Experiment #1:

Testing the approach of “Who Will be Interested in? A Contributor Recommendation Approach for Open Source Projects” (Xunhui Zhang et al., 2017

Rationale

This approach is very simple to understand and to implement. It works under three entities: developers, projects and terms. Developers and Projects are in GitHub, and Terms are strings associated to Developers and Projects. In our case, we replace Developers for Users and Projects for Questions, both in the context of the ROS Answers website. Terms are the tags associated to them.

Their approach is based on the idea that: “People who know each other are more likely to collaborate with each other”. I am not sure that this is the case of ROS Answers, but it is worth to test due to the small size of the community. Although simple, It also uses a Content-Based approach by using the tags of questions and users.

Adapting the approach, the formulas look like this:

- Activity(u,q) = {number of answers/comments made by u in q, + 1 if u is the asker of that question}
- $$U_q := \{\forall q \in Q, u \in U, \text{Activity}(u, q) > 0\}$$
- $$Q_a := \{\forall u \in U, q \in Q, \text{Activity}(u, q) > 0\}$$
- $$R_{uq}(u, q) = \frac{\text{Activity}(u, q)}{\sum_{i=1}^{U_q} \text{Activity}(U_q[i], q)}$$
- $$R_{uu}(u_a, u_b) = \frac{\sum_{q:\ \{Q_a \cap Q_b\}} R_{uq}(u_a, q) \cdot R_{uq}(u_b, q)}{\sqrt{\sum_{q:\ Q_a} R^2_{uq}(u_a, q) \cdot \sum_{q:\ Q_b} R^2_{uq}(u_b, q)}}$$
- $$\text{result}(u_a, q) = \sum_{U_q} R_{uq}(u, q) \cdot R_{uu}(u_a, u)$$

```
import sqlite3
import pandas as pd
from math import sqrt
from operator import itemgetter, attrgetter

conn = sqlite3.connect("v1.db")

## Activity: asking and commenting a question
query_activity_ans_comm = """ select active_users.q_id, active_users.u_id as u_id , count(*) as a
ctivity
                                from
                                (
                                    select rqa.ros_question_id as q_id, ra.author as u_id
                                    from ros_question_answer as rqa
                                    left join ros_answer as ra on rqa.ros_answer_id = ra.id
                                ) as active_users
                                group by q_id, u_id
"""
act_ans_comm = pd.read_sql_query(query_activity_ans_comm, conn)
```

```
## Activity: asking a question
query_activity_ask = """ select q_id, author as u_id, 1 as activity
                            from
                            (
                                select distinct ros_question_answer.ros_question_id as q_id
                                from ros_question_answer
                            ) as questions
                            left join ros_question as rq on questions.q_id = rq.id
"""
act_ask = pd.read_sql_query(query_activity_ask, conn)

conn.close()
```

```
act_ans_comm.head()
```

```
0       3
1      11
2     139
3       3
4    6791
Name: u_id, dtype: int64
```

```
act_ask.head()
```

	q_id	u_id	activity
0	9033	2	1
1	9036	2	1
2	9037	2	1
3	9038	2	1
4	9039	2	1

```
def activity_ans_comm(user,question):
    val = act_ans_comm[(act_ans_comm['u_id'] == user) & (act_ans_comm['q_id'] == question)]["acti
vity"]
    if val.empty:
        return 0
    return val.values[0]

# Tests
print("Tests")
print("-----")
print(activity_ans_comm(0,9045) == 0)
print(activity_ans_comm(3,9045) == 1)
print(activity_ans_comm(23668,9045) == 2)
```

```
def activity_ask(user,question):
    val = act_ask[(act_ask['u_id'] == user) & (act_ask['q_id'] == question)]["activity"]
    if val.empty:
        return 0
    return val.values[0]

# Tests
print("Tests")

print("-----")
print(activity_ask(7,9045) == 1 ) # True
print(activity_ask(3,9045) == 0 ) # True
print(activity_ask(23668,9045) == 0 ) # True
```

```
Tests
-----
True
True
True
```

```
# All users
def all_users():
    return pd.concat([act_ans_comm['u_id'],act_ask['u_id']]).drop_duplicates()

# Activity
def activity(user,question):
    return activity_ans_comm(user,question) + activity_ask(user,question)

# List of participants in a question - U_{q}
def participants_of_question(question):
    answerers = act_ans_comm[(act_ans_comm['q_id'] == question) & (act_ans_comm['activity'] > 0)]["u_id"]
    askers = act_ask[(act_ask['q_id'] == question) & (act_ask['activity'] > 0)]["u_id"]
    return pd.concat([answerers,askers]).drop_duplicates()

print("Asker ID = 7, Answers/Commenters = 3, 5184, 23668")
print(participants_of_question(9045))

# Relation between a user and a question
def r_uq(user,question):
    if activity(user,question) == 0:
        #print("activity zero: "+str(sum(map(lambda u : activity(u,question), participants_of_question(question)))))
        return 0
    return activity(user,question)/sum(map(lambda u : activity(u,question), participants_of_question(question)))

print("")
print("Asker: r_uq(7,9045)= "+str(r_uq(7,9045)))
print("Answerer : r_uq(3,9045)= "+str(r_uq(7,9045))+" (provides the accepted answer)")
print("Answerer : r_uq(5184,9045)= "+str(r_uq(5184,9045))+" (participated twice)")

# List of questions in which a user participates - Q_{u}
def questions_for_user(user):
    questions_answered = act_ans_comm[(act_ans_comm['u_id'] == user) & (act_ans_comm['activity'] > 0)]["u_id"]
    questions_asked = act_ask[(act_ask['u_id'] == user) & (act_ask['activity'] > 0)]["u_id"]
    return pd.concat([questions_answered,questions_asked]).drop_duplicates()
```

```python
# Relation between two users
def r_uu(user_a,user_b):
    questions_in_common = pd.Series(list(set(questions_for_user(user_a)) & set(questions_for_user(user_b))))
    a = sum(map(lambda q : r_uq(user_a,q)*r_uq(user_b,q), questions_in_common))
    b = sqrt(sum(map(lambda q : r_uq(user_a,q)**2, questions_for_user(user_a))) * sum(map(lambda q : r_uq(user_b,q)**2, questions_for_user(user_b))))

    if a == 0:
        # print("b: "+str(0))
        return 0
    return a/b

def result(user,question):
    return user,sum(map(lambda u : r_uq(u,question)*r_uu(user,u), participants_of_question(question)))

## TODO: Remove limit!!!!
def ranking_for_question(question):
    limit = 15 # let's work with 15 the top results only
    results = map(lambda u: result(u,question), all_users()[:300])
    return sorted(results, key=itemgetter(1), reverse=True)[:limit]

print("Ranking for q=9045")
print("------------------")
print(str(ranking_for_question(9045)))
for result in ranking_for_question(9045):
    print(str(result[0])+" - "+str(result[1]))
```

```
Asker ID = 7, Answers/Commenters = 3, 5184, 23668
25        3
26     5184
27    23668
10        7
Name: u_id, dtype: int64

Asker: r_uq(7,9045)= 0.16666666666666666
Answerer : r_uq(3,9045)= 0.16666666666666666 (provides the accepted answer)
Answerer : r_uq(5184,9045)= 0.3333333333333333 (participated twice)
Ranking for q=9045
------------------
[(3, 0.0), (11, 0.0), (139, 0.0), (6791, 0.0), (28, 0.0), (119, 0.0), (27, 0.0), (25, 0.0), (31, 0.0), (7, 0.0), (33207, 0.0), (44, 0.0), (51, 0.0), (437, 0.0), (9, 0.0)]
3 - 0.0
11 - 0.0
139 - 0.0
6791 - 0.0
28 - 0.0
119 - 0.0
27 - 0.0
25 - 0.0
31 - 0.0
7 - 0.0
33207 - 0.0
44 - 0.0
51 - 0.0
```

```
437 - 0.0
9 - 0.0
```