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: papa
Team members:
Brigitte Aznar
Bonasmitha Behura
Ilia Tugushi

1 Generative models (abstract) (10 points)

In the lecture 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.

- State the background and problem
- Name the methodology you have used
- Formulate 1 to 3 precise research question
- Talk about your unique solution or idea
- Demonstrate the results
- Conclude with a point of impact

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

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.

1. frozen network of agents
2. limited attention
3. memory mechanism
4. retweet model
5. time model

1. The paper assumes a frozen network of agents, which means that contrary to reality individual's don't follow or unfollow other users. The network would stay unchanged.
2. It is true that people have limited attention, but it relies on that the memory of users have a limited capacity, which can be true, but as long as the meme keeps propagating, even if it out of a specific user's screen or memory chances are that it comes back by another user.
3. We do agree with the memory mechanism, except maybe that each user might have different memory period, it can't be uniform.
4. The process of retweeting gets propagated to every follower, and not just the neighbor.
5. The time model is representeb by the amount of time agents use the platform, but in real life depends on every user's character and not the frequency of use.

3 Graph and its properties (10 points)

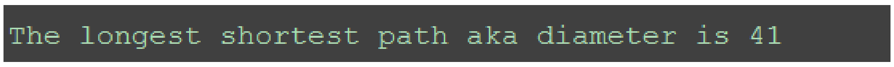
Last week we provided you with a graph of out-links[3] of Simple English Wikipedia which should be reused this week.

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.

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.

Answer:
The diameter is 41



```
The longest shortest path aka diameter is 41
```

Figure 1: Diameter of the given graph

```
import pandas as pd
import networkx as nx
#Functional way, but it takes ages stopped after 6h
def bfs(graph, start, end, visited = []):
    print("{} --- {}".format(start, end))
    # maintain a queue of paths
    queue = []
    if end in visited:
        visited = []
    # push the first path into the queue
    queue.append([start])
    while queue:
        # get the first path from the queue
        path = queue.pop(0)
        # get the last node from the path
        node = path[-1]
        # path found
        if node == end:
            return path
        # enumerate all adjacent nodes, construct a new path and push it into the
        for adjacent in graph.get(node, []):
            if adjacent not in visited:
                visited.append(adjacent)
```

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

```
                new_path = list(path)
                new_path.append(adjacent)
                queue.append(new_path)
    return new_path
def diameter(G, df):
    length = 0
    sub_graph = None
    for subgraph in nx.strongly_connected_components(G):
        if len(subgraph) > length:
            sub_graph = subgraph
            length = len(subgraph)

    graph2 = df.ix[list(sub_graph)]['out_links']
    graph2dict = graph2.to_dict()
    G = nx.DiGraph(graph2dict)
    pG = G.subgraph(sub_graph)
    return nx.diameter(pG)
def createDictionary(df):
    graph = {}
    names = []
    for item in df.itertuples():
        graph[item.name] = item.out_links
    return graph
def readStore():
    store = pd.HDFStore('store.h5')
    return store['df2']
def main():
    df = readStore()
    df = df.set_index("name")
    graph = df.to_dict()['out_links']
    G = nx.DiGraph(graph)
    longest_shortest = diameter(G, df)

    #for i in range(0, len(names)):
     #for j in range(0, len(names)):
        #shortest.append(bfs(graph, names[i], names[j]))

    #print(shortest)
    #longest_shortest = max(shortest)
    print("The longest shortest path aka diameter is {}".format(longest_shortest)
if __name__ == "__main__":
        main()
```

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`.