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: Tango

1. Mariya Chkalova
   mchkalova@uni-koblenz.de
2. Arsenii Smyrnov
   smyrnov@uni-koblenz.de
3. Simon Schauß
   sschauss@uni-koblenz.de
4. Lukas Härtel
   lukashaertel@uni-koblenz.de

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. 6 potential parts are:

1) research background and problem 2) description of the methodology that was applied in the article 3) research questions that are answered in the paper 4) unique solution or ideas 5) demonstration of the results 6) conclusion with the point of impact

2. In the given abstract we could find the following parts:

1) background 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.

2) description of the methodology that was applied in the article 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.

3) research questions that are answered in the paper "We investigated this paradox..."- that is above "predict which products will succeed"

5) demonstration of the results 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] 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, 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

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

post remains in an agent's screen or memory. For all agents, posts are removed 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:

- **frozen network of agents**: The conflict here lies in the assumption, that the number of users stays fixed. But in the real world users sign up and opt out of services like twitter.
- **each about a specific meme**: The agents only see and post memes, where in the real world a diverse set of contents is produced and consumed.
- **allowing posts to survive in an agent's list or memory only for a finite amount of time**: Memory may be recovered by certain trigger events and is not necessarily lost forever.
- **If people use the system once weekly on average, the time unit corresponds to a week.**: The time unit for each agent is fixed. But usage may fluctuate based on vacation etc, which may lead to the rise of memes spread in times of a high occupancy rate of the system.
- **Agents interact on a directed social network of friends/followers.** The Agents have no capability to follow or *unfollow* other agents based on their interest. So the model's edges are static and not in a constant flux like in the real world.

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.

Listing 1 Task 3

```
from tqdm import tqdm

CURSOR_UP_ONE = '\x1b[1A'
ERASE_LINE = '\x1b[2K'


def read_input():
    """
    Reads the input data and returns a matrix representing
    the graph, as well as node->index and index->node mapping.
    :return: Returns a matrix and two mappings
    """
    from pandas import read_hdf

    # Read the data set, remove some duplicate rows, and set index.
    dataset = read_hdf("store.h5", "df2")
    dataset.drop_duplicates("name", inplace=True)
    dataset.set_index("name", inplace=True, verify_integrity=True)

    # Make forward and backward node mapping dicts
    bwd = dict()
    fwd = dict()

    index = 0
    for n in dataset.index:
        bwd[index] = n
        fwd[n] = index
        index += 1

    # Create the matrix
```

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

```
31:     result = dict()
32:
33:     # Initialize the matrix
34:     for node, row in tqdm(dataset.iterrows(),
35:                           total=dataset.size,
36:                           desc="Preparing matrix",
37:                           unit="row"):
38:         # Map first node
39:         n1 = fwd.get(node)
40:         for edge in row.out_links:
41:             # Map second node
42:             n2 = fwd.get(edge)
43:             if n2 is not None:
44:
45:                 # Add a connection, create a new row if not present
46:                 row = result.get(n1)
47:                 if row is None:
48:                     row = {n2: 1}
49:                     result[n1] = row
50:                 else:
51:                     row[n2] = 1
52:
53:     # Return the matrix and the node mapping rules
54:     return result, fwd, bwd
55:
56:
57: def dijkstra(matrix, s):
58:     """
59:     Calculates Dijkstra and reports to a common progress bar.
60:     :param matrix: The matrix representation of the graph
61:     :param s: The node to start from
62:     :return: Returns the distance dictionary for node s
63:     """
64:     from heapq import heappush, heappop
65:
66:     # Value greater than everything else, not necessarily infinity
67:     inf = len(matrix) + 1
68:
69:     # Initialize result vector and queue
70:     res = {}
71:     queue = []
72:
73:     # Set initial distances
74:     for v in matrix:
75:         res[v] = inf
76:     res[s] = 0
77:
78:     heappush(queue, (res[s], s))
79:
```

```
80:     # Maximum units of work
81:     while queue:
82:         d, u = heappop(queue)
83:
84:         if d < res[u]:
85:             res[u] = d
86:
87:         for v in matrix[u]:
88:             x = res[u] + matrix[u][v]
89:             if res[v] > x:
90:                 res[v] = x
91:                 heappush(queue, (x, v))
92:
93:     return {k: v for k, v in res.items() if v != inf}
94:
95:
96: def uow(matrix, n):
97:     print(CURSOR_UP_ONE + ERASE_LINE + CURSOR_UP_ONE)
98:     print('progress: %.3f' % (n / len(matrix)))
99:     return max(dijkstra(matrix, n).values())
100:
101:
102: def diameter(matrix):
103:     """
104:     The diameter is the longest shortest path in the graph. Dijkstra returns a
105:     map from node to distance. The maximum value is the longest shortest path
106:     for one node. The maximum off this value over all nodes returns the diameter
107:     of the graph.
108:     :param matrix: The matrix representing the graph.
109:     :return: Returns the diameter as an integer.
110:     """
111:
112:     from concurrent.futures import ProcessPoolExecutor
113:     from functools import partial
114:
115:     # Execute on a process pool
116:     with ProcessPoolExecutor(max_workers=8) as executor:
117:         # Bind the unit of work: for a node, Dijkstra's algorithm is
118:         # executed to find all shortest paths; then, the maximum from the
119:         # values is taken. Pulling this into the unit of work allows to free
120:         # the memory of the distance dictionary. This should keep the memory
121:         # overhead low.
122:         return max(executor.map(partial(uow, matrix), matrix.keys()))
123:
124:
125: def task3():
126:     """
127:     Reads link data, calculates all pairs shortest path and selects the longest
128:     shortest path.
```

```
129:     :return: Returns None
130:     """
131:
132:     # Read the data and the mapping
133:     matrix, _, _ = read_input()
134:     print(diameter(matrix))
135:
136:
137: if __name__ == "__main__":
138:     task3()
```

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`.