Introduction to Web Science

Assignment 9

Prof. Dr. Steffen Staab
staab@uni-koblenz.de

René Pickhardt
rpickhardt@uni-koblenz.de

Korok Sengupta
koroksengupta@uni-koblenz.de

Olga Zagovora
zagovora@uni-koblenz.de

Institute of Web Science and Technologies
Department of Computer Science
University of Koblenz-Landau

Submission until: January 18, 2016, 10:00 a.m.
Tutorial on: January 20, 2016, 12:00 p.m.

For all the assignment questions that require you to write scripts, make sure to **include the scripts in the answer sheet, along with a separate python file.** Where screen shots are required, please add them in the answers directly and not as separate files.

Team Name: QUEBEC

1. Daniel Kostic
2. Stefan Vujovic
3. Igor Fedotov

1 Generative models (abstract) (10 points)

In the lecture sessions you will learn about 6 potential parts you could find in research paper abstracts. Consider the following research paper abstract[1]

> Hit songs, books, and movies are many times more successful than average, suggesting that "the best" alternatives are qualitatively different from "the rest"; yet experts routinely fail to predict which products will succeed. We investigated this paradox experimentally, by creating an artificial "music market" in which 14,341 participants downloaded previously unknown songs either with or without knowledge of previous participants' choices. Increasing the strength of social influence increased both inequality and unpredictability of success. Success was also only partly determined by quality: The best songs rarely did poorly, and the worst rarely did well, but any other result was possible.

1. Name the 6 potential parts you could find in research paper abstracts.
2. Mark all parts you can find in the given abstract.

Answer:

1.1. Potential parts of research paper abstracts:

1. Background and Problem tackled with the research.
2. Used methodology.
3. One to three precise research question that are answered in the paper.
4. Unique solution or idea.
5. Demonstration of results.
6. Conclusion with a point of impact.

1.2.

1. "Hit songs, books, and movies are many times more successful than average, suggesting that "the best" alternatives are qualitatively different from "the rest"; yet experts routinely fail to predict which products will succeed." - **Background and Problem tackled with the research.**
2. "We investigated this paradox experimentally, by creating an artificial "music market" in which 14,341 participants downloaded previously unknown songs either with or without knowledge of previous participants' choices." - **Used methodology**
3. "Increasing the strength of social influence increased both inequality and unpredictability of success. Success was also only partly determined by quality: The best

[1] https://www.princeton.edu/~mjs3/salganik_dodds_watts06_full.pdf

songs rarely did poorly, and the worst rarely did well, but any other result was possible." - **Research results**

2 Meme spreading model (10 points)

We provide you with the following excerpt from the meme paper[2] which will be discussed at the lecture. This part of the paper contains and explanation of their basic model. Your task is to **list five model choices** that stay in conflict with reality and **discuss the conflict**.

> Our basic model assumes a frozen network of agents. An agent maintains a time-ordered list of posts, each about a specific meme. Multiple posts may be about the same meme. Users pay attention to these memes only. Asynchronously and with uniform probability, each agent can generate a post about a new meme or forward some of the posts from the list, transmitting the cor- responding memes to neighboring agents. Neighbors in turn pay attention to a newly received meme by placing it at the top of their lists. To account for the empirical observation that past behavior affects what memes the user will spread in the future, we include a memory mechanism that allows agents to develop endogenous interests and focus. Finally, we model limited attention by allowing posts to survive in an agent's list or memory only for a finite amount of time. When a post is forgotten, its associated meme become less represented. A meme is forgotten when the last post carrying that meme disappears from the user's list or memory. Note that list and memory work like first-in-first-out rather than priority queues, as proposed in models of bursty human activity. In the context of single-agent behavior, our memory mechanism is reminiscent of the classic Yule-Simon model.
>
> The retweet model we propose is illustrated in Fig. 5. Agents interact on a directed social network of friends/followers. Each user node is equipped with a screen where received memes are recorded, and a memory with records of posted memes. An edge from a friend to a follower indicates that the friend's memes can be read on the follower's screen (#x and #y in Fig. 5(a) appear on the screen in Fig. 5(b)). At each step, an agent is selected randomly to post memes to neighbors. The agent may post about a new meme with probability p_n (#z in Fig. 5(b)). The posted meme immediately appears at the top of the memory. Otherwise, the agent reads posts about existing memes from the screen. Each post may attract the user's attention with probability pr (the user pays attention to #x, #y in Fig. 5(c)). Then the agent either retweets the post (#x in Fig. 5(c)) with probability $1 - p_m$, or tweets about a meme chosen from memory (#v triggered by #y in Fig. 5(c)) with probability p_m. Any post in memory has equal opportunities to be selected, therefore memes that appear more frequently in memory are more likely to be propagated (the memory has two posts about #v in Fig. 5(d)). To model limited user attention, both screen and memory have a finite capacity, which is the time in which a post remains in an agent's screen or memory. For all agents, posts are removed

[2] http://www.nature.com/articles/srep00335

after one time unit, which simulates a unit of real time, corresponding to Nu steps where Nu is the number of agents. If people use the system once weekly on average, the time unit corresponds to a week.

Answer: Model choices that are in conflict with reality:

1. "An agent maintains a time-ordered list of posts, each about a specific meme." - **One post can contain multiple memes. Also, users see posts ordered by evaluated relevance to them, with the most relevant posts displayed first**
2. "Asynchronously and with uniform probability, each agent can generate a post about a new meme or forward some of the posts from the list" - **People make and share posts with memes in their free time, so long periods of inactivity could be expected (sleep, work), followed by periods in which many posts are created and shared (free time)**
3. "Users pay attention to these memes only." - **usually other content which is competing for the users attention is also on the screen (pictures, videos, advertisements)**
4. "we model limited attention by allowing posts to survive in an agent's list or memory only for a finite amount of time" - **people don't forget the funniest memes as quickly as the ones they found less fun and interesting**
5. "Our basic model assumes a frozen network of agents" - **this means that users neither get new followers, nor expand the list of users they follow. In the real world the network would change very often**

3 Graph and its properties (10 points)

Last week we provided you with a graph of out-links[3] of Simple English Wikipedia which should be reused this week.

Write a function that returns the diameter of the given directed network. The diameter of a graph is the longest shortest path in the graph.
Answer:

- Calculating the graph diameter can be accomplished by using different algorithms, some being:
  - Dijkstra's
  - Bellman-Ford
  - Floyd-Warshall
  - Breadth First Search

We have decided to implement Floyd-Warshall algorithm.

```
import pandas as pd
df2 = pd.read_csv('df2.csv', encoding='utf-8', index_col=0, header=0)
# CSV parsing
df2.out_links = df2.out_links.apply(lambda x: x.replace('[', ''))
df2.out_links = df2.out_links.apply(lambda x: x.replace(']', ''))
df2.out_links = df2.out_links.apply(lambda x: x.split(', '))

df2['link_num'] = df2.out_links.apply(lambda x: len(x))
df2 = df2.sort_values(by="link_num", ascending=False)
df2 = df2.drop_duplicates(subset = "name")
df2 = df2.set_index('name')
# df2 = df2.head(500)

diction = df2.to_dict()['out_links']


from time import time
def floydwarshall(graph):
    start = time()
    dist = {}
    pred = {}
    for u in graph.keys():
        dist[u] = {}
        pred[u] = {}
```

[3] http://141.26.208.82/store.zip

```
            for v in graph.keys():
                dist[u][v] = 1000
                pred[u][v] = -1
            dist[u][u] = 0
            for neighbor in graph[u]:
                dist[u][neighbor] = 1
                pred[u][neighbor] = u

    print(time() - start)

    diameter = 0

    for t in graph.keys():
        # given dist u to v, check if path u - t - v is shorter
        for u in graph.keys():
            for v in graph.keys():
                newdist = dist[u][t] + dist[t][v]
                if newdist < dist[u][v]:
                    dist[u][v] = newdist
                  # pred[u][v] = pred[t][v]
                    # route new path through t

    print(time() - start)

    return dist, pred


dist, pred = floydwarshall(diction)
# Ignore nodes that are not connected
only_d = [l for key in dist.keys() for l in dist[key].values() if l < 1000]
print(max(only_d))
```

Unfortunately, we have run into run time issues. When testing with only 1000 nodes the program finishes in 4-5 minutes.
The diameter we get is **8**. This makes sense as these long articles should be well connected. Running this algorithm for all the instances would take a long time, so we decided to use a library to speed things up. We made the graph using the Networkx python library.

```
import networkx as nx
G=nx.DiGraph(diction)
```

```
G.number_of_nodes()
```

```
34965
```

```
G.number_of_edges()
```

```
600288
```

A problem we faced was that the graph was not a strongly connected component, so we separated the largest SC component (24814 nodes.)

```
length = 0
subg = None
for subgraph in nx.strongly_connected_components(G):
    if len(subgraph) > length:
        subg = subgraph
        length = len(subgraph)
```

```
len(subg)
```

```
24814
```

```
df1 = df2.ix[list(subg)]['out_links']
```

```
df1_dict = df1.to_dict()
G=nx.DiGraph(df1_dict)
# print(nx.diameter(G))
```

```
G.number_of_nodes()
```

```
31980
```

```
len(df1)
```

```
24814
```

```
pG = G.subgraph(subg)
```

```
print(nx.diameter(pG))
```

```
41
```

The end result: **We got a diameter of 41.**

```
import pandas as pd
```

```
df2 = pd.read_csv('df2.csv', encoding='utf-8', index_col=0, header=0)
# CSV parsing
df2.out_links = df2.out_links.apply(lambda x: x.replace('[', ''))
df2.out_links = df2.out_links.apply(lambda x: x.replace(']', ''))
df2.out_links = df2.out_links.apply(lambda x: x.split(', '))

df2['link_num'] = df2.out_links.apply(lambda x: len(x))
df2 = df2.sort_values(by="link_num", ascending=False)
df2 = df2.drop_duplicates(subset = "name")
df2 = df2.set_index('name')
# df2 = df2.head(500)

diction = df2.to_dict()['out_links']


from time import time
def floydwarshall(graph):
    start = time()
    dist = {}
    pred = {}
    for u in graph.keys():
        dist[u] = {}
        pred[u] = {}
        for v in graph.keys():
            dist[u][v] = 1000
            pred[u][v] = -1
        dist[u][u] = 0
        for neighbor in graph[u]:
            dist[u][neighbor] = 1
            pred[u][neighbor] = u

    print(time() - start)

    diameter = 0

    for t in graph.keys():
        # given dist u to v, check if path u - t - v is shorter
        for u in graph.keys():
            for v in graph.keys():
                newdist = dist[u][t] + dist[t][v]
                if newdist < dist[u][v]:
                    dist[u][v] = newdist
                  # pred[u][v] = pred[t][v]
                    # route new path through t
```

```
    print(time() - start)

    return dist, pred


dist, pred = floydwarshall(diction)
# Ignore nodes that are not connected
only_d = [l for key in dist.keys() for l in dist[key].values() if l < 1000]
print(max(only_d))
```

3.1 Hints

1. You can first write a function that returns the shortest path between nodes and then find the diameter.
2. Do not forget to use proper data structures to avoid a memory shortage.

Important Notes

Submission

- Solutions have to be checked into the github repository. Use the directory name `groupname/assignment9/` in your group's repository.
- The name of the group and the names of all participating students must be listed on each submission.
- Solution format: all solutions as *one* PDF document. Programming code has to be submitted as Python code to the github repository. Upload *all* `.py` files of your program! Use `UTF-8` as the file encoding. *Other encodings will not be taken into account!*
- Check that your code compiles without errors.
- Make sure your code is formatted to be easy to read.
  - Make sure you code has consistent indentation.
  - Make sure you comment and document your code adequately in English.
  - Choose consistent and intuitive names for your identifiers.
- Do *not* use any accents, spaces or special characters in your filenames.

Acknowledgment

This latex template was created by Lukas Schmelzeisen for the tutorials of "Web Information Retrieval".

LaTeX

Currently the code can only be build using LuaLaTeX, so make sure you have that installed. If on Overleaf, there's an error, go to settings and change the LaTeXengine to `LuaLaTeX`.