> Note 1: If charts below won't load on GitHub, either run jupyter notebook locally, or open the PDF or HTML file in directory.
>
> Note 2: 'download.py' has been limited to download only 50 files for demonstration purposes. Remove limit in the script if needed.

News Headline Analysis (using GDELT data and Parse trees)

Introduction

Why am I doing this project?

Inspired by this project (https://github.com/AYLIEN/headline_analysis/blob/master/main-chunks.ipynb), I will largely emulate the analysis that was done but on a different data set from a more unique and up-to-date source – **GDELT**, or more specifically, the **GDELT 2.0: Our Global World in Realtime** (https://blog.gdeltproject.org/gdelt-2-0-our-global-world-in-realtime/).

My main goal for this personal project is to learn, practice, and explore `Python` scripting as I'm career switching and coding is fairly new to me. Secondly, as an introduction via parse trees (see below), I aim to further explore the **Natural Language Toolkit** (http://www.nltk.org/) or **NLTK** and how it is used in sentiment analysis. Lastly, I'm familiarising and leveraging on the GDELT database because I think it will be a good source of data for any future projects.

Other 'side-quests' are to explore integration with `SQLite` databases, practising SQL,

What is this project about?

As per the project by AYLIEN referenced above, instead of authors, I'm looking into the similarities and differences between the ways headlines for news articles are written on two websites (among many in the GDELT data). Namely:

1. `website1` = chinadaily.com.cn; and
2. `website2` = bbc.co.uk

Where is the data from?

In this project, I am using a sample size of 1000 random headlines for each website. The headlines are HTML parsed from URL links in the "Master CSV Data File List – English" (http://data.gdeltproject.org/gdeltv2/masterfilelist.txt) from GDELT. Python and SQL were used to scrape and organise the data which is now Pickled and ready for analysis using python.

The pickled files are easily found in the GitHub directory of this project. One can also run the other python scripts listed in the table below to download the entire database and work from there (e.g. to analyse other websites), but I suggest running the scripts locally as it will take up a lot of space.

How to get started (on local PC)?

Essentially, just running this jupyter notebook would suffice for the analysis as the dataset needed has been filtered and saved in the same directory.

But initially, I wrote 4 separate python scripts that have individual functions for downloading, processing, and extracting the GDELT data. The bulk of my learning (and time) came from scripting these 4 files to automate the data retrieval and wrangling.

I suggest to only run on a local computer files 1, 2, and 4 as the database it will download may be huge. On the other hand, the jupyter notebook (file 5) can be safely run online as it is using a filtered dataset.

#	Filename	Language	Type	Description	Comments
1.	download.py	python	main	Downloads all GDELT's CSV files	Run locally! Downloaded files can reach > 40 GB
2.	process.py	python	main	Processes the CSVs into an SQL database	Run locally after 1. as database can be big ~ 10 GB!
3.	dbhelper.py	SQL, python	support	Helps build the SQLite Database used in 2.	No need to run
4.	extract.py	python	main	Extracts required a list of dictionary for analysis	Run locally after 2
5.	analysis.ipynb	python	notebook	Main analysis done in Jupyter for visualisation	.

For more details on each file's function as well as improvements ideas, please refer to its dedicated notebook that is coming soon. In there, I will share instructions on how to run them and what to expect for

Initial setup

Parse trees primer – a quick introduction

In linguistics, a parse tree is an ordered, rooted tree that represents the syntactic structure of a string (sentence) according to some context-free grammar. [wiki] (https://en.wikipedia.org/wiki/Parse_tree)

For a simple sentence like "The cat sat on the mat", a parse tree might look like this:

 Parse Tree Example

> See here (https://www.clips.uantwerpen.be/pages/MBSP-tags) for more on the part-of-speech tags.

In this project, we are using the Pattern library (https://www.clips.uantwerpen.be/pages/pattern-en#tree) for Python to parse the headlines and create parse trees for each of them.

However, in future projects, I aim to use and explore NLTK instead of Patterns library because (1) it is more up to date and has more extensive features, and (2) I ran into a couple of issues while using the Patterns library in Python3.

Below, we load the Patterns library and see it in action using a test sentence:

In [1]:

```
from pattern.en import parsetree

# Example...
s = parsetree('The cat sat on the mat.')
for sentence in s:
    for chunk in sentence.chunks:
        print (chunk.type, [(w.string, w.type) for w in chunk.words])
```

```
NP [('The', 'DT'), ('cat', 'NN')]
VP [('sat', 'VBD')]
PP [('on', 'IN')]
NP [('the', 'DT'), ('mat', 'NN')]
```

Load the data

OK! First, we load the headlines for our first website. To do this, we'll borrow a function I wrote in the 'process.py' script which will help us load the pickled (https://wiki.python.org/moin/UsingPickle) data file that contains all the headlines.

In [2]:

```
from process import pickle_it, unpickle_it                # To save/load objects
 (variables) for future use.

website1 = unpickle_it("Website1.pickle")
```

The data is stored as a list of dictionaries with "title" as the keys and the headlines as its values. As shown below:

In [3]:

```
website1[:3]         # Shows the first 3 elements in the list.
```

Out[3]:

```
[{'title': 'Building boom will help boost overall capacity'},
 {'title': 'White House encourages FBI to complete probe'},
 {'title': 'China and US would gain from ditching of zero-sum mental
ity'}]
```

Parse the data

Next, we create the parse trees for each headline to help us with the analysis. We store the information in the same object (the dictionary) along with other basic information such as its length and word counts.

In [4]:

```
for story in website1:
    story["title_length"] = len(story["title"])
    story["title_words"] = len(story["title"].split())
    story["title_chunks"] = [chunk.type for chunk in parsetree(story["title"])[0
].chunks]
    story["title_chunks_length"] = len(story["title_chunks"])
```

In [5]:

```
website1[15]        # Previews the result of parsing for the 1st element.
```

Out[5]:

```
{'title': "Xi's Swiss visit to focus on China's growth, global probl
ems",
 'title_length': 60,
 'title_words': 10,
 'title_chunks': ['NP', 'NP', 'VP', 'PP', 'NP', 'NP', 'NP'],
 'title_chunks_length': 7}
```

In our main analysis, for the main indicator of the overall structure of a headline, we will use the first level of the parse tree (i.e. the chunk type sequence) of each headline.

So using the above example, we will be using the following sequence of chunk type:

```
['NP', 'NP', 'VP', 'PP', 'NP', 'NP', 'NP']
```

Similarity metric

Now that we have prepared the data for our first website, we need a way to tell how different or similar are two headlines. (e.g. how different or similar is A, B, C to A, B, D)

To do that, we have to look at it from a structural perspective (hence using the chuck type sequence) and employ the help of a similarity metric.

We use the `SequenceMatcher` class of `difflib` for this, which produces a similarity score between 0 and 1 for any two sequences (Python lists), as per the example below:

In [6]:

```
import difflib

# Example...
print ("Similarity scores for...\n")
print ("Two identical sequences: ", difflib.SequenceMatcher(None,["A","B","C"],[
"A","B","C"]).ratio())
print ("Two similar sequences: ", difflib.SequenceMatcher(None,["A","B","C"],[
"A","B","D"]).ratio())
print ("Two completely different sequences: ", difflib.SequenceMatcher(None,["A"
,"B","C"],["X","Y","Z"]).ratio())
```

```
Similarity scores for...

Two identical sequences:  1.0
Two similar sequences:  0.6666666666666666
Two completely different sequences:  0.0
```

Let's test it on our headline's chunk type sequence for two random articles from the first website:

In [7]:

```
v1 = website1[33]["title_chunks"]
v2 = website1[333]["title_chunks"]

print (v1, v2, difflib.SequenceMatcher(None,v1,v2).ratio())
```

```
['NP', 'NP', 'NP', 'VP'] ['VP', 'NP', 'VP', 'ADJP', 'PP', 'NP'] 0.4
```

Pair-wise similarity matrix for the headlines

Here we're going to apply the same sequence similarity metric to all of our headlines, and create a 1000x1000 matrix of pairwise similarity scores between the headlines:

(as the matrix size is big, the code may take some time to run)

In [8]:

```
import numpy as np

chunks = [site["title_chunks"] for site in website1]
m = np.zeros((1000,1000))
for i, chunkx in enumerate(chunks):
    for j, chunky in enumerate(chunks):
        m[i][j] = difflib.SequenceMatcher(None,chunkx,chunky).ratio()
```

Visualisation using Bokeh

Visualising all the headlines on a 2D scatter plot will help us get a clearer and better understanding of the way the websites headlines are structured. We will see that the similarly structured headlines will be close to each other.

To do that we're going to first use **t-Distributed Stochastic Neighbor Embedding** or **tSNE** to reduce the dimensionality of our similarity matrix from 1000 down to 2:

In [9]:

```
from sklearn.manifold import TSNE
tsne_model = TSNE(n_components=2, verbose=1, random_state=0)
```

In [10]:

```
tsne = tsne_model.fit_transform(m)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 1000 samples in 0.050s...
[t-SNE] Computed neighbors for 1000 samples in 2.727s...
[t-SNE] Computed conditional probabilities for sample 1000 / 1000
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration:
54.966888
[t-SNE] Error after 1000 iterations: 0.428370
```

And to add some bit of color, we use K-Means to identify 5 clusters of similar headlines, which we will then use in our visualization:

In [11]:

```
from sklearn.cluster import MiniBatchKMeans

kmeans_model = MiniBatchKMeans(n_clusters=5, init='k-means++', n_init=1,
                         init_size=1000, batch_size=1000, verbose=False, max_ite
r=1000)
kmeans = kmeans_model.fit(m)
kmeans_clusters = kmeans.predict(m)
kmeans_distances = kmeans.transform(m)
```

Finally, we use Bokeh (http://bokeh.pydata.org/en/latest/) to plot the chart:

In [12]:

```
import bokeh.plotting as bp
from bokeh.models import HoverTool, BoxSelectTool, ColumnDataSource
from bokeh.plotting import figure, show, output_notebook

colormap = np.array(["#1f77b4", "#aec7e8", "#ff7f0e", "#ffbb78", "#2ca02c", \
                     "#98df8a", "#d62728", "#ff9896", "#9467bd", "#c5b0d5", \
                     "#8c564b", "#c49c94", "#e377c2", "#f7b6d2", "#7f7f7f", \
                     "#c7c7c7", "#bcbd22", "#dbdb8d", "#17becf", "#9edae5"])

# colormap = np.array(["Matisse", "Spindle", "Flamenco", "Macaroni and Cheese",
 "Forest Green", \
#                      "Feijoa", "Punch", "Mona Lisa", "Wisteria", "Lavender Gra
y", \
#                      "Spicy Mix", "Quicksand", "Orchid", "Chantilly", "Gray",
 \
#                      "Silver", "Key Lime Pie", "Deco", "Java", "Regent St Blu
e"])

output_notebook()

plot_website1 = bp.figure(plot_width=900, plot_height=700, title="website1", \
                          tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave"
, \
                          x_axis_type=None, y_axis_type=None, min_border=1)

source = ColumnDataSource(data=dict(x=list(tsne[:,0]), \
                                    y=list(tsne[:,1]), \
                                    chunks = [x["title_chunks"] for x in website
1], \
                                    title = [x["title"] for x in website1], \
                                    cluster = kmeans_clusters, \
                                    color = colormap[kmeans_clusters]))

plot_website1.scatter('x','y', color='color', source=source)

hover = plot_website1.select(dict(type=HoverTool))
hover.tooltips = {"chunks": "@chunks", "title": "@title", "cluster": "@cluster"}

show(plot_website1)
```

BokehJS 0.13.0 successfully loaded.
(https://bokeh.pydata.org)

website1

Above is an interactive chart that shows a few groupings of headlines as well as some sparse ones.

Some dense group of headlines that stands out are:

1. NP, PP, NP group on the top left has short and to the point updates like "Monopoly law under revision";
2. NP, VP, NP group on the far right has headlines that indicate an action about to occur, in the "China to..." format; and
3. NP, VP, NP, PP, NP group at the bottom has headlines that are similar to group 2 above but with more context/details, for example: "SF sues Trump over sanctuary city executive order" or "Xinjiang imports 21 prized horses from Kazakhstan".

One can find other interesting groups using the 'box zoom' feature, as well as note the similarities to other points in proximity.

Comparing the two websites

Load and prepare the second website

And now, we load the second website and see how they compare. Following similar steps to the above, we will then calculate a similarity matrix for both website's headlines and store it in a 2000x2000 matrix.

In [13]:

```
website2 = unpickle_it("Website2.pickle")

for story in website2:
    story["title_length"] = len(story["title"])
    story["title_words"] = len(story["title"].split())
    story["title_chunks"] = [chunk.type for chunk in parsetree(story["title"])[0
].chunks]
    story["title_chunks_length"] = len(story["title_chunks"])
```

(as one can imagine, the huge matrix means it requires some time to build it)

In [14]:

```
chunks_joint = [site["title_chunks"] for site in (website1+website2)]
m_joint = np.zeros((2000,2000))
for i, chunkx in enumerate(chunks_joint):
    for j, chunky in enumerate(chunks_joint):
        sm = difflib.SequenceMatcher(None,chunkx,chunky)
        m_joint[i][j] = sm.ratio()
```

Common headlines

Next, we see how many common chunk type patterns exist between the two websites.

In [15]:

```
set1 = [site["title_chunks"] for site in website1]
set2 = [site["title_chunks"] for site in website2]
list_new = [itm for itm in set1 if itm in set2]
len(list_new)
```

Out[15]:

```
777
```

We see that there are 777/1000 or 77.7% of headlines with similar structure!

Visualising the headlines of the two websites

Similar to the steps above, but instead of color coding the clusters, we colors to indicate the different websites (Orange for website1 and Green for website2).

In [16]:

```
tsne_joint = tsne_model.fit_transform(m_joint)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2000 samples in 0.185s...
[t-SNE] Computed neighbors for 2000 samples in 20.514s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2000
[t-SNE] Computed conditional probabilities for sample 2000 / 2000
[t-SNE] Mean sigma: 0.000000
[t-SNE] KL divergence after 250 iterations with early exaggeration:
55.283096
[t-SNE] Error after 1000 iterations: 0.440748
```

In [17]:

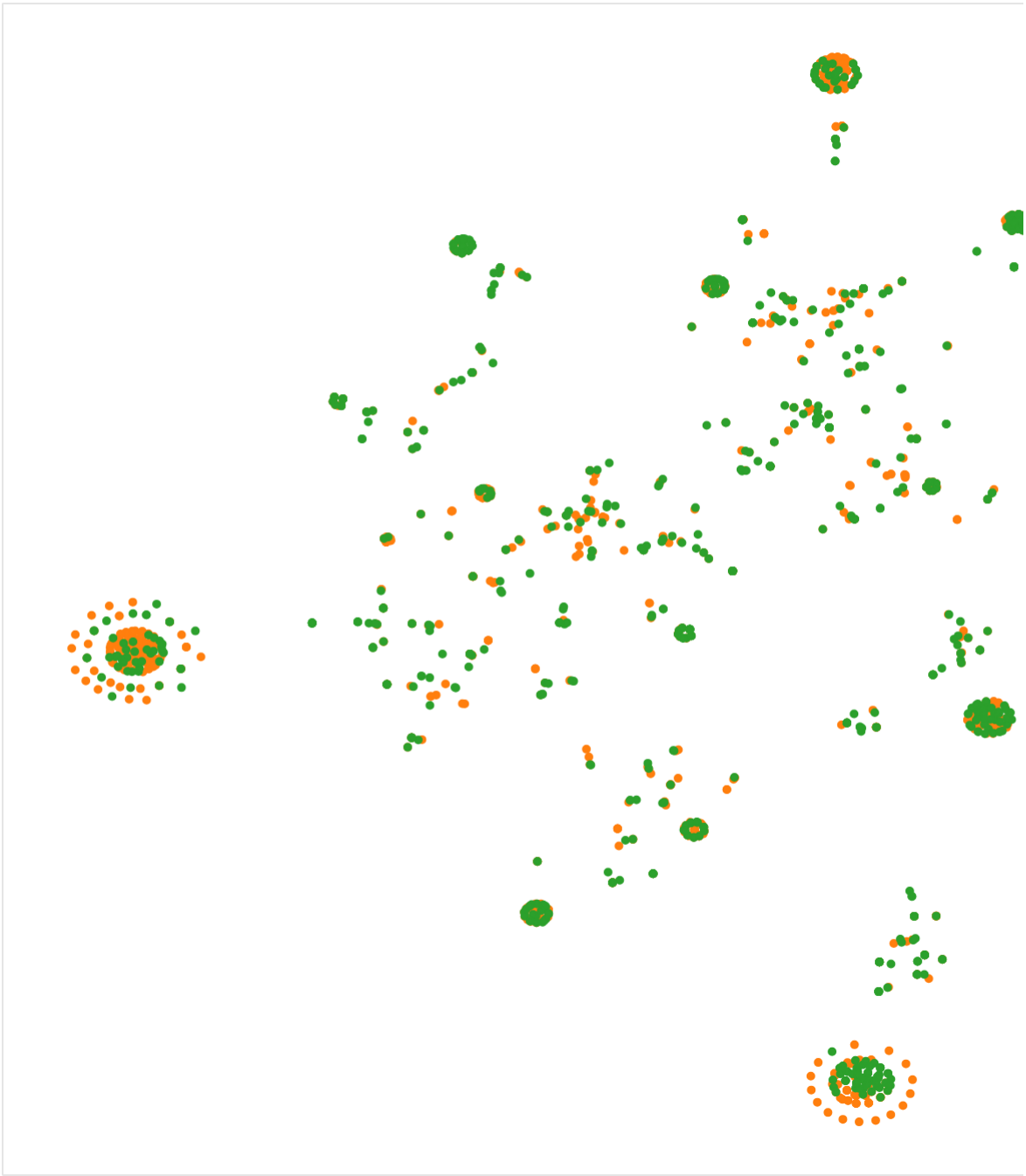
```
plot_joint = bp.figure(plot_width=900, plot_height=700, title="Website1 (Green)
 vs. Website2 (Orange)", \
                       tools="pan,wheel_zoom,box_zoom,reset,hover,previewsave",
 \
                       x_axis_type=None, y_axis_type=None, min_border=1)

source = ColumnDataSource(data=dict(x = tsne_joint[:,0], \
                                    y = tsne_joint[:,1], \
                                    chunks = [x["title_chunks"] for x in website
1] \
                                    + [x["title_chunks"] for x in website2], \
                                    title = [x["title"] for x in website1] + [x[
"title"] for x in website2], \
                                    color = colormap[([2] * 1000 + [4] * 1000
)]))

plot_joint.scatter(x='x', y='y', color='color', source=source)

hover = plot_joint.select(dict(type=HoverTool))
hover.tooltips={"chunks": "@chunks", "title": "@title"}
show(plot_joint)
```

Website1 (Green) vs. Website2 (Orange)



The chart above allows us to view, in the same way, the groups of headlines style which both websites shares or are unique to each.

Some observations:

- We can see the top portion of the chart has small clusters of Green points which suggest website2 (bbc.co.uk) has its unique but consistent style of headlines with forms like "Mexico prison riot: UN calls for investigation".
- But website1 (chinadaily.com.cn) has most of its Orange points in the major/bigger clusters in the top-, bottom-, left-, and bottom-right most portion of the chart.
- Visually the chart reaffirms the finding in "Common headlines" above, which showed that 77.7% of the headlines were similar!
- Unfortunately, as compared to the original AYLIEN project which analysis produced a chart that could visually distinguish the topic or themes of the authors' headlines in different clusters, my chart only separates them by the chunk patterns. For example, the leftmost cluster and the rightmost cluster may have different chunk patterns (longer vs. shorter), but both may be talking about the same topic such as the economy or Donald Trump.

Despite the drawback stated in the last point, a closer examination could yield new observations or could spark further points for investigation.

Conclusion and Future Work

From this project I have felt my python scripting has improved as I was able to automate retrieval of data (headlines) from multiple links, build an `SQLite` database to store and organise the large datasets, performed data wrangling and parsing using the `pandas` and `patterns` library, evaluate headline structures and similarities, and finally plotting the data points on a clear and interactive visual map using `Bokeh`.

For future projects, I may wish to:

1. Perform the analysis on other websites in the database (e.g. "indiatimes.com", "mainichi.jp", etc.)
2. Address the weakness of this approach as stated in the original project, i.e.:
   A. Using entire parse trees instead of just chunks;
   B. Using a tree or graph similarity metric instead of a sequence similarity one (ideally a linguistic-aware one too); and
   C. Better pre-processing to identify and normalize Named Entities, etc.
3. Employ the use of `NLTK` instead of `pattern` library as it has more features and could address the 2.C. issue above.