Marco Bonzanini

Python, Data Science, Text Analytics

Mining Twitter Data with Python: Part 5 – Data Visualisation Basics

April 1, 2015June 16, 2015 · Marco

A picture is worth a thousand tweets: more often than not, designing a good visual representation (http://en.wikipedia.org/wiki/Data_visualization) of our data, can help us make sense of them and highlight interesting insights. After collecting and analysing Twitter data, the tutorial continues with some notions on data visualisation with Python.

Tutorial Table of Contents:

- Part 1: Collecting data (https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/)
- Part 2: Text Pre-processing (https://marcobonzanini.com/2015/03/09/mining-twitter-data-with-python-part-2/)
- Part 3: Term Frequencies (https://marcobonzanini.com/2015/03/17/mining-twitter-data-with-python-part-3-term-frequencies/)
- Part 4: Rugby and Term Co-Occurrences (https://marcobonzanini.com/2015/03/23/mining-twitter-data-with-python-part-4-rugby-and-term-co-occurrences/)
- Part 5: Data Visualisation Basics (this article)
- Part 6: Sentiment Analysis Basics (https://marcobonzanini.com/2015/05/17/mining-twitter-data-with-python-part-6-sentiment-analysis-basics/)
- Part 7: Geolocation and Interactive Maps (https://marcobonzanini.com/2015/06/16/mining-twitter-data-with-python-and-js-part-7-geolocation-and-interactive-maps/)

From Python to Javascript with Vincent

While there are some options to create plots in Python using libraries like matplotlib or ggplot, one of the coolest libraries for data visualisation is probably D3.js (http://d3js.org/) which is, as the name suggests, based on Javascript. D3 plays well with web standards like CSS and SVG, and allows to create some wonderful interactive visualisations.

Vincent (https://github.com/wrobstory/vincent) bridges the gap between a Python back-end and a front-end that supports D3.js visualisation, allowing us to benefit from both sides. The tagline of Vincent is in fact "The data capabilities of Python. The visualization capabilities of JavaScript". Vincent, a Python library, takes our data in Python format and translates them into Vega (https://github.com/trifacta/vega), a JSON-based visualisation grammar that will be used on top of D3. It sounds quite complicated, but it's fairly simple and *pythonic*. You don't have to write a line in Javascript/D3 if you don't want to.

Firstly, let's install Vincent:

```
sudo pip install vincent
```

Secondly, let's create our first plot. Using the list of most frequent terms (https://marcobonzanini.com/2015/03/17/mining-twitter-data-with-python-part-3-term-frequencies/) (without hashtags) from our rugby data set (https://marcobonzanini.com/2015/03/23/mining-twitter-data-with-python-part-4-rugby-and-term-co-occurrences/), we want to plot their frequencies:

```python
import vincent

word_freq = count_terms_only.most_common(20)
labels, freq = zip(*word_freq)
data = {'data': freq, 'x': labels}
bar = vincent.Bar(data, iter_idx='x')
bar.to_json('term_freq.json')
```

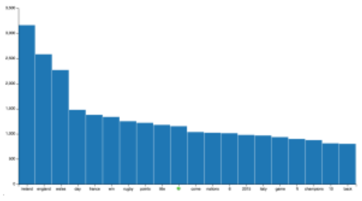
At this point, the file `term_freq.json` will contain a description of the plot that can be handed over to D3.js and Vega. A simple template (taken from Vincent resources) to visualise the plot:

```html
<html>
<head>
    <title>Vega Scaffold</title>
    <script src="http://d3js.org/d3.v3.min.js (http://d3js.org/d3.v3.min.js)" charset="utf-8"></script>
    <script src="http://d3js.org/topojson.v1.min.js (http://d3js.org/topojson.v1.min.js)"></script>
    <script src="http://d3js.org/d3.geo.projection.v0.min.js (http://d3js.org/d3.geo.projection.v0.min.js)" charset="utf-8"></s
    <script src="http://trifacta.github.com/vega/vega.js (http://trifacta.github.com/vega/vega.js)"></script>
</head>
<body>
    <div id="vis"></div>
</body>
<script type="text/javascript">
// parse a spec and create a visualization view
function parse(spec) {
  vg.parse.spec(spec, function(chart) { chart({el:"#vis"}).update(); });
}
parse("term_freq.json");
</script>
</html>
```

Save the above HTML page as `chart.html` and run the simple Python web server:

```
python -m http.server 8888 # Python 3
python -m SimpleHTTPServer 8888 # Python 2
```

Now you can open your browser at http://localhost:8888/chart.html (http://localhost:8888/chart.html) and observe the result:

 (https://marcobonzanini.files.wordpress.com/2015/04/term_freq.png)

Notice: you could save the HTML template directly from Python with:

```
bar.to_json('term_freq.json', html_out=True, html_path='chart.html')
```

but, at least in Python 3, the output is not a well formed HTML and you'd need to manually strip some characters.

With this procedure, we can plot many different types of charts with Vincent. Let's take a moment to browse the docs (http://vincent.readthedocs.org/en/latest/) and see its capabilities.

Time Series Visualisation

Another interesting aspect of analysing data from Twitter is the possibility to observe the distribution of tweets over time. In other words, if we organise the frequencies into temporal buckets, we could observe how Twitter users react to real-time events.

One of my favourite tools for data analysis with Python is Pandas (http://pandas.pydata.org/), which also has a fairly decent support for time series. As an example, let's track the hashtag `#ITAvWAL` to observe what happened during the first match.

Firstly, if we haven't done it yet, we need to install Pandas:

```
sudo pip install pandas
```

In the main loop which reads all the tweets, we simply track the occurrences of the hashtag, i.e. we can refactor the code from the previous episodes (https://marcobonzanini.com/2015/03/23/mining-twitter-data-with-python-part-4-rugby-and-term-co-occurrences/) into something similar to:

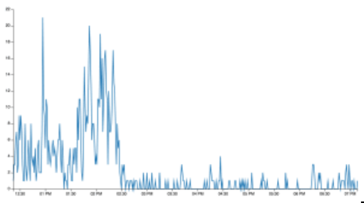
```
import pandas
import json

dates_ITAvWAL = []
# f is the file pointer to the JSON data set
for line in f:
    tweet = json.loads(line)
    # let's focus on hashtags only at the moment
    terms_hash = [term for term in preprocess(tweet['text']) if term.startswith('#')]
    # track when the hashtag is mentioned
    if '#itavwal' in terms_hash:
        dates_ITAvWAL.append(tweet['created_at'])

# a list of "1" to count the hashtags
ones = [1]*len(dates_ITAvWAL)
# the index of the series
idx = pandas.DatetimeIndex(dates_ITAvWAL)
# the actual series (at series of 1s for the moment)
ITAvWAL = pandas.Series(ones, index=idx)

# Resampling / bucketing
per_minute = ITAvWAL.resample('1Min', how='sum').fillna(0)
```

The last line is what allows us to track the frequencies over time. The series is re-sampled with intervals of 1 minute. This means all the tweets falling within a particular minute will be aggregated, more precisely they will be summed up, given `how='sum'`. The time index will not keep track of the seconds anymore. If there is no tweet in a particular minute, the `fillna()` function will fill the blanks with zeros.

To put the time series in a plot with Vincent:

```
time_chart = vincent.Line(ITAvWAL)
time_chart.axis_titles(x='Time', y='Freq')
time_chart.to_json('time_chart.json')
```

Once you embed the `time_chart.json` file into the HTML template discussed above, you'll see this output:

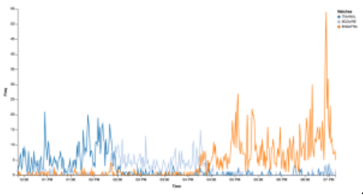
 (https://marcobonzanini.files.wordpress.com/2015/04/time1.png)

The interesting moments of the match are observable from the spikes in the series. The first spike just before 1pm corresponds to the first Italian try. All the other spikes between 1:30 and 2:30pm correspond to Welsh tries and show the Welsh dominance during the second half. The match was over by 2:30, so after that Twitter went quiet.

Rather than just observing one sequence at a time, we could compare different series to observe how the matches has evolved. So let's refactor the code for the time series, keeping track of the three different hashtags `#ITAvWAL`, `#SCOvIRE` and `#ENGvFRA` into the corresponding `pandas.Series`.

```python
# all the data together
match_data = dict(ITAvWAL=per_minute_i, SCOvIRE=per_minute_s, ENGvFRA=per_minute_e)
# we need a DataFrame, to accommodate multiple series
all_matches = pandas.DataFrame(data=match_data,
                               index=per_minute_i.index)
# Resampling as above
all_matches = all_matches.resample('1Min', how='sum').fillna(0)

# and now the plotting
time_chart = vincent.Line(all_matches[['ITAvWAL', 'SCOvIRE', 'ENGvFRA']])
time_chart.axis_titles(x='Time', y='Freq')
time_chart.legend(title='Matches')
time_chart.to_json('time_chart.json')
```

And the output:


(https://marcobonzanini.files.wordpress.com/2015/04/time2.png)

We can immediately observe when the different matches took place (approx 12:30-2:30, 2:30-4:30 and 5-7) and we can see how the last match had the all the attentions, especially in the end when the winner was revealed.

Summary

Data visualisation is an important discipline in the bigger context of data analysis. By supporting visual representations of our data, we can provide interesting insights. We have discussed a relatively simple option to support data visualisation with Python using Vincent. In particular, we have seen how we can easily bridge the gap between Python and a language like Javascript that offers a great tool like D3.js, one of the most important libraries for interactive visualisation. Overall, we have just scratched the surface of data visualisation, but as a starting point this should be enough to get some nice ideas going. The nature of Twitter as a medium has also encouraged a quick look into the topic of time series analysis, allowing us to mention `pandas` as a great Python tool.

If this article has given you some ideas for data visualisation, please leave a comment below or get in touch.

@MarcoBonzanini (http://www.twitter.com/marcobonzanini)

Tutorial Table of Contents:

- Part 1: Collecting data (https://marcobonzanini.com/2015/03/02/mining-twitter-data-with-python-part-1/)
- Part 2: Text Pre-processing (https://marcobonzanini.com/2015/03/09/mining-twitter-data-with-python-part-2/)
- Part 3: Term Frequencies (https://marcobonzanini.com/2015/03/17/mining-twitter-data-with-python-part-3-term-frequencies/)
- Part 4: Rugby and Term Co-Occurrences (https://marcobonzanini.com/2015/03/23/mining-twitter-data-with-python-part-4-rugby-and-term-co-occurrences/)
- Part 5: Data Visualisation Basics (this article)
- Part 6: Sentiment Analysis Basics (https://marcobonzanini.com/2015/05/17/mining-twitter-data-with-python-part-6-sentiment-analysis-basics/)
- Part 7: Geolocation and Interactive Maps (https://marcobonzanini.com/2015/06/16/mining-twitter-data-with-python-and-js-part-7-geolocation-and-interactive-maps/)

DATA MINING DATAVIZ NLP PYTHON TWITTER

18 thoughts on "Mining Twitter Data with Python: Part 5 – Data Visualisation Basics"

1. ***zuijiawoniu***
   says:
   APRIL 7, 2015 AT 6:10 AM
   thanks very much. it works.

   Reply
2. ***Pad***
   says:
   DECEMBER 22, 2015 AT 3:11 PM
   Hi Marco,
   I have a problem with timeseries,

   I just get one line with the value 1. looks like the resample doesnt work..

   json file looks like this:

"values": [
{
"col": "data",
"idx": 1445146969000,
"val": 1
},
{
"col": "data",
"idx": 1445147104000,
"val": 1
},
{

“col”: “data”,
“idx”: 1445147434000,
“val”: 1
},

every time a new “idx” shouldn’t it combine them for every minute?

Reply

1. ***pad***
   says:
   DECEMBER 24, 2015 AT 9:26 AM
   ok got it now, this line was the problem:

   time_chart = vincent.Line(ITAvWAL)

   changed to:

   time_chart = vincent.Line(per_minute)

   :D

   Reply

   1. ***Matt***
      says:
      APRIL 5, 2016 AT 2:33 PM
      Thanks a lot!

2. ***Ivan***
   says:
   MAY 16, 2016 AT 9:01 PM
   hallo, why i just got nothing on the webpage, could you tell me wether there are some important steps that i missed, thanks a lot!

   Reply

3. ***t0in4 (@VictorisUltra)***
   says:
   JANUARY 2, 2016 AT 4:05 PM
   I do not know why, but first example with vincent show nothing

   Reply

4. ***Dongqi Guo***
   says:
   MARCH 28, 2016 AT 9:43 AM
   I think there must be some problems in the second visua except the “per_minute” one, I revised the code but got nothing on the webpage…

   Reply

5. ***Kel3vra***
   says:
   MAY 10, 2016 AT 1:32 PM
   ERROR 1st example cannot create the ‘term_freq.json’
   bar = vincent.Bar(data, iter_idx=’x’)
   AttributeError: ’NoneType’ object has no attribute ‘Series’
   ——

```
import sys
import json
from collections import Counter
import re
from nltk.corpus import stopwords
import string
import vincent

punctuation = list(string.punctuation)
stop = stopwords.words(‘english’) + punctuation + [‘rt’, ‘via’]

emoticons_str = r”””
(?:
[:=;] # Eyes
[oO\-]? # Nose (optional)
[D\)\]\(\]/\\OpP] # Mouth
)”””

regex_str = [
emoticons_str,
r’]+>’, # HTML tags
r'(?:@[\w_]+)’, # @-mentions
r”(?:\#+[\w_]+[\w\’_\-]*[\w_]+)”, # hash-tags
r’http[s]?://(?:[a-z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-f][0-9a-f]))+’, # URLs

r'(?:(?:\d+,?)+(?:\.?\d+)?)’, # numbers
r”(?:[a-z][a-z’\-_]+[a-z])”, # words with – and ‘
r'(?:[\w_]+)’, # other words
r'(?:\S)’ # anything else
]

tokens_re = re.compile(r'(‘+’|’.join(regex_str)+’)’, re.VERBOSE | re.IGNORECASE)
emoticon_re = re.compile(r’^’+emoticons_str+’$’, re.VERBOSE | re.IGNORECASE)

def tokenize(s):
return tokens_re.findall(s)
```

```
def preprocess(s, lowercase=False):
tokens = tokenize(s)
if lowercase:
tokens = [token if emoticon_re.search(token) else token.lower() for token in tokens]
return tokens

if __name__ == '__main__':
fname = 'python.json'

with open(fname, 'r') as f:
count_all = Counter()
for line in f:
tweet = json.loads(line)
tokens = [term for term in preprocess(tweet['text'])
if term not in stop and
not term.startswith(('#', '@'))]
count_all.update(tokens)
word_freq = count_all.most_common(20)
labels, freq = zip(*word_freq)
data = {'data': freq, 'x': labels}
bar = vincent.Bar(data, iter_idx='x')
bar.to_json('term_freq.json')
```

Reply

1. ***Kevin***
   says:
   JULY 13, 2016 AT 1:13 AM
   Kel3vra and anyone else having this issue – I had the same, running 2.7.11 on Windows bash of all things. Switched over to Python 3 and the issue was resolved.

   Reply

6. ***Mandla***
says:
SEPTEMBER 7, 2016 AT 9:01 AM
Hi Marco, this might be a stupid question… but in this line of code:

match_data = dict(ITAvWAL=per_minute_i, SCOvIRE=per_minute_s, ENGvFRA=per_minute_e)

What are the "per_minute_i/s/e" values? how do you derive them?

Reply

7. ***Mandla***
says:
SEPTEMBER 7, 2016 AT 2:09 PM
Never mind… figured it out… It was really a stupid question. lol!

Reply

8. ***pallavi***
says:
FEBRUARY 15, 2017 AT 2:53 AM
how to save the HTML template directly from Python with:
1.bar.to_json('term_freq.json', html_out=True, html_path='char

Reply

1. ***Marco***
   says:
   FEBRUARY 28, 2017 AT 4:42 PM
   Hi,
   the full line is:
   bar.to_json('term_freq.json', html_out=True, html_path='chart.html')

   Cheers
   Marco

   Reply

9. ***pallavimini@gmail.com***
says:
MARCH 4, 2017 AT 4:58 PM
my chart.html is not working
it worked earlier
but now i dont know what happened
somebody please help me

Reply

10. ***isabellaleehs***
says:
DECEMBER 19, 2017 AT 7:38 AM
Hi, how can I use chart.html to display two charts at once?

Reply

11. ***klaus***
says:
FEBRUARY 4, 2018 AT 10:16 AM
Why there is nothing showing on the html webpage? Can anyone help?

Reply

12. ***faqih sadid (@sydd7)***
says:
MAY 23, 2018 AT 3:12 AM
Why there is nothing showing on the html webpage? Can anyone help?

Reply

13. *Michael (@modimikel)*

says:

MAY 26, 2018 AT 12:24 PM

There is an issue on Windows with the to_json function when saving the charts.html, the newline characters from the template are somehow switched to raw \n character, making the HTML format invalid and then the web browser can't display it. I looked into it and it seems there is something wrong with the function resource_string inside pkg_resources module that vincent uses, I did not see how to fix it.

In any case, replacing all the r'\n' with '\r\n' works.

Reply