Visualizing Longitudinal Tag Co-Occurrence on Stack Overflow Questions with Gephi

Alex Memory
memory@cs.umd.edu
University of Maryland
Information Visualization, Fall 2013
November 11, 2013

1 Introduction

Stack Overflow\(^1\) is an online programming Q&A web site started in 2008. Users of the web site can post questions about programming and have them answered by other users. As of 2013, the web site has millions of registered users and millions of questions. When a user posts a question, they may tag it with one or more topic categories. Since it is a web site about programming, common tags include programming languages, e.g., Java and .net.

The presence of two tags on the same questions generally has significance because both tags are relevant to the question, so it may be interesting to see which tags frequently co-occur on questions. And, since Stack Overflow continues to grow with new questions every day, it may also be interesting to see whether there are trends in co-occurrence over time.

2 Data

The data to do this are available because Stack Overflow makes its data available as a data dump, and Chris DuBois prepared summary data for nearly four months of 2009 from this dump.\(^2\) This summary includes records for 263,540 answers to 83,423 questions tagged with 14,248 tags and posted by 26,837 users. A few tags are used on many questions and there are many tags that are used on just a few questions, cf. Fig. 2.

Each record of the data identifies the question and the user that posted the question, the time of the question, the tags on the question, the views the question has received, identifiers for an answer and the answering user, and timestamps and scores for the question and answer (Stack Overflow users vote for the best questions and answers). The data on answers to questions isn’t needed for this analysis.

3 Observations

After extracting tag and tag pair counts for questions in the data it is possible to visualize tags as nodes in a graph and the co-occurrence of two tags as an edge between them. Using Gephi\(^3\) to visualize this graph, some interesting observations are possible.

---

\(^1\)www.stackoverflow.com
\(^2\)Chris DuBois’s version of the data: http://www.ics.uci.edu/ duboisc/stackoverflow/
\(^3\)Version 0.8.2. Gephi is available at http://gephi.org.
3.1 Diverse and strong co-occurrences among common tags

First, visualizing less than 100 of the most common tags over all months using the Fruchterman Reingold layout in Fig. 3.1, it is clear that the graph is densely connected among the most common tags. This suggests that questions posted to the web site often cross diverse programming topics instead of being strongly separated into disconnected categories, e.g., Windows programming versus web programming. Also, the weight of the edges is the number of co-occurrences. The co-occurrence between c# and .net is exceptionally strong, suggesting that the two tags are used together as a single tag by users.

3.2 A longitudinal view of tag co-occurrence

The remaining two observations use a longitudinal view in Gephi using dynamic attributes of the graph's nodes and edges. Each attribute value has an interval of time over which it holds and, using a playback feature of Gephi in combination with the Force Atlas layout\(^4\), the longitudinal history of tag co-occurrence within those intervals can be visualized. The view is an aggregate based on a sliding, week-long window that starts in February 2009 and ends in June 2009.

To avoid the dense connections among common tags, the view is based only on the top 2% of tags. With these tags as nodes, the view then limits edges by contributing to weights on edges only from days on which a the pair of tags has at least five co-occurrences. A final step to prepare the view is to remove any remaining nodes without neighbors during the entire period of time covered by the data. In addition, the labels on tags are sized according to the number of questions with that tag, and some tags that may have interest across many programming topics, e.g., career-development and best-practices, are highlighted in red.

The recording of the longitudinal view is available at the following URL. \(https://dl.dropboxusercontent.com/u/59003290/CMSC734/gephi_tag_co-occurence.mp4\)

3.2.1 Memory management concerns with iPhone programming

The second observation comes at time 0:55 in the longitudinal view, which corresponds to late March. A group of tags associated with iPhone programming is visible in the lower left side of the view, and at this point there is a new, strong co-occurrence between one of its nodes, iphone-sdk, and the memory-management tag, cf. Fig 3.2.1. This suggest that there may have been a surge in concern about memory management among iPhone developers during that period.

3.2.2 Programmers adopting Python

The final observation also comes at time 1:16 in the view. A group of tags near the center of the view having to do with Python development has a new co-occurrence with beginner, cf. Fig 3.2.2. The beginner tag is

\(^4\)Repulsion:8000, Autostabilize:2000, Gravity:500
Figure 2: The most commonly-used tags on Stack Overflow questions over all months. Tags are nodes and those that co-occur have edges between them. The graph is densely connected, suggesting that questions posted on the web site cross diverse topics. Also, the weight of the edges is the number of co-occurrences; the co-occurrence between c# and .net is exceptionally strong, suggesting that the two tags are used together as a single tag by users.
Figure 3: A new co-occurrence between iPhone development and memory management in late March 2009, suggesting that there may have been a surge in concern about this topic in that period.

Figure 4: A new co-occurrence suggesting that beginners are adopting the Python programming language.

4 Gephi

Gephi’s greatest positive aspect is that it is free, open source and portable to many platforms including Windows, Linux and Mac OS. Gephi also has a valuable interface for developing plug-ins, including new layout algorithms that were helpful in analyzing the tag co-occurrence data. Some features that were especially helpful in Gephi included the detailed controls on layout algorithms, the ability to interact with the graph while layout algorithms continue to run and flexible filtering controls. For static views of graphs, Gephi has an effective export feature for generating high quality images.

Unfortunately, some problems did arise during this analysis. The Gephi software proved to be prone to crashing, making it necessary to restart the software dozens of times during analysis. Also, dynamic filtering in combination with the use of the longitudinal view was not workable and could not be used.
There are ways that Gephi could be improved. There should be a details view showing the attributes of selected nodes and edges. Many controls are not fully documented in the interface, e.g., the MCL clustering algorithm. Also, it would be useful to have additional clustering algorithms, since the clustering algorithm that was available did not turn out to be useful.

5 Conclusion

Using Gephi to visualize co-occurrence of tags on Stack Exchange questions over time from February to June 2009, several interesting observations arise, suggesting trends involving specific programming problems and shifts in programming language adoption. These types of insights may be useful to organizations that track trends in programming, in topics covered by other Q&A web sites, and in other sources of tagged content.
Appendix: Data Preparation

November 11, 2013

Part I

Stack Overflow Tag Co-Occurrence Data


```python
import pandas as pd
import datetime
from xml import *
from xml.etree import *
from xml.dom import minidom
from xml.etree.ElementTree import Element, SubElement, Comment, tostring

pd.set_printoptions(max_rows=1000)
```

1 Basic cleanup

```python
sof1 = pd.read_csv('data/Stack_Overflow/DuBois/answers.csv', index_col=0)
sof1 = sof1.rename(columns={'qid':'q_id',
                             'i':'q_user_id',
                             'qs':'q_score',
                             'qt':'q_timestamp',
                             'qvc':'q_views',
                             'qac':'q_answers',
                             'aid':'a_id',
                             'j':'a_user_id',
                             'as':'a_score',
                             'at':'a_timestamp'})

sof1['q_id'] = sof1.q_id.apply(lambda x: "q%d"%x)
sof1['q_user_id'] = sof1.q_user_id.apply(lambda x: "u%0.f"%x)
sof1['a_id'] = sof1.a_id.apply(lambda x: "a%d"%x)
sof1['a_user_id'] = sof1.a_user_id.apply(lambda x: "u%0.f"%x)
sof1['q_timestamp'] = sof1.q_timestamp.apply(lambda x: datetime.datetime.fromtimestamp(x))
sof1['a_timestamp'] = sof1.a_timestamp.apply(lambda x: datetime.datetime.fromtimestamp(x))

sof1.head(2)
```

```plaintext
<table>
<thead>
<tr>
<th>q_id</th>
<th>q_user_id</th>
<th>q_score</th>
<th>q_timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>q563355</td>
<td>u62701</td>
<td>0</td>
<td>2009-02-18 18:34:41</td>
</tr>
<tr>
<td>q563355</td>
<td>u62701</td>
<td>0</td>
<td>2009-02-18 18:34:41</td>
</tr>
</tbody>
</table>
```

```python
tags q_views q_answers a_id
```
2 Descriptive info

```python
print 'users', len(unique(sof1.q_user_id))
print 'qs', len(unique(sof1.q_id))
print 'as', len(unique(sof1.a_id))
```

users 26837
qs 83423
as 263540

tags = pd.Series([tag for tag_list in sof1.tags.values for tag in tag_list.strip()].split())
tcounts = tags.value_counts()
print 'tags:', len(tcounts); print tcounts[:10]; print tcounts[-10:]
```
```
figsize(3,3)
plot(tcounts.values)
plt.show()
plot(np.log(tcounts.values))
plt.show()
```

tags: 14248

c# 38399
java 20003
.net 19509
c++ 17445
asp.net 14525
php 12910
javascript 12589
subjective 12416
python 10028
sql 9695
dtype: int64
```
```
jhat 1
googlemap 1
asnet-merge 1
dos2unix 1	ibco-gi 1
magazine 1
custom-method 1
subscriptions 1
point-of-service 1
stencil-buffer 1
dtype: int64
```
In [6]:
```
sof2 = sof1.dropna()
```

2.1 When do questions occur?

In [7]:
```
sof5 = sof2[['q_id', 'q_timestamp']].set_index('q_timestamp')
sof5.head(2)
```

Out [7]:
```
    q_timestamp
2009-02-18 18:34:41 q563355
2009-02-18 18:34:41 q563355
```

In [8]:
```
figsize(10,2)
sof5.q_id.resample('D',how='count').plot()
title('Counts of questions over time')
plt.show()
```
Weekends are clearly visible.

### 3 Extract tag pairs

Take the top 2% most frequently used tags.

```python
TOP_TAGS_PERC = 0.02
top_tags = list(tcounts[:int(len(tcounts)*TOP_TAGS_PERC)].index.values)
print 'top tags:', len(top_tags)
print top_tags[:4]
top tags: 284
['c#', 'java', '.net', 'c++']
```

```python
sof4 = sof2
sof5 = sof4[['q_id', 'q_user_id', 'q_score', 'q_timestamp', 'tags', 'q_views', 'q_answers']].drop_duplicates()
print len(sof4), len(sof5)
263540 83423
```

Out [91]:

```
q_id q_user_id q_score q_timestamp \
1 q563355 u62701 0 2009-02-18 18:34:41
3 q563356 u15842 10 2009-02-18 18:35:40
```

```python
print tags q_views q_answers
1 php,error,gd,image-processing 220 2
3 lisp,scheme,subjective,clojure 1047 16
```

Loop through the questions, generating a pair for each. The pairs are undirected, so canonicalize the pair by sorting them before adding them.

```python
for i, r in sof5.iterrows():
tag = r['tags'].split(',')
pairs_row = []
pairs_row += [(r['q_timestamp'], tuple(sort((tags[i], tags[j]))) for i in range(len(tags)) for j in range(i+1, len(tags))) if (tags[i] in top_tags) and (tags[j] in top_tags)]
tag_pairs += pairs_row
```

```python
print 'records:', len(sof5), 'pairs:', len(tag_pairs)
```
4 Extract tags question counts

We will use the number of questions for each tag each day in the visualization.

```python
In [93]:
    tp3 = tp2.groupby('pair')

4 Extract tags question counts

We will use the number of questions for each tag each day in the visualization.

```python
In [94]:
tag_counts = []
    for i,r in sof5.iterrows():
        tags = r['tags'].split(',')
        tags_row = []
        tags_row += [(r['q_timestamp'],tags[i]) for i in range(len(tags)) if tags[i] in top_tags]
        tag_counts += tags_row
    tag_counts = pd.DataFrame(tag_counts, columns=['timestamp','tag']).set_index('timestamp')
    tc2 = tc2.groupby('tag')

Out [94]:

```

```python
In [95]:
    tp4 = tp3.pair.resample(D,how='count')
    tp5 = tp4.reset_index() tp5 = tp5.rename(columns={0:'q_coocc','timestamp':'end'})
    tp5['tag1'] = tp5.pair.apply(lambda p: p[0])
    tp5['tag2'] = tp5.pair.apply(lambda p: p[1])
    tp5['start'] = tp5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 0, 0, 0))
    tp5['end'] = tp5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 23, 59, 59))
    tp5.head(4)
```

5 Resampling tag pairs to a day time period

```python
In [96]:
    print 'pair-days:',len(tp4); print
    tp5 = tp4.reset_index()
    tp5 = tp5.rename(columns={0:'q_coocc','timestamp':'end'})
    tp5['tag1'] = tp5.pair.apply(lambda p: p[0])
    tp5['tag2'] = tp5.pair.apply(lambda p: p[1])
    tp5['start'] = tp5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 0, 0, 0))
    tp5['end'] = tp5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 23, 59, 59))
    tp5.head(4)

Out [97]:

```

```python
In [98]:
    figsize(6,2)
    tp5.q_coocc.hist(bins=30)
    title('Frequencies of numbers of question for tag pairs')
    plt.show()
    tp5.q_coocc.value_counts()[:5]
```
6 Resampling tags to day time period

```python
In [99]:
tc4 = tc3.tag.resample('D', how='count')

In [101]:
print('tag-days:', len(tc4)); print
tc5 = tc4.reset_index()
tc5 = tc5.rename(columns={0: 'q_count', 'timestamp': 'end'})
tc5['start'] = tc5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 0, 0, 0))
tc5['end'] = tc5.end.apply(lambda end: datetime.datetime(end.year, end.month, end.day, 23, 59, 59))
tc5.head(4)
tag-days: 30647
```

```text
0  432412
1  59749
2  8528
3  2672
4  1234
dtype: int64
```

```python
Out [98]:
```

```text
0  432412
1  59749
2  8528
3  2672
4  1234
dtype: int64
```

```python
In [101]:
figsize(6,2)
tc5.q_count.hist(bins=30)
title('Frequencies of numbers of questions for tags')
plt.show()
tc5.q_count.value_counts()[:5]
```
7 Write to GEXF

Include only tag pairs on days where those pairs have at least five questions in common.

```python
MIN_COOCC = 5
tp6 = tp5[tp5.q_coocc >= MIN_COOCC].copy()
tp7 = tp6.groupby('pair')
tc7 = tc5.groupby('tag')

top = Element('gexf', attrib={'version': '1.2'})
gr = SubElement(top, 'graph', attrib={'defaultedgetype': 'undirected', 'timeformat': 'date', 'mode': 'dynamic'})
gr_atts_nodes = SubElement(gr, 'attributes', attrib={'class': 'node', 'mode': 'dynamic'})
SubElement(gr_atts_nodes, 'attribute', attrib={'id': 'questions', 'title': 'Questions', 'type': 'float'})
gr_atts_edges = SubElement(gr, 'attributes', attrib={'class': 'edge', 'mode': 'dynamic'})
SubElement(gr_atts_edges, 'attribute', attrib={'id': 'weight', 'title': 'Weight', 'type': 'float'})

# Nodes
nodes = SubElement(gr, 'nodes')
for tag, periods in tc7:
    node = SubElement(nodes, 'node', attrib={'id': tag, 'label': tag, 'start': str(periods[1].start.iloc[0].date()), 'end': str(periods[-1].end.iloc[0].date())})

attvals = SubElement(node, 'attvalues')
for i, per in periods.iterrows():
    attval = SubElement(attvals, 'attvalue', attrib={'for': 'questions', 'value': str(per['q_count']), 'start': str(per['start'].date()), 'endopen': str(per['end'].date())})
```

Out[102]:

<table>
<thead>
<tr>
<th></th>
<th>6428</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5376</td>
</tr>
<tr>
<td>2</td>
<td>5109</td>
</tr>
<tr>
<td>3</td>
<td>3441</td>
</tr>
<tr>
<td>4</td>
<td>2374</td>
</tr>
</tbody>
</table>

dtype: int64
# Edges

```python
eds = SubElement(gr, 'edges')

for pair, periods in tp7:
    ed = SubElement(eds, "edge", attrib={'
        source':pair[0],
        'target':pair[1],
        'start':str(periods[1].start.iloc[0].date()),
        'end':str(periods[-1].end.iloc[0].date()),
    })

    attvals = SubElement(ed, 'attvalues')
    for i, per in periods.iterrows():
        attval = SubElement(attvals, "attvalue", attrib={'
            for':weight,
            'value':str(per['q_coocc']),
            'start':str(per['start'].date()),
            'endopen':str(per['end'].date())
        })

pret = minidom.parseString(top).toprettyxml(indent="    ")
with open('tag_pairs_by-day.gexf', 'w') as f:
    f.write(pret)
```