Netzerkanalyse mit GitHub

Semesterarbeit 2

EDS-Einführung in Data Science

Klasse: BSc INF-P-IN010, BE1, HS20/21
Dozent: Dr. Tim vor der Brück
Autor: Sandro Bürki, Michael Friderich
Datum: 09.10.2020

Einleitung ¶

In der Entwicklung von Software steht für viele Firmen die effizienz an erster Stelle. Oftmals stellt sich hierbei die Frage, ob sich der Einsatz von spezialisierten Fachpersonen im Vergleich zu Entwicklern mit breitem Know-How von verschiedenen Projekten positiv auf die Effizienz im Team auswirkt. Viele Projektmethodiken oder Projektframeworks (wie beispielsweise SCRUM) setzen dabei auf eine Fokussierung der Arbeitskraft, anstelle eines breiten Einsatzgebietes.

Diese Mentalität sollte sich somit in einer Netzwerkanalyse zeigen. Die Repositories von Firmen sollten primär von spezifischen Personen gepflegt werden, welche sich nicht weiter stark an anderen Projekten betreffen. Mit dieser Frage beschäftigt sich die vorliegende Arbeit.

Es gilt herauszufinden, wie stark sich die Repositories und ihre Contributors *segregieren*. Hierbei werden die Centrality-Measurements miteinander verglichen. Sollte sich unsere These bestätigen, dass diese Segregierung eintritt, sollten folgende Eigenschaften gelten:

1. Die Anzahl an Edges liegt nicht weit über der Anzahl an Contributors.
2. Die Betweenness-Centrality der meisten Benutzergeht ist 0.
3. Die Betweenness-Centrality der restlichen Benutzer geht gegen 0.

Vorgehen

Der Code ist weitgehen in zwei Teile aufgeteilt. Zuerst weden die Daten von GitHub abgerufen. Anschliessend wird von den Daten ein Backup in ein .json-File erstellt, da die Abfrage über viele Repositories viel Zeit in anspruch nehmen kann, und somit die wiederverwendbarkeit nach einem Neustart des Jupyter-Notebooks vereinfacht wird.

Im zweiten Teil des Codes werden anhand der Daten findings gezogen.

Daten abrufen

Zuerst werden die Informationen zu einer Firma aus GitHub geladen. Hierbei muss die entsprechende Firma und der Access-Token gesetzt werden.

In [1]:
```
# HTTP request to GitHub's API
import json
from math import ceil
import requests
from IPython.core.display import display
from ipywidgets import HTML

ACCESS_TOKEN = '<INSERT-ACCESS-TOKEN-HERE>'

organisationName = "Google"

organisationUrlTemplate = 'https://api.github.com/orgs/{}'
organisationUrl = organisationUrlTemplate.format(organisationName)

response = requests.get(organisationUrl)
organisation = response.json()

organisationRepoCount = organisation["public_repos"]
print("number of repos found: {}".format(organisationRepoCount))
```

```
number of repos found: 1870
```

Anschliessend werden alle Repositoriesder Firma geladen. Da jeweils nur 100 Repos pro Aufruf möglich sind, werden sie in entsprechenden Teilen zusammengefügt.

In [2]:
```
pageSize = 100

repositories = []
for page in range(0, ceil(organisationRepoCount / pageSize) + 1):

    organisationReposUrlTemplate = '{}?type=source&per_page=100&page={}&
    organisationReposUrl = organisationReposUrlTemplate.format(organisat
    response = requests.get(organisationReposUrl)
    repositories.extend(response.json())
    print("Repository page #{} loaded. ({} / {})".format(page, len(repos
```

```
Repository page #0 loaded. (100 / 1870)
Repository page #1 loaded. (200 / 1870)
Repository page #2 loaded. (300 / 1870)
Repository page #3 loaded. (400 / 1870)
Repository page #4 loaded. (500 / 1870)
Repository page #5 loaded. (600 / 1870)
Repository page #6 loaded. (700 / 1870)
Repository page #7 loaded. (800 / 1870)
Repository page #8 loaded. (900 / 1870)
Repository page #9 loaded. (1000 / 1870)
Repository page #10 loaded. (1100 / 1870)
Repository page #11 loaded. (1200 / 1870)
Repository page #12 loaded. (1300 / 1870)
Repository page #13 loaded. (1400 / 1870)
Repository page #14 loaded. (1500 / 1870)
Repository page #15 loaded. (1600 / 1870)
Repository page #16 loaded. (1700 / 1870)
Repository page #17 loaded. (1800 / 1870)
Repository page #18 loaded. (1900 / 1870)
Repository page #19 loaded. (1928 / 1870)
```

Zu den Repositories können nun die Informationen der Contributors geladen werden. Da dies je nach Firma viele Repositories sein können (im Beispiel von Google ca. 1900), kann dies einige Zeit in Anspruch nehmen.

In [3]:
```
print("Loading contributors of repositories. Depending on the size, this
numberOfRepositories = len(repositories)

for index, repository in enumerate(repositories):
    contributorUrlTemplate = "{}?per_page=100&page={}&access_token={}"
    repository["contributors"] = []
    page = 0
    while True:
        response = requests.get(contributorUrlTemplate.format(repository
        if response.status_code == 204:
            contributors = []
        else:
            contributors = response.json()

        page += 1
        if len(contributors) != 100:
            break

    if "message" in contributors:
        contributors = []

    repository['contributors'].extend(contributors)
    print("Contributors for repo #{} of {} loaded".format(index + 1, num
```

```
Loading contributors of repositories. Depending on the size, this m
ight take a while.
Contributors for repo #1 of 1928 loaded
Contributors for repo #2 of 1928 loaded
Contributors for repo #3 of 1928 loaded
Contributors for repo #4 of 1928 loaded
Contributors for repo #5 of 1928 loaded
Contributors for repo #6 of 1928 loaded
Contributors for repo #7 of 1928 loaded
Contributors for repo #8 of 1928 loaded
Contributors for repo #9 of 1928 loaded
Contributors for repo #10 of 1928 loaded
Contributors for repo #11 of 1928 loaded
Contributors for repo #12 of 1928 loaded
Contributors for repo #13 of 1928 loaded
Contributors for repo #14 of 1928 loaded
Contributors for repo #15 of 1928 loaded
Contributors for repo #16 of 1928 loaded
Contributors for repo #17 of 1928 loaded
Contributors for repo #18 of 1928 loaded
```

Für die einfache Wiederverwertung können hier BackUps der geladenen Daten erstellt werden.

In [4]:
```
with open('repositories_backup.json', 'w') as fp:
    json.dump(repositories, fp)
```

In [5]:
```
import json

with open('repositories_backup.json', 'r') as fp:
    repositories = json.load(fp)
```

GitHub bietet bei einem kostenlosen Plan nicht für beliebige Grössen der Repositories an, alle Contributors zu laden, weshalb hier die Fehlermeldungen einiger Repos entfernt werden.

In [6]:
```
for repo in repositories:
    if "message" in repo["contributors"]:
        repo["contributors"] = []
```

Und um schlussendlich die Anzahl an Knoten an die Aufgabenstellung anzupassen, werden hier nur Repositories berücksichtigt, welche über mehr als 80 Contributors umfassen. Die ungefilterte Liste würde ansonsten für Google ca. 12000 Knoten erstellen, was den Ramen dieser Arbeit sprengen würde.

In [7]:
```
largeRepos = []
for repo in repositories:
    if len(repo["contributors"]) > 80:
        largeRepos.append(repo)
```

Graphen erstellen

Im folgenden wird ein Graph der Daten erstellt. Ebenfalls wird daraus ein .gefx-File generiert, welches die Visualisierung in Gephi ermöglicht.

In [8]:
```
import networkx as nx

# Create a directed graph
g = nx.Graph()
for repository in largeRepos:
    g.add_node(repository["name"]+'(Repo)', type='Repo', color="green")
    contributors = repository["contributors"]
    for contributor in contributors:
        g.add_node(contributor["login"]+'(Contributor)', type='Contribut
        g.add_edge(contributor["login"], repository["name"]+'(Repo)', ty


# Write to File (open with Gephi)
nx.write_gexf(g, 'graph.gexf')
```

Messwerte berechnen

Nachfolgend werden die gestellten Fragen mathematisch beantwortet

In [14]:
```
from operator import itemgetter


# Check findings mathematically
bc = sorted(nx.betweenness_centrality(g).items(), key=itemgetter(1), rev

# 1. property
numberOfContributors = 0
for node in bc:
    if "Contributor" in node[0]:
        numberOfContributors += 1

print('1. Die Anzahl an Edges liegt nicht weit über der Anzahl an Contri
print('# of edges: {}'.format(nx.number_of_edges(g)))
print('# of contributors: {}'.format(numberOfContributors))

# 2. property

numberOfContributorsWithZeroBetweenness = 0
for node in bc:
    if "Contributor" in node[0] and node[1] == 0:
        numberOfContributorsWithZeroBetweenness += 1

print()
print('2. Die Betweenness-Centrality der meisten Benutzer ist 0.')
print('# of contributors with 0 bewteenness-centrality: {} (of {})'.form

print()
print('3. Die Betweenness-Centrality der restlichen Benutzer geht gegen
print(' -> ergibt sich aus 2.')
```

```
1. Die Anzahl an Edges liegt nicht weit über der Anzahl an Contributor
s.
# of edges: 988
# of contributors: 971

2. Die Betweenness-Centrality der meisten Benutzer ist 0.
# of contributors with 0 bewteenness-centrality: 971 (of 971)

3. Die Betweenness-Centrality der restlichen Benutzer geht gegen 0.
 -> ergibt sich aus 2.
```

Findings

Die Resultate bestätigt unsere These, zumindest im Fall von Google. Insbesondere in der Visualisierung des Graphen in Gephi lassen sich die Repository-Inseln gut erkennen, und es gibt nur einzelne Benutzer, welche an mehreren Projekten gleichzeitig arbeiten. Für diese Visualisierung wurde der Layout-Algorithmus *ForceAtlas 2* verwendet.

In [13]:
```
from IPython.display import HTML
from IPython.core.display import display

display(HTML('<img src="gephi.png" width="600px" style="margin: auto" />
```

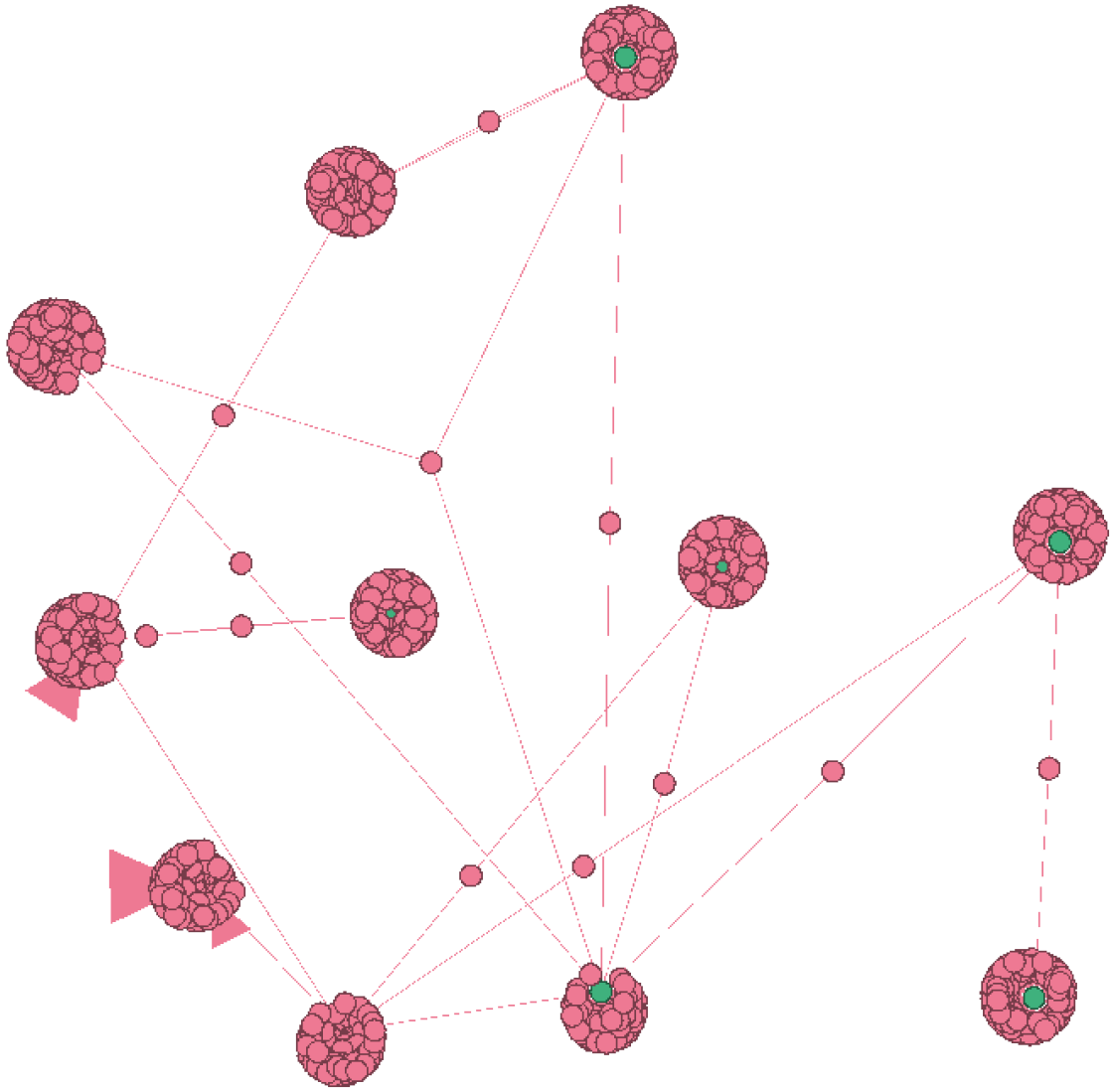

Abbildung 1 - Visualisierung in Gephi

Abbildungsverzeichnis

Literaturverzeichnis

Russell, Matthew A. / Mikhail Klassen (2019): Mining the Social Web: Data Mining Facebook, Twitter, LinkedIn, Instagram, GitHub, and More, 3. Aufl., Sebastopol, USA, California: O'Reilly Media.