Mining social media for citizen insight: Handover document

April 20, 2023

1 Background

Social media sites such as Twitter, Reddit, and Mumsnet can be an important source of information for health research. They provide an archive of the thoughts, feelings, and concerns of large parts of the population on a wide range of topics. This can be used to explore citizen sentiment towards a topic, track changes over time, and reveal new bodies of concern that traditional research methods may miss. We can scrape data from websites such as Mumsnet or use the Twitter Academic API to search for tweets relevant to our research question. Natural language processing methods such Latent Dirichlet Allocation can then be used to structure the collections of tweets or posts into topics. Qualitative methods can also be used to interpret the topics found, or independently applied to generate an understanding of small numbers of posts or tweets. Below are the steps we have previously used for this type of work.

Steps:

- Identify sources of interest, e.g. the forum or social media you wish to search.
- Formulate a search strategy: search terms and a method for applying them.
- Collate your posts or tweets.
- Clean and pre-process your posts or tweets.
- Fit an LDA topic model.
- Interpret the topics found.

2 Identifying sources and collating posts

We have used Mumsnet and Reddit for recent work, and also have access to the Twitter Academic API [1]. Mumsnet is a UK based parenting forum with no specific API in place for access. We have previously scraped posts from here using a custom Python script which calls the BeautifulSoup parser [2]. Within the Github repository, the file mumsnetGetData.py will scrape all posts on the site according to the parameters it is given. These parameters are set at the top of the file and should be modified to specify the url of the specific Mumsnet forum, and page ranges, that the user wishes to download, as well as a location to store the resulting posts. This parameter setting code is shown below, where we specify that we would like to scrape pages 1 to 788:

```
#%% Set up parameters for where and when to scrape
#specify the URL of the specific mumsnet forum to scrape
url_health = "https://www.mumsnet.com/talk/childrens_health"
#set the location to store the file containing the posts
save_location = "H:/redditProject/mumsnetparentinghealth.feather"
#specify pages to scrape from - if you want all pages look on the website for how many there are
start_page = 1
final_page = 788
```

The parsing of the Mumsnet webpages is sensitive to any changes that the web developers make to their site. If the script does not work this could be the cause and should be investigated. Reddit is accessed using the Pushshift API [3]. The pullfromReddit.py file provides a method for collating all posts from a specific subreddit using this API. Again we must set some parameters to specify which posts we want, we specify the url supplied to the API and can here change the subreddit. We also specify the time frame we wish to select from, and a save location.

```
# specify the URL to the api for the subreddit of interest - see pushshift for details
api_url = 'https://api.pushshift.io/reddit/search/submission/?subreddit=Parenting'
api_url = api_url + '&size=100'
# specify the save location for the posts
save_location = './reddit_posts.feather'
# specify date range
start_date = datetime.date(2010,1,1)
end_date = datetime.date(2022,2,1)
today = datetime.date.today()
day = start_date
```

The Twitter API requires a password or bearer token to access. The Innovation Observatory has a bearer token stored on the Soft Intelligence Group teams channel, accessed through your IO teams account. It is stored within the Twitter Academic Research folder. The following code uses the Twarc tool [4] to access the Twitter API and download a set of tweets using a search query. To formulate your query consult the Twitter API documentation [1].

```
#specify the location of the bearer token for the API and a location to save results to
token_location = './BearerToken.txt'
save_location = r'./spirometers.pkl'
with open(token_location, 'r') as file:
    token = file.read()
client = Twarc2(bearer_token=token)
# Specify the start time in UTC for the time period you want Tweets from
start_time = datetime.datetime(2010, 1, 1, 0, 0, 0, 0, datetime.timezone.utc)
# Specify the end time in UTC for the time period you want Tweets from
end_time = datetime.datetime(2021, 12, 30, 0, 0, 0, 0, datetime.timezone.utc)

# This is where we specify our query
query = " spirometer at home -is:nullcast lang:en -is:retweet"
# The search_all method call the full-archive search endpoint to get Tweets based on the query, start ar
search_results = client.search_all(query=query)
alltweets = []
for p in search_results:
    result = expansions.flatten(p)
    for tweet in result:
        alltweets.append(tweet)
with open(save_location, 'wb') as f:
    pickle.dump(alltweets, f)
```

3 Pre-processing your posts and fitting and LDA topic model

Latent Dirichlet Allocation (LDA) is one of a number of natural language processing methods that can generate a topic model from a corpus of texts [5]. The file twitterTopicModellingTutorial.py shows how to use the gensim library [6] to do basic pre-processing and apply an LDA topic model to a collection of tweets. This script loads the tweets (these can be generated using twarc and stored either as csv file or a pickled list), pre-processes them, generates a dictionary of terms used in them, and then fits the topic model itself, visualises the topic model, then calculates the sentiment of each post.

3.1 Pre-processing

Pre-processing improves the performance of LDA by removing syntactic words to leave a smaller number of semantic words and bigrams. Homogenisation of language is also important in this step, including differences in spelling or the use of forum specific slang. Gensim [6] provides a standard set of pre-processing methods that provide good performance on typical data sets. We apply these here as well as removing a custom set of stopwords (words that we wish to exclude, typically include syntactic words such as "and", "or", etc, but can also include domain specific words). We also detect common bigrams (words that occur together) and include these in the dictionary in their own right.

```
#%% Preprocessing the tweets
# This creates a list of words that you don't want to include
# stopwords.words('english') is a list of all commonly used words in English
# we can add to this any additional words, in this case http is excluded because it
# often appears in links.
stop_words = stopwords.words('english') + ['http']

# Use gensim built in preprocessing - does all the things preprocessing you would typically do in topic
posts_preprocessed = gensim.parsing.preprocessing.preprocess_documents(posts)

posts_processed = []
# remove the stopwords and any words with fewer than three characters
for post in posts_preprocessed:
    post = [p for p in post if p not in stop_words and len(p)>3]
    posts_processed.append(post)

# Find all bigrams - word pairs that occur together more than 20 times
# e.g. feel like, fall asleep ect. convert to single words feel_like
bigram = Phrases(posts_processed, min_count=20)

# for each post add bigrams as additional words
for idx in range(len(posts_processed)):
    for token in bigram[posts_processed[idx]]:
        if '_' in token:
            # Token is a bigram, add to post.
            posts_processed[idx].append(token)
```

3.2 Fitting a topic model

Having pre-processed the data and created the dictionary and corpus, we can pass these to the gensim LdaModel function, which will fit a topic model to them. This function takes a number of parameters, we keep the default parameters in most cases, but the optimum number of topics must be derived empirically. This can be achieved through a parameter scan across a range of numbers.

```
#%% Apply the topic model
# We can change the parameters here depending on the dataset.
# The most obvious is the num_topics, here we have six but if we suspect
# that our data naturally contains more we may wish to increase this
#lda_models = []
number_of_topics  = 8
lda_model = LdaModel(corpus=corpus,
                         id2word=dictionary,
                         random_state=100,
                         num_topics=number_of_topics,
                         passes=20,
                         chunksize=1000,
```

```
                    decay=0.5,
                    offset=64,
                    eval_every=10,
                    iterations=1000,
                    gamma_threshold=0.001,
                    per_word_topics=True)

top_topics = lda_model.top_topics(corpus)

# Average topic coherence is the sum of topic coherences of all topics, divided by the number of topics.
avg_topic_coherence = sum([t[1] for t in top_topics]) / 6
print('Average topic coherence: ' + str(round(avg_topic_coherence,4)))
```

In a large model, high performance computing will be required to run this. The script topicModellingParamScan.py shows how this can be done for a dataset taken from Mumsnet and Reddit. This was run on the Newcastle University rocket HPC [7] which uses the SLURM workload manager to manage jobs submitted to it. The file topicmodel.sbatch is an example script that can be used to run a the topicModellingParamScan.py file on Rocket. Coherence is one measure used to evaluate the goodness of fit of a topic model. Figure 1 shows the coherence of a model run using the above scripts, we can use this type of plot to choose an optimal number of topics. The library LDAvis [8] is another useful tool for evaluating an LDA model. It provides a

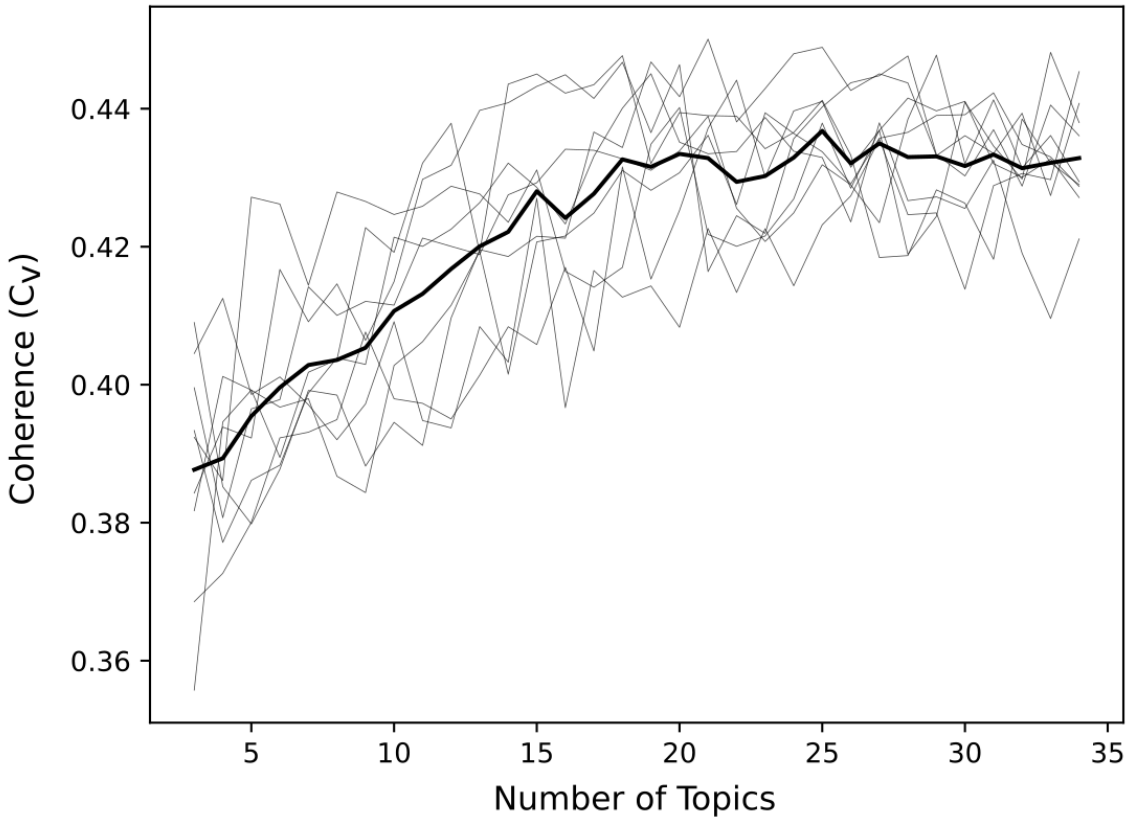

Figure 1: The coherence of an LDA topic model of posts on Mumsnet and Reddit.

visualisation of each topic, allowing the user to asses whether the topics are sufficiently distinct or whether

there are too few topics. Figure 2 shows an example of the output it produces. To use this library simply import the module and provide it with your fitted gensim LDA model, it generates an html file which can be opened in your browser. The following code illustrates this:

```
import pyLDAvis.gensim_models as gensimvis
import pyLDAvis
lda_viz = gensimvis.prepare(lda_model, corpus, dictionary)
pyLDAvis.save_html(lda_viz, 'twitterLDA15topics.html')
display_data = pyLDAvis.display(lda_viz)
```

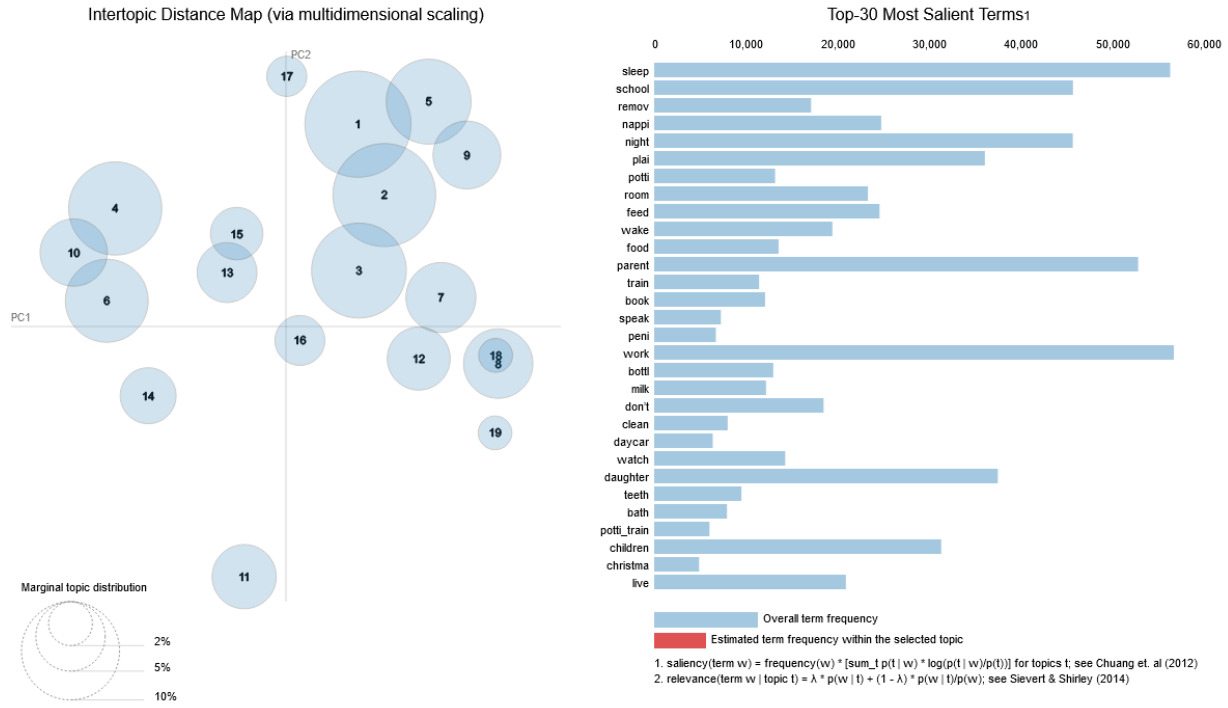

Figure 2: The output LDAvis on an example topic model of posts on Mumsnet and Reddit.

4 Interpreting a topic model

The output of the LDA process provides a list of most frequently found words within each topic. These can be found using LDAvis, or programmatically using the following code:

```
modelwords = lda_viz[1]
topicnamesstr = []
for t in range(1,number_of_topics+1):
    topic = 'Topic' + str(t)
    topicwords = modelwords[modelwords.Category==topic]
    order = list(np.argsort(topicwords.loglift + topicwords.logprob)[-4:])
    order.reverse()
    top5 = topicwords.iloc[order]
    topic_string = ''
    for tp in top5.Term:
```

```
        topic_string = topic_string + str(tp) + '+'
```

These words provide a good start in any interpretation of a topic, and LDAvis provides the best way to browse through them. If you are struggling to see any coherence in the words of your topics then it is likely that you need to improve your model fit, either by doing further pre-processing, using a different number of topics, or getting better source data. However, frequently used words cannot provide a complete picture of the content of a topic, and qualitative analysis is required to gain insight beyond word frequency. A sample of posts most closely associated with each topic can be extracted and a close reading of these using qualitative methods can provide a list of themes within each topic. We do this using the following code, which finds a list of posts that have a greater than 75% probability of being generated by a single topic and labels them with that topic:

```
postslikely_df = posts_df[posts_df.singleprobperdoc>=0.75]
individualtopics = pd.DataFrame()
for i in postslikely_df.singletopicperdoc.unique():
    p_A = postslikely_df[postslikely_df.singletopicperdoc==i]
    p_A['topicwords'] = topicnamesstr[i]
    individualtopics = individualtopics.append(p_A)
```

text	ngletopicperdo	singleprobperdoc	topicwords
@porkpiegate @JamesDamla @charlesr1971 @AngelaR…	5	0.964983	plai+week+hear+mile
@668Jayne @ThePoke Well some of us do need to l…	5	0.8249419	plai+week+hear+mile
Call the political correctness police @joelycet…	5	0.8749419	plai+week+hear+mile
@RossKroft96 Ignore them. If they are blocking …	5	0.8748972	plai+week+hear+mile
@SkySportsNews Gareth the 'yes man' gets reassu…	5	0.9027512	plai+week+hear+mile
Great to see @samanthallen and Sir James Mackey…	5	0.9326493	plai+week+hear+mile
@pequeno0229 @simpinato @Magic_Of_Peds https://…	5	0.85414124	plai+week+hear+mile

Figure 3: Shows a snippet of the table of posts and their associated topics and probabilities.

References

[1] *Twitter Academic Research API.* https://developer.twitter.com/en/products/twitter-api/academic-research

[2] *Beautiful Soup Python Library Documentation* https://beautiful-soup-4.readthedocs.io/en/latest/

[3] Baumgartner J, Zannettou S, Keegan B, Squire M, Blackburn J. The Pushshift Reddit Dataset. Proceedings of the International AAAI Conference on Web and Social Media. 2020. pp. 830–839. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/7347

[4] A command line tool (and Python library) for archiving Twitter JSON https://twarc-project.readthedocs.io/en/latest/

[5] Kherwa P, Bansal P, (2019). Topic Modeling: A Comprehensive Review. ¡i¿EAI Endorsed Transactions on Scalable Information Systems¡/i¿, 7(24), 1–16. https://doi.org/10.4108/EAI.13-7-2018.159623

[6] Rehurek R, Sojka P (2011). Gensim–python framework for vector space modelling. NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic, 3(2).

[7] Rocket HPC Service (high performance computing) https://services.ncl.ac.uk/itservice/research/hpc/

[8] Carson Sievert and Kenneth Shirley. 2014. LDAvis: A method for visualizing and interpreting topics. In Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces, pages 63–70, Baltimore, Maryland, USA. Association for Computational Linguistics