Media Coverage of the British General Election
Collecting and coding “big” media data

Dr. Travis G. Coan
t.coan@exeter.ac.uk

Dr. Iulia Cioroianu
i.cioroianu@exeter.ac.uk

University of Exeter
Exeter Q-Step Centre
Computer assisted approaches to collection and coding

Motivations. The impetus for this project was based on several observations:

1. Scholars of political communication and media studies are blessed (cursed?) with vast amounts of information and recent scholarship is beginning to make use of this information (Grimmer, 2015; Ansolabehere and Hersh, 2012).

2. Traditional content analytic methods do not scale.

Question: How can we use computational techniques to facilitate data collection, content coding, and analysis? When do computational methods work? When do they not?
Disclaimer: some programming required!
Learn Python!

Why Python?

1. Any language named after the Monty Python is worth giving a try!
2. It’s easy.
3. It’s fast enough (and pretty easy to speed up).
4. Large user community.
5. It is becoming industry standard for scientific computing.
Learn **Python!** (and/or learn **R**!)

There are many great resources to learn **Python** online:

1. Code Academy is a great place to start!  

2. Python for beginners:  
   [https://www.python.org/about/gettingstarted/](https://www.python.org/about/gettingstarted/)

3. Recommended books:  

Or if you don’t want to learn **Python**, then **R** is a must:

1. DataCamp is a great place to start: [https://www.datacamp.com/](https://www.datacamp.com/)


Social Media Data: Twitter
The use of Twitter in social science research is growing exponentially—Jungherr (2014) provides a comprehensive overview of 115 on the use of Twitter in research on elections alone.

Why is Twitter so popular? Some possibilities:

1. The Twitter APIs are open(ish) and easily accessible.
2. Twitter status update conventions—hashtags, @messages, @mentions, retweets, follower and following relationships, etc.—make the platform ideal for examining the networked interactions of users.
3. Considerable user base.
Using Twitter data with caution

As with most research focusing on social media platforms, the main challenge is potential **selection bias** (add citations):

1. Research shows that Twitter users are predominately young individuals, living in urban areas (insert citations).

2. The Twitter API only provides users with a small sample (i.e., the “garden hose”) of the total overall tweets (i.e., the “fire hose”) and no one seems to know how this sample is selected.

Doing a study on Bieber fever? Use Twitter!
Social media: taming Twitter

How does one actually download Twitter data? Three easy steps:

1. Set up a Twitter account and register your application (see https://spring.io/guides/gs/register-twitter-app/)
2. Decide on a software platform and send calls to the Twitter REST (e.g., the Search API) or Streaming APIs (see https://dev.twitter.com/rest/public).
3. Download the data and analyse!

Simple, right? Not so much!
There is no shortage of options for interacting with the Twitter API. We like the following options:

1. **R**: The `twittR` package is a good place to start (see [http://davetang.org/muse/2013/04/06/using-the-r_twitter-package/](http://davetang.org/muse/2013/04/06/using-the-r_twitter-package/) for a great tutorial).

2. **Python**: There are a number of different modules that wrap the Twitter API. Our favourite at the moment is `tweepy` (see [http://www.tweepy.org/](http://www.tweepy.org/))

There are also a bunch of “no programming required options:”
[http://blogs.lse.ac.uk/impactofsocialsciences/2015/07/10/social-media-research-tools-overview/](http://blogs.lse.ac.uk/impactofsocialsciences/2015/07/10/social-media-research-tools-overview/)
Free access to Twitter data comes in two versions:

1. **REST APIs.** These APIs basically queries Twitter’s backend server. **Upside:** you can perform more complex queries and you can go back in time (only a week, but sometimes that is important). **Downside:** tweets not collected in real time and need to deal with duplicated tweets.

2. **Streaming APIs.** These APIs keep a connection open continuously and “listen” for specific keywords. **Upside:** you can track keywords in real time! **Downside:** search queries need to be simple and more work on the programmer’s side to keep functioning correctly.
# load the tweepy module
import tweepy

# Authenticate, generate, and return api object
auth = tweepy.OAuthHandler(consumer_token, consumer_secret)
auth.set_access_token(access_key, access_secret)
api = tweepy.API(auth)

# Send a call for 10 tweets including the term "election"
call_api = tweepy.Cursor(api.search, q="election", result_type="recent", include_entities=True).items(10)

# Extract the tweets from the call
tweets = [tweet for tweet in call_api]

# Print the text of the first tweet
print tweets[0].text
What Twitter data did we actually collect?

We collected the following data:

1. **Opinion leaders.** We collected all of the tweets for 35 key individuals throughout the campaign (e.g., from David Cameron to Russel Brand).

2. **Search terms.** We queried 70 different search terms over the course of the election (e.g., items “tuition fees” to “terrorism”).

3. **Search “election” by geography.** We searched the terms “election” in 14 major cities in the UK (e.g., from London to Glasgow).

We averaged about 100 calls to the Twitter API for each 15 minute interval over the sample period.
How can we use these data?

Using Twitter data. How have Twitter data traditionally been used in the literature?

1. Social network analysis.
2. Sentiment analysis.

Connecting social and traditional media. Twitter often serves as a socially networked platform to link individuals to “traditional” news sources.
Social Media Data: Blogs
Collection blog information (web scraping in 10 minutes!)
Giving selenium a go

# Import the splinter package (wrapper for selenium)
```python
from splinter import *
```

# Fire up the browser!
```python
uagent = ("Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:24.0" + 
" Gecko/20100101 Firefox/24.0")
```
```python
browser = Browser(user_agent=uagent)
```

# Visit a blog
```python
browser.visit('http://www.conservativehome.com/')
```

# Click on a link
```python
link = browser.find_link_by_partial_text('David Thomas: If we want')
link.click() # click on the link
```

# Grab relevant features
```python
para = browser.find_by_tag('p') # grab all the paragraphs
html = browser.html # save the entire html
```
Great, we have a bunch of HTML... now what?
Everybody standback. I know regular expressions.
What is a regular expression?

A **regular expression** is a special language for describing a search pattern in a string (see [http://www.regular-expressions.info](http://www.regular-expressions.info)).

In **Python**, this would look something like:

```python
# Import the re module
import re

# Draw a box around the relevant text and extract paragraphs
# using the html saved from conservativehome.com.
search_str = (r'<!-- end article header -->(.+?)<!---->' +
             r' end article section'
box = re.findall(search_str, html, re.S)[0]
paras = re.findall(r'<p .*?>(.+?)</p>', box, re.S)
print paras[0]
```
With great power, comes great responsibility.

– Stan Lee
Traditional Media Data
Newspaper data: Nexis

News

Search

Tips for using search connectors

Custom date

From mm/dd/yyyy To mm/dd/yyyy

All English Language News

More sources

Add Index Terms

- Group duplicates
- Exclude Newswires
- Exclude Non-business news (obituaries, sports, reports, etc.)
- Exclude Websites
- Exclude documents with fewer than 500 words
Television

We collected national and regional television programmes from the following sources:

1. **National news.** We recorded shows using the Box of Broadcasts (http://bobnational.net/) for the following programmes: BBC News at 10pm, ITV News 10pm, Channel 4 7pm, Channel 5 7pm, Sky 9pm, BBC2 Newsnight.

2. **Scottish and Welsh news.** We also recorded shows using the following shows using the Box of Broadcasts (http://bobnational.net/): BBC Wales Today, BBC Reporting Scotland

3. **Regional news.** We recorded the following programmes using the BBC iPlayer: BBC Midlands Today, BBC Spotlight, BBC Look North (Yorkshire). Note that we used BoB for BBC London.
BoB transcripts

Thank you BoB!

Channel 4 News, Wednesday 11 Feb 2015 at 19:00

Transcript

0:00:02 - 0:00:04
Has their face fallen on one side?

0:00:04 - 0:00:08
Can they smile?

0:00:08 - 0:00:12
Can they raise both arms and keep them there?

0:00:12 - 0:00:13
Is their speech slurred?

0:00:13 - 0:00:16
(Slurs)
When all is said and done, how much data are we left with? Here are some back-of-the-envelope calculations:

1. Tweets \(\approx 23\) million
2. Blog posts \(\approx 8\) thousand
3. Newspaper articles \(\approx 476\) thousand
4. TV programmes \(\approx 1\) thousand

“Here’s a list of 100,000 warehouses full of data. I’d like you to condense them down to one meaningful warehouse.”
Content Analysis
Supervised classification: Traditional vs. computer assisted approaches

The “traditional” approach. The typical (inductive) quantitative content analysis looks something like this:

1. **Category definition.** Choose a set of theoretically meaningful categories based on one’s research question.

2. **Refine categories.** Randomly sample a small set of documents and apply categories based on criterion in (1). Revise as necessary (i.e. subsume or expand categories). REPEAT until happy with the categories.

3. **Check reliability.** Take a larger random sample of documents, code, and check reliability. Revise as necessary. REPEAT until you have hit your reliability targets.

4. **Code the texts.** Unleash your coders, collect the data, and bask in the glory!
Traditional vs. computer assisted approaches

The “computational” approach. The typical (supervised) learning approach looks something like this:

1. **Take a random sample for training and testing.** We start with a small(ish) random sample that will be used for model building and selection.

2. **Code the sample documents.** Use the “traditional approach” to content analysis for this small(ish) sample of documents.

3. **Train a classifier (or classifiers) using the coded data.** Split the data into training and testing sets and assess model performance based on accuracy, precision, and recall. REPEAT until you hit performance targets.

4. **Code the remainder of the texts.** Unleash your algorithm, collect the data, and bask in the glory!
Computational approaches to text analysis

Source: Grimmer and Stewart (2013), Table 1.
Human coding

After acquiring the documents, the first step for any “supervised” classification approach is to code a sample of data using traditional methods.

What categories did we code?

1. For all news stories, we code:
   - Date
   - Length of story (words or length of time)
   - Type of story (for newspaper articles)
   - Primary, secondary, and tertiary topic or theme

2. For election news stories, we add the following:
   - Election-specific themes
   - Primary, secondary, and tertiary actors
   - Tone
   - Disposition of actors (e.g., defending, attacking, etc.)
Human coding

**Procedure for training the human coding.**

1. Nine coders (undergraduate and graduate students)
2. All given a two hour training session where the classification criterion was outlined.
3. 1st training set: All coders were given the same sample of outlets to code over 10 days. Difficulties were discussed and the classification criterion was adjusted accordingly.
4. Additional training sets: REPEATED process in (3) until reliability was sufficiently high for independent coding.
A simple example of supervised classification

**Step 1: Research objective.** How much of the overall “news hole” was devoted to the election? To answer this question, we need to classify stories into election vs. non-election news.

**Step 2: Preprocessing.** Before applying computational methods, it is common to convert the text to lower-case, “stem” the text, and remove “stop words,” punctuation and numbers.

---

**An example of “preprocessing” text**

(untreated text) If this exit poll is right, Andrew, I will publicly eat my hat on your programme.

(processed text) if exit poll right andrew i publicly eat hat programme
Decide on an algorithm (or algorithms)

**Step 3: Choose an algorithm, fit model, and examine performance.** For instance, using scikit-learn and the trusty naive Bayes algorithm (via Python’s NLTK package):

```python
# Import NLTK
import nltk

# Train the classifier using "training" data
classifier = nltk.classify.NaiveBayesClassifier.train(train)

# Examine performance vs. a test set
classifier.classify_many(test)
nltk.classify.accuracy(classifier, test)
```
What’s next?

After successfully training a classifier and using it to code your corpus, what comes next? In short, the analysis! Yet several questions centring on measurement error need further elaboration in the literature:

1. How should one represent measurement error in coded documents? We know that there is error—should we ignore it?

2. How should this error be incorporated in statistical analyses?
Example Analyses
Content analysis

1. Simple statistics, word counts, frequencies
2. Text similarity measures
3. Unsupervised learning (clustering)
4. Supervised classification (already covered)
Text statistics

Are words being used differently by the Guardian and Telegraph?

- Count words

\[
\text{relative\_word\_mentioning} = \frac{\text{guardian.count(word)/len(guardian))}}{\text{telegraph.count(word)/len(telegraph))}}
\]

Guardian-Telegraph relative "majority" mentioning: 0.996722011669
Guardian-Telegraph relative "coalition" mentioning: 1.42277865316

- Analyze context; find other words that are commonly used in the same context.

```
guardian_text.similar("majority")
government labour would many minority party said city confidence conservative david deal focus says seats support supporters victory
telegraph_text.similar("majority")
party government confidence labour leader many minority money queen said tories vote allegations anonymity appeal apply back battle cabinet
```
Text similarity measures

How do we measure article similarity?

1. Use a Tf-Idf transformation to represent each article as a vector of weighted word scores.
   - Terms which occur a small number of times in a small number of documents receive a higher score.
   - Terms which occur in all documents receive a lower weighted score.

2. Compute cosine similarity
   - Calculate the cosine of the angle between the vector representations of documents
   - This provides a measure of document similarity on a normalized space.

Can use both NLTK and scikit-learns to measure text similarity.
from sklearn.feature_extraction.text import TfidfVectorizer
vect = TfidfVectorizer(min_df=1)
tfidf = vect.fit_transform(["As the City of London celebrates a Conservative victory, it must also face a referendum on the UK’s membership of the European Union... ",
"Relief. That was the reaction in the City as the election results came in. This was not the tightest national poll in decades..",
"Mumbai Indians claimed a vital Indian Premier League win at bottom side Kings XI Punjab, as they won by 23 runs in Mohali..",
"Kings XI Punjab’s woes continued in the Indian Premier League as they lost by nine wickets to Delhi Daredevils."])
cosine=(tfidf * tfidf.T).A
print cosine
1.00  0.73  0.38  0.53
0.73  1.00  0.42  0.56
0.38  0.42  1.00  0.55
0.53  0.56  0.55  1.00
Unsupervised learning

- Useful in the absence of labeled data and human coding of document into classes.
- Most common method of unsupervised learning is clustering.
- Goal: create such that we maximize similarity within each cluster and minimize it between clusters.
- K-Means clustering: minimize average squared Euclidean distance of documents in each cluster from their mean, which is the center of the cluster.
- Equivalent to minimizing residual sum of squares of observations from cluster centroids.
- Algorithm: select initial cluster centres randomly. Move them around in space, searching for the positions that minimize residual sum of squares.
K-Means Clustering
K-Means Clustering Example

- Cluster the articles published in The Guardian and The Telegraph the day after the elections.
- 2196 articles, N=2 clusters. Use only cleaned body of articles, no other identifiers or information.
- Use the scikit-learns K-Means algorithm.
- What would you expect the top features to be?
K-Means Clustering Example

Top terms per cluster (ordered features):

Cluster 0: labour, party, miliband, cameron, vote, said, cent, leader, ukip, poll, would, seats, polling, election, exit, farage, david, voters, minister, tories

Cluster 1: would, first, last, said, also, could, people, years, like, time, world, league, year, three, england, back, game, club, london, second

How would you label the clusters?