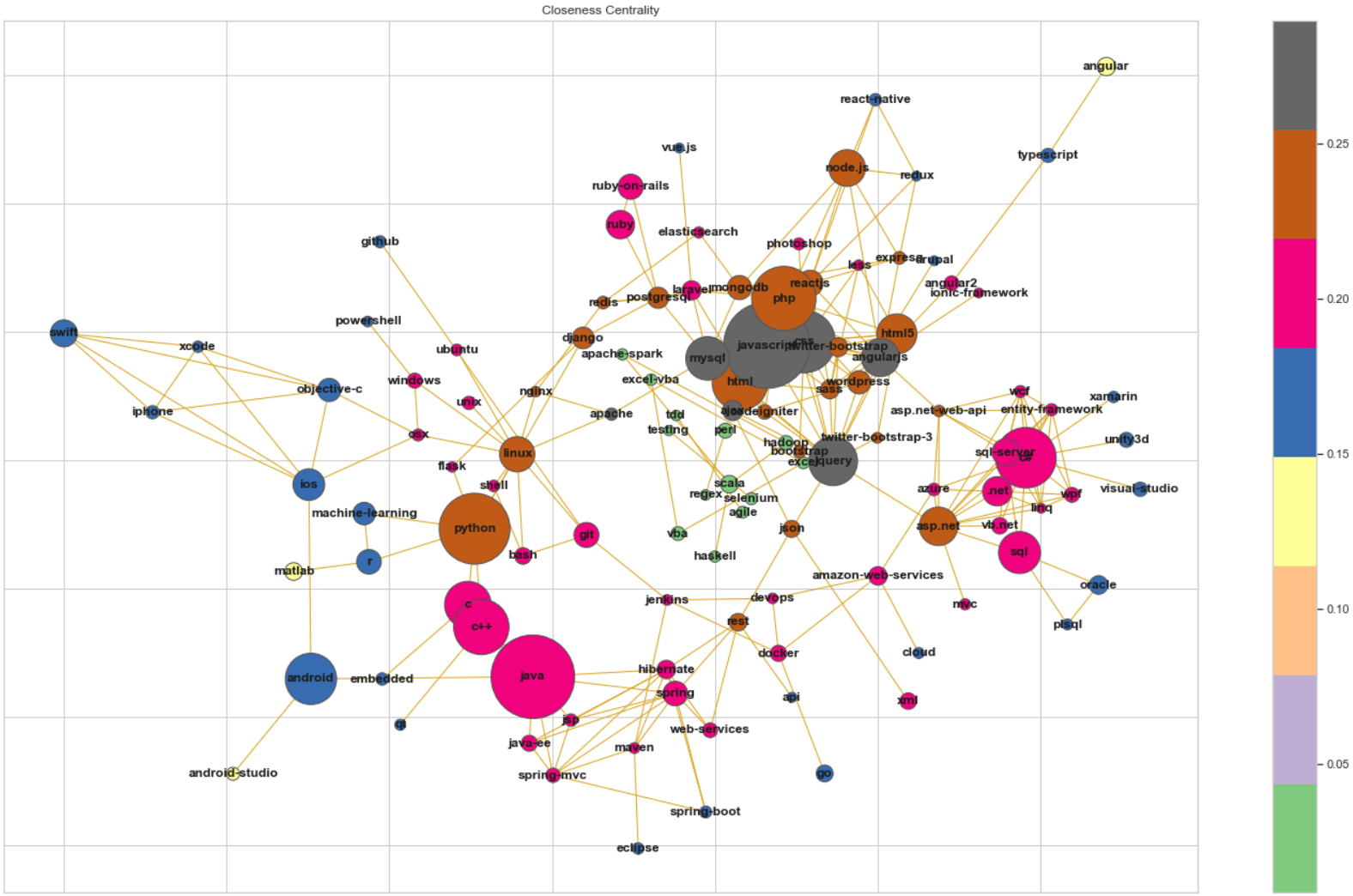


Figure 6: Closeness Centrality Tag Network

Eigen Vector Centrality:

Eigenvector centrality is a basic extension of degree centrality, which defines centrality of a node as proportional to its neighbours importance. When we sum up all connections of a node, not all neighbours are equally important.

Observing Eigen vector centrality values, even here we have jQuery in the first position, Which implies that jQuery have direct connection to many other important node which is already popular in our network i.e, have many connection. This can be visualised from graph that jQuery is directly connected to javascript, asp.net and html which are already very popular technology tag in IT world so in our analysis as well they have high number of connections with other node-tag in the network, but one interesting thing we observed about linux node-tag that it is in top 10 list of degree measure but the eigenvector Centrality value is too low (close to 0), so this measure gives you more insight information about important node as compared to degree measure.

Also there are some important nodes which are very popular in real world(IT world) & even have high frequency but the value of Eigen vector centrality is too low close to zero (can be observed clearly from below graph). Like 'Python', 'Java' & 'Android' these are some of the most popular actors of our network which have very high frequency which tells us it occurs many times in the developer stories, But these nodes doesn't have good and popular neighbour in the network due to which importance of these actors is less and have low Eigen vector centrality value. So in this metrics not only connection with neighbour matters but even what type of connection neighbour have in the network also matters.

Top 10 node-tags with highest Eigenvector centrality:

Node-Tags	Eigenvector Centrality
jquery	0.3657
css	0.3387
javascript	0.3256
html5	0.2681
php	0.2653
angularjs	0.2652
sass	0.2520
mysql	0.2393
twitter-bootstrap	0.2070
html	0.2038

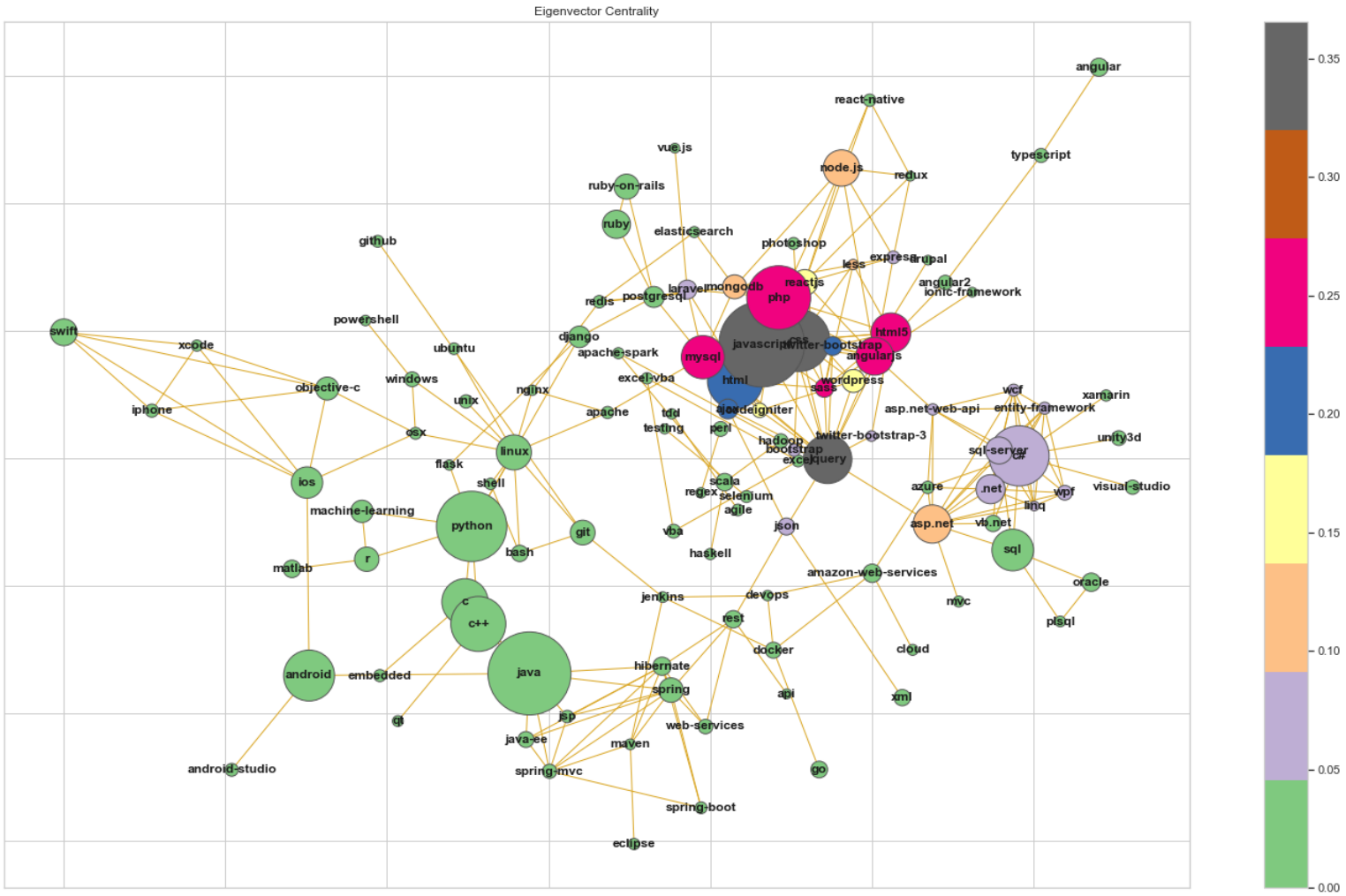


Figure 7: Eigenvector Centrality Tag Network

Betweenness Centrality:

Betweenness centrality is a way of detecting the amount of influence a node has over the flow of information in a graph. It is often used to find nodes that serve as a bridge from one part of a graph to another.

Betweenness centrality is a bit different from the other three measures in that it doesn't care about the number of edges any one node or set of nodes has. Betweenness centrality, which is also expressed on a scale of 0 to 1, is fairly good at finding nodes that connect two otherwise disparate parts of a network. If we're the only thing connecting two clusters, every communication between those clusters has to pass through us. In contrast to a hub, this sort of node is often referred to as a broker. Betweenness centrality is not the only way of finding brokerage (and other methods are more systematic), but it's a quick way of giving you a sense of which nodes are important not because they have lots of connections themselves but because they stand between groups, giving the network connectivity and cohesion.

From the below graph and by seeing the betweenness centrality of top 10 technology we observe that technology like "Linux" & "MySql" also play an important role in the network. If will see the degree centrality of linux it won't be high because it is not connected to many nodes. But it is very important node in the network as it influence the flow around the graph. If will remove linux node-tag from the network, there will not be any other tags in the network which will link two major sub-networks i.e, Python and Javascript.

So when we compare the values with degree and closeness centrality values, important nodes like 'JavaScript', 'html', 'css', 'java' have low score. This is because they are don't stand between two groups in our network. they are not helpful when it comes to act as a bridge between two different technologies. So this tells us an interesting thing that being popular and having high degree doesn't implies that those are important tags, In some scenario/situation having high betweenness will play a crucial role in the network. If will see the actors like 'jquery', 'mysql', 'json' these are some node-tags connecting two clusters, every communication between those clusters has to pass through them. Like 'json' node-tag stands between the whole cloud technologies ('web-services','rest') with Java based frameworks ('jquery' and 'ajax').

Interestingly enough, all the high betweenness node-tags shown above are somewhat "general" ones, in the sense that they are not related with specific software or hardware platforms. This makes perfect sense intuitively, and it demonstrates how the network structure can capture relations and characteristics in the data that are not easily or directly expressed in simple aggregate measures, such as the frequency of occurrence.

Top 10 node-tags with highest betweenness centrality:

Node-Tags	Betweenness Centrality
jquery	0.2555
linux	0.2084
mysql	0.1976
asp.net	0.1740
apache	0.1308
json	0.1231
angularjs	0.1228
rest	0.1137
python	0.1101
postgresql	0.0876

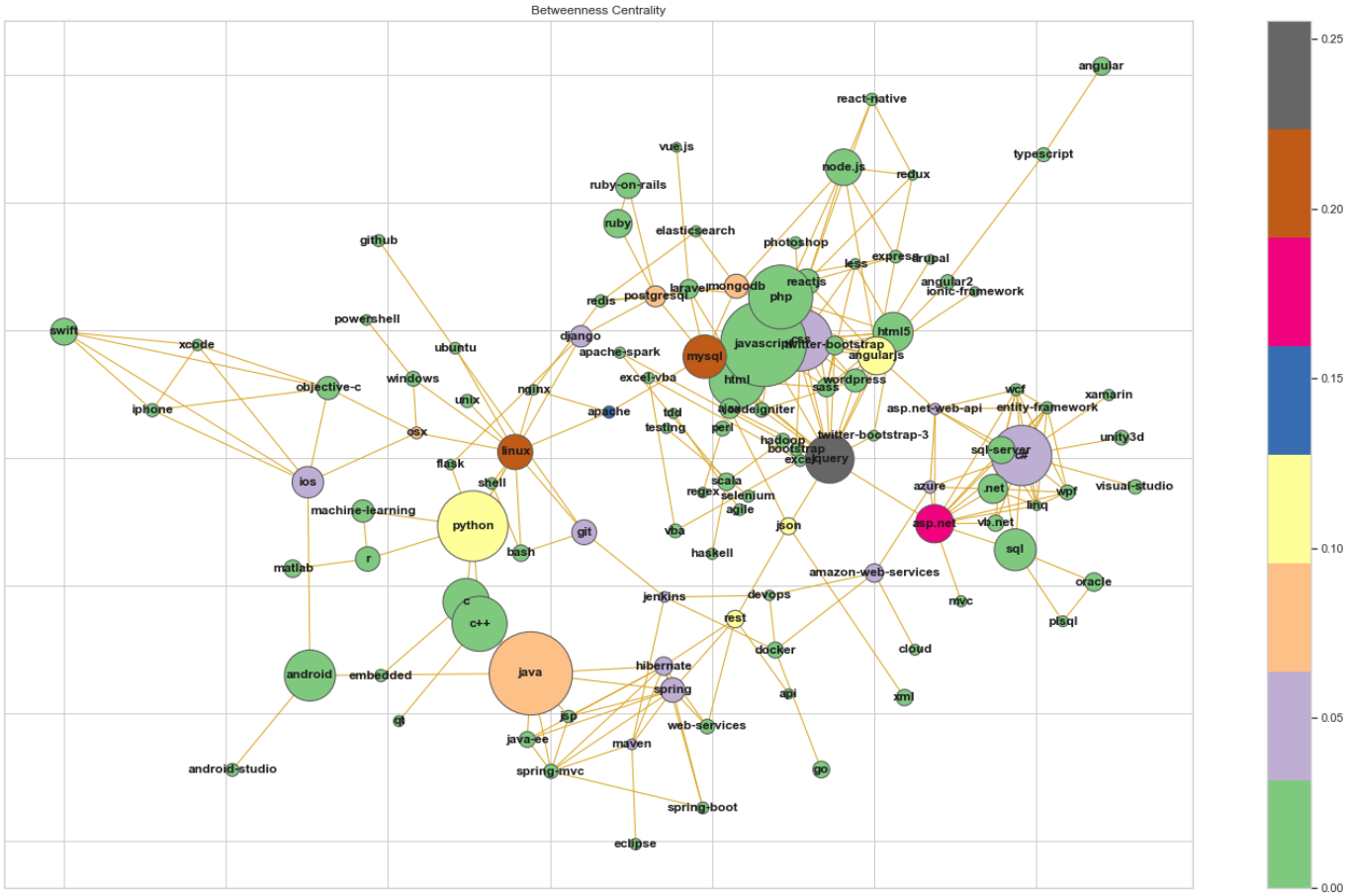


Figure 8: Betweenness Centrality Tag Network

Cliques

In general we consider cliques as groups of people who are closely connected to each other but not connected to people outside the group. In network theory a clique is defined as a maximal complete subgraph of a graph where each node is connected to all the other nodes. The word 'maximal' means that if we add another node to the clique the clique will cease to be a clique. We can also extract all the cliques from the tag network.

Number of cliques in Tag Network: 89

Language Specific Ego Network And Cliques

For each programming language there's a tag in the network. E.g 'python' will refer to the python language. So we can check the cliques that contains that node. We can also visualise the ego network for a node. Ego network for a node is the subgraph containing that node and all its neighbours with a specified depth range.

Here we can see that the ego network for python with radius 2, which means that we get the subgraph containing python and all it's direct neighbours which are 1 edge away from python and also the nodes which are 2 edge away from python.

Ego networks can be used for checking shortest paths or generally conducting analysis of who is connected to whom, but cliques are helpful because it shows us the data in a more granular way.

```
print(nx.ego_graph(G,'python',radius=2).nodes())
```

```
['windows', 'bash', 'r', 'c', 'linux', 'ubuntu', 'machine-learning',
'nginx', 'postgresql', 'flask', 'matlab', 'django', 'qt', 'apache',
'osx', 'c++', 'java', 'shell', 'python', 'embedded', 'git', 'unix']
```

```
nx.algorithms.clique.cliques_containing_node(G,"python")
```

```
[['c', 'c++', 'python'],
 ['linux', 'python'],
 ['django', 'python', 'flask'],
 ['r', 'machine-learning', 'python']]
```

Python participates in 4 different cliques, one for web development with django and flask, one for open source development presumably which is connected to linux. One for machine learning where it's adjacent to R. I think the fourth one is for porting python and C/C++ back and forth.

```
nx.algorithms.clique.cliques_containing_node(G,"java")
```

```
[['java-ee', 'java', 'spring-mvc', 'hibernate', 'spring'],
 ['c', 'c++', 'java'],
 ['android', 'java'],
 ['spring-mvc', 'spring', 'hibernate', 'java', 'jsp']]
```

Java participates in 4 different cliques, two are for java application development frameworks (spring, hibernate, 'spring-mvc'), one for mobile development with android, One for porting python and C/C++ back and forth.
Similarly we can see for javascript below:

```
nx.algorithms.clique.cliques_containing_node(G,"javascript")
```

```
[['node.js', 'reactjs', 'javascript', 'angularjs'],
 ['reactjs', 'sass', 'angularjs', 'javascript'],
 ['jquery',
  'css',
  'javascript',
  'twitter-bootstrap',
  'sass',
  'html5',
  'angularjs'],
 ['jquery', 'css', 'javascript', 'php', 'html5'],
 ['jquery', 'css', 'javascript', 'php', 'mysql', 'html'],
 ['jquery', 'css', 'javascript', 'php', 'mysql', 'ajax'],
 ['jquery', 'css', 'javascript', 'sass', 'html']]
```

Javascript participates in 7 different cliques, two are for java application development frameworks (spring, hibernate, 'spring-mvc'), one for mobile development with android, One for porting python and C/C++ back and forth.

Visualize Maximal Clique

It's possible that visualizing the largest cliques will let us see some pattern in the data. We have 3 cliques of size 7 which are the biggest, however we've only taken the unique nodes while extracting the subgraphs, so we can see two different clusters containing javascript and .net related tags.

Metric	Number
Number of nodes:	15
Number of edges	49
Average Degree	6.5333

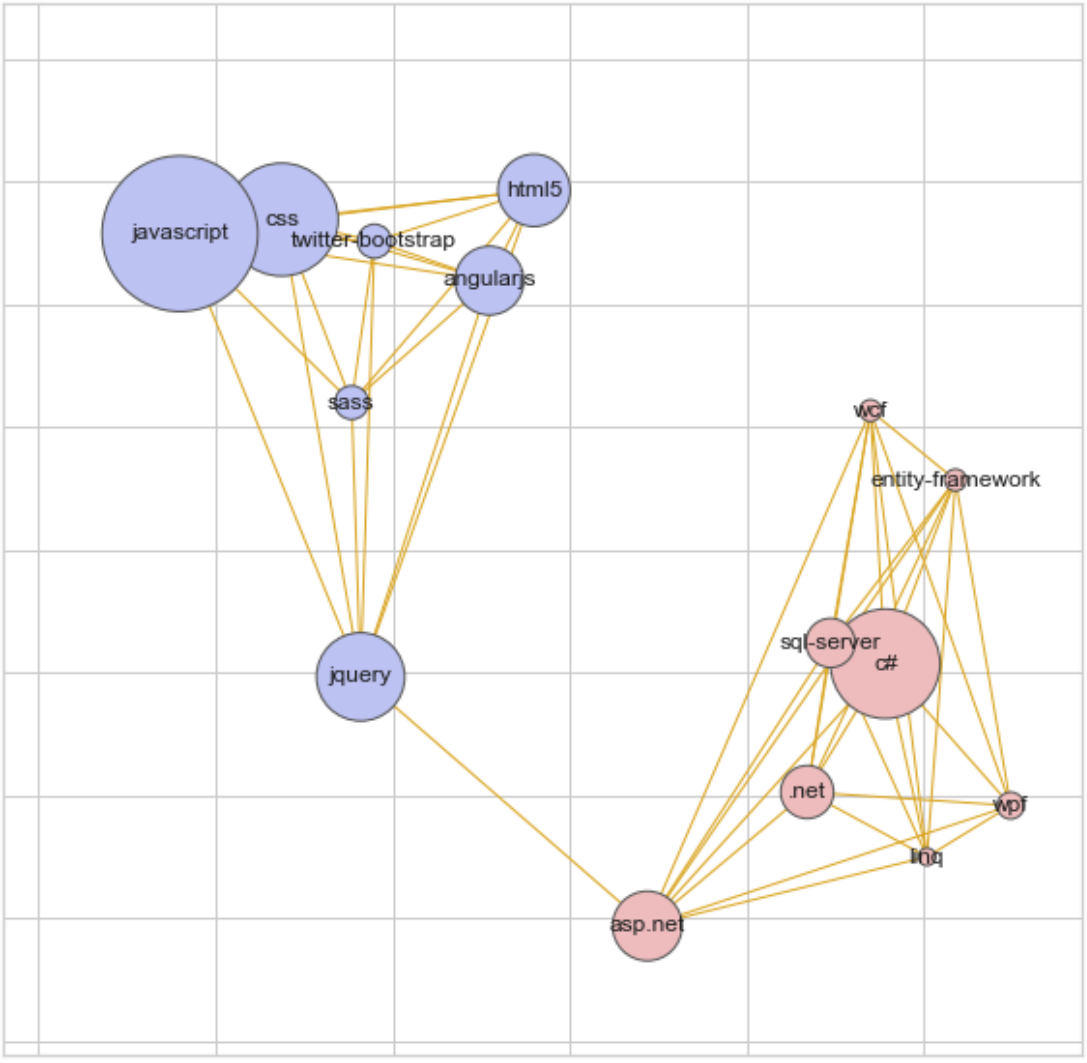


Figure 9: Maximal Clique (Two Clusters)

Community Detection

For those who use Facebook, our Facebook friends probably come from different aspects of our life. Some are your friends from college, others are your co-workers, and maybe some old friends from your hometown. Because your friends can be broken down into different groups like this, you may wonder if we could identify these different communities in your social network. The answer is yes! Using community detection algorithms, we can break down a social network into different potentially overlapping communities.

The criteria for finding good communities is similar to that for finding good clusters. We want to maximize intra-community edges while minimizing inter-community edges. Formally, the algorithm tries to maximize the modularity of network, or the fraction of edges that fall within the community minus the expected fraction of edges if the edges were distributed by random. Good communities should have a high number of intra-community edges, so by maximizing the modularity, we detect dense communities that have a high fraction of intra-community edges.

Girvan Newman: Edge Betweenness Algorithm

The Girvan Newman technique for the detection and analysis of community structure depends upon the iterative elimination of edges with the highest number of the shortest paths that pass through them. By getting rid off the edges, the network breaks down into smaller networks, i.e. communities.

The algorithm, as the name suggests, is introduced by Girvan & Newman. The idea was to find which edges in a network occur most frequently between other pairs of nodes by finding edges betweenness. The edges joining communities are then expected to have high edge betweenness. The underlying community structure of the network will be much fine-grained once we eliminate edges with high edge betweenness. For the removal of each edge, the calculation of edge betweenness is O(EN); therefore, this algorithm's time complexity is O(E^2N). We can express Girvan-Newman algorithm in the following procedure:

1) Calculate edge betweenness for every edge in the graph.
2) Remove the edge with highest edge betweenness.
3) Calculate edge betweenness for remaining edges.
4) Repeat steps 2–4 until all edges are removed.

In order to calculate edge betweenness it is necessary to find all shortest paths in the graph. The algorithm starts with one vertex, calculates edge weights for paths going through that vertex, and then repeats it for every vertex in the graph and sums the weights for every edge.

We have an affordably small network with not too complex clustering. As our network contain 115 nodes of many users with less than 415 connections, Girvan Newman Edge Betweenness algorithm should work fine for detecting communities.

```
Modularity: 0.3716711339194898
Total number of Communities = 7
```

Community Num	Members
0	ajax | amazon-web-services | android | android-studio | angular | angular2 | angularjs | apache | api | bash | bootstrap | c | c++ | cloud | codeigniter | css | devops | django | docker | drupal | eclipse | elasticsearch | embedded | express | flask | git | github | go | hibernate | html | html5 | ionic-framework | ios | iphone | java | java-ee | javascript | jenkins | jquery | json | jsp | laravel | less | linux | machine-learning | matlab | maven | mongodb | mysql | nginx | node.js | objective-c | osx | photoshop | php | postgresql | powershell | python | qt | r | react-native | reactjs | redis | redux | rest | ruby | ruby-on-rails | sass | shell | spring | spring-boot | spring-mvc | swift | twitter-bootstrap | twitter-bootstrap-3 | typescript | ubuntu | unix | vue.js | web-services | windows | wordpress | xcode | xml
1	.net | asp.net | asp.net-web-api | azure | c# | entity-framework | linq | mvc | oracle | plsql | sql | sql-server | unity3d | vb.net | visual-studio | wcf | wpf | xamarin
2	apache-spark | hadoop | haskell | scala
3	excel | excel-vba | vba
4	agile | tdd
5	selenium | testing
6	perl | regex

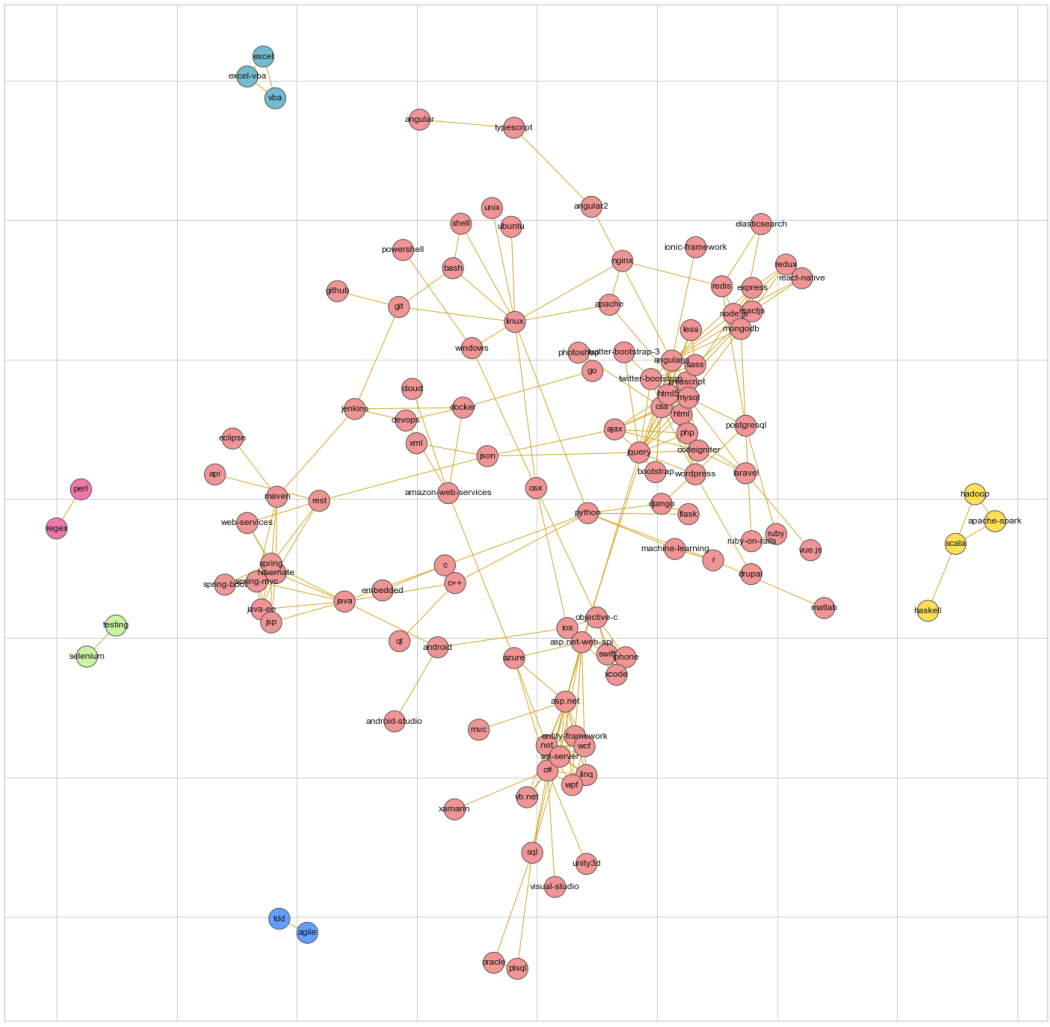


Figure 10: Girvan Newman Algorithm Community Structure

When we observe the communities created by Girvan using edge betweenness we found some interesting but few weird thing as well. Community 5, 6, 2 are some communities which have same members and occurred separately in Louvain community detection as well and looks related, as all the Big Data related technologies makes a separate community because developers use those tags together very frequently. Similarly selenium and testing also makes which is also acceptable as mentioned earlier "selenium" is a tool used by many testers nowadays. But when observe the members of community 0, here all the front-end('css','html'), backend ('java','angularjs','python','c'), framework('Django','flask','ionic') technologies appears together even Applications development technologies, cloud based technologies ('amazon-webservice') are also present in this communities. Even we can see all the different computer and mobile OS technologies in this community('android','ios','linux','unix'). There might be be chances few person in the survey work as an full-stack developers and uses most of these technologies together but there won't be much scenario like this its hard to believe that all these technology tag are used so often that they have dense network among themselves and hence appearing in the same community.

Community Structure: Louvain Algorithm

The Louvain method for community detection is an algorithm for detecting communities in networks. It maximises a modularity score for each community, where the modularity quantifies the quality of an assignment of nodes to communities. This means evaluating how much more densely connected the nodes within a community are, compared to how connected they would be in a random network.

This algorithm works on the principle of partitioning a network into mutually exclusive communities such that the number of edges across different communities is significantly less than expectation, whereas the number of edges within each community is significantly greater than expectation. The Louvain algorithm is one of the most widely used for identifying communities due its speed and high modularity. Modularity values can span from -1 to 1, and the higher the value, the better the community structure that is formed.

```
Modularity:  0.7503995634499011
Total number of Communities= 14
```

Community_Num	Members
0	html | css | html5 | javascript | jquery | php | mysql | less | sass | ajax | angularjs | laravel | json | xml | wordpress | codeigniter | twitter-bootstrap | ionic-framework | vue.js | drupal | bootstrap | twitter-bootstrap-3 | photoshop
1	hibernate | spring | c | c++ | spring-mvc | spring-boot | java | java-ee | maven | jsp | web-services | rest | qt | embedded | eclipse | api
2	ruby | ruby-on-rails | redux | reactjs | react-native | express | node.js | mongodb | redis | elasticsearch | postgresql
3	ios | swift | objective-c | iphone | android | xcode | android-studio | osx
4	asp.net | c# | .net | sql-server | entity-framework | linq | wcf | wpf | asp.net-web-api | plsql | oracle | visual-studio | sql | vb.net | unity3d | xamarin | azure | mvc
5	hadoop | apache-spark | scala | haskell
6	github | git | apache | nginx | bash | linux | windows | ubuntu | unix | shell | powershell
7	excel | excel-vba | vba
8	django | python | flask | machine-learning | r | matlab
9	angular2 | typescript | angular
10	tdd | agile
11	testing | selenium
12	jenkins | docker | amazon-web-services | go | devops | cloud
13	regex | perl

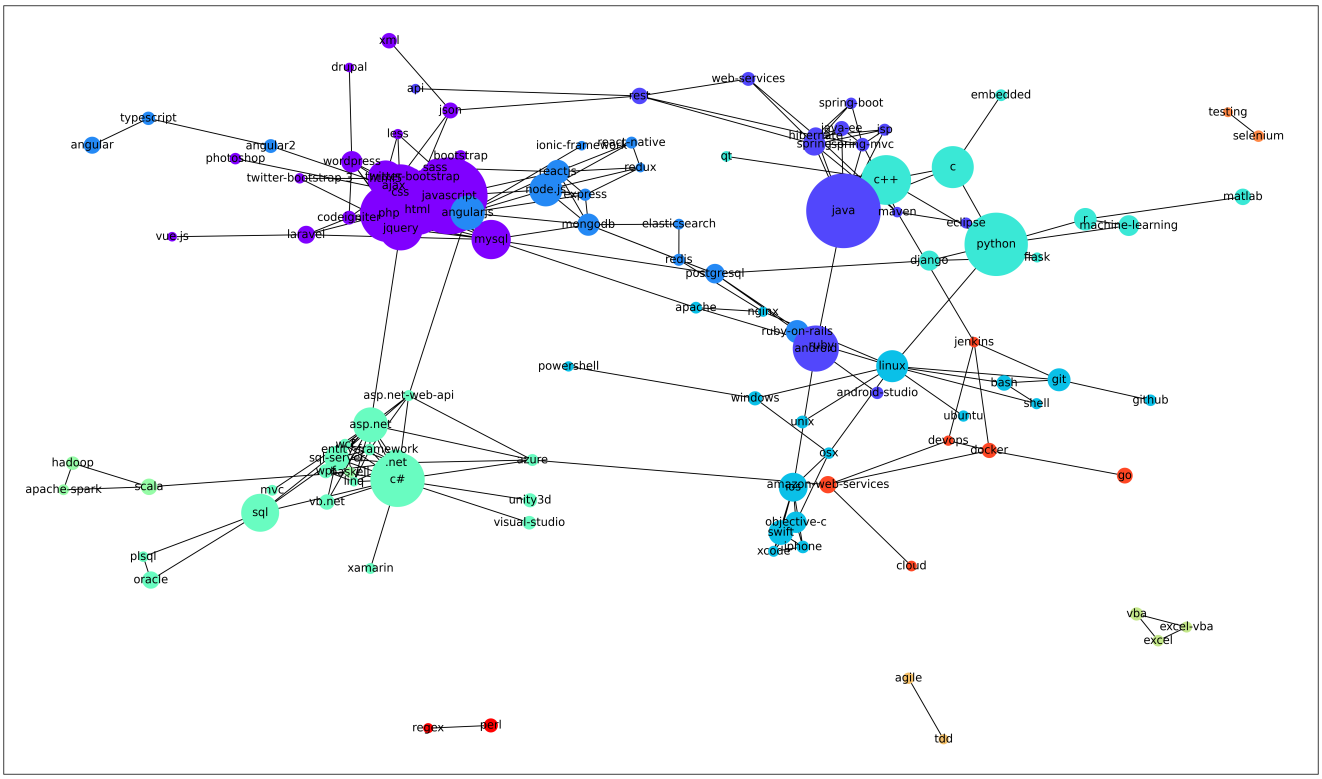


Figure 11: Louvain Algorithm Community Structure

We performed the Louvain algorithm on this dataset, and the results are given in Figure 3. As we see, we have 14 communities, and a modularity of 0.751, which is a pretty good solution. Also we have noticed there are few communities which have most popular languages are in those communities. For example JavaScript, jquery, html and CSS are all in community 0. By seeing the members of all the communities below we observe many interesting thing. We got some communities which looks legitimate by intuition as well as these technology tags are related to each other and they are often comes together like 'hadoop', 'apache-spark', 'scala' and 'haskell' groups together to make a community named 6 in our case, As all of these are Big data technologies and framework so its possible that developer used these tags together. Surprisingly Tags like html, css, javascript, mysql & word-press are making community with photoshop and bootstrap this help us is understanding that people are using these tags together quite frequently. So there will be definitely dense connection between these nodes in the intra community as compare to sparse connections between inter community.

While observing community 3, it also seems acceptable as all the IOS development technologies are appearing together in this community.

If will observe the member of few other communities which is detected by the Louvain is making sense intuitionally as well. Like 'selenium' is making a community with 'testing' which seems true as it is an automation testing tool used by different companies. Similarly 'jenkins', 'docker', 'devops' make a community with amazon-web-services, As all of these are software development & deployment tool used together most of the times in different cloud services.

By seeing all the communities and the members of that community we can say that what are the tags that occur most often with a few important languages like C#, C++, Java, JavaScript, and Python in developers stories and what are the other technologies which appears with these important languages.

Discussion

After identifying the top 10 nodes according to all of the centrality measures (they overlap a lot, presumably because it's just a co-occurance network of tags and undirected) we can conclude that “jquery” is most important node-tag in our Tag Network.

Also among all the centrality measure betweenness centrality and eigenvector centrality measures result can be assumed to proper based on our intuition, as it is making more sense when we compare them with other measures.

Another thing we can notice in this network is that some technologies act as bridges between tech ecosystems. Python, one of the most commonly used languages on Developer Stories, connects to the front-end cluster (through Django), to a Linux/systems administration cluster, to a C/C++/embedded cluster, and to R and machine learning. We see time and again how unique a language Python is becoming in today’s technology landscape. Java, git, and JSON are other “bridge” technologies that connect parts of this network.

Also we observed and compared the community detection methods we can clearly conclude that communities generated by Louvain is more meaningful and making sense as it separate the technologies in different categories as per their uses. Hence for our network we can say that Louvain is more effective and ideal as compare to Girvan algorithm for detecting communities.

References

http://snap.stanford.edu/class/cs224w-2010/slides/14-communities_annot.pdf

https://medium.com/@96mehakmohammad/community-detection-in-social-networks-through-girvan-newman-algorithm-78f303912907

https://dl.acm.org/doi/pdf/10.1145/2961111.2962588

https://www.nodalpoint.com/wp-content/uploads/2017/07/CI_Tsatsoulis_Stackoverflow_Tags_Analysis.pdf

Code can be found on below link:
https://colab.research.google.com/drive/1hSttgV0s4TC3AXOL4TdQPECZHlxUk5cf#scrollTo=EKFShd0CB6H4