REPORTING ON TERRORISM
Case study of Document Clustering And Topical Modelling

1. Problem

The primary purpose of this case study is to find similarities between articles about terrorist attacks as reported by The Guardian. Using document clusterization and topical analysis, the goal is to identify common expressions and language used to report on the attacks and find out if there are differences in how different attacks are being covered. The initial assumptions is that it should be possible to distinguish meaningful groups of articles based on the place of an attack as mentioned in the Paris, Beirut, and the Language Used to Describe Terrorism article.

The secondary purpose of this case study is to create a reusable pipeline for document cluseristation, which could be reused to power bespoke data visualisations.

2. Approach

The task of clusterization and topical analysis can be split into 8 subtasks which will be discussed separately:

2.1 Project setup

2.1.1 Python environment

The majority of work on the project has been done on the Mid 2015 Macbook Pro running OS X El Capitan. This OS X comes with a default Python installation, version 2.7.0. However, for better future-proofing, this project has been developed using newer version of Python 3.

Instead of installing a new Python version globally, pyenv, Python version manager, has been used to install Python version 3.6.0. Pyenv allows to switch easily between Python versions on per-project basis.

Similarly, to avoid installing project dependencies (libraries etc.) globally, pipenv has been used to create a self-contained environment specific for this project. Dependencies installed with pipenv are automatically added to the Pipfile file. Unlike other Python dependency solutions (e.g. pip and its requirements.txt), the Pipfile is guaranteed to list all the necessary dependencies. Pipenv also creates a lock file which specifies precisely which versions of libraries to use. Setting up Python dependencies with pipenv should be more straightforward.

2.1.2 MongoDB

All the articles are stored as documents in the MongoDB database. MongoDB has been installed using homebrew. The connection to the local MongoDB instance is stored as a mongo connection string in the MONGODB_URI environment variable.

The individual steps necessary to set up the project are listed in the project readme file.

2.2 Retrieving and storing the data

The data used in this project, articles about terrorism published by The Guardian, are retrieved from via Guardian API called Open Platform.

Guardian allows to search through their collection of published articles with their search endpoint. This endpoint allows to retrieve articles based on keywords. In our case, the keyword used is ‘terrorist’.

Since the API response is paginated, only subsets of found articles is returned for each request, it was necessary to make repeated API request to collect all the relevant articles. In total, there were over 41,000 articles related to terrorism available, as of July 2017.

Before storing the retrieved articles in the MongoDB collection, we add a query used for the API call with the article document. Should we retrieve articles for different keywords, we needed to have a way of finding out from which query the given document comes from.

Finally, the retrieved documents are stored in the articles collection of the MongoDB database.
Since we’re interested in articles reporting on specific attacks, not general pieces about terrorism, additional filtering of the stored results was necessary. After the initial analysis, only articles from World and UK-News sections were used for the clustering and topic analysis.

The main script to fetch articles from the Guardian API is fetch_data.py which makes a heavy use of the fetchers module.

2.3 Processing data

We don’t modify the fetched data and store them as they are retrieved from the Guardian API. This allows us to be agnostic about how the data can be used. However, it also means we have to preprocess the data on-the-fly every time we run the clusterization algorithm.

2.3.1 Get article content

Each article is stored with a wide range metadata. For the clustering and topical analysis, though, the only relevant field is the actual content of the article which has to be retrieved from the ‘fields’ object.

2.3.2 Removing html

The articles contain links and, possibly, other HTML markup. The HTML tags within tags would hinder the tokenization process later on and they have to strip. BeautifulSoup, a popular library for web-scraping, is used to parse the article html content and retrieve only plain text.

2.3.3 Tokenizing and stemming

As a part of the vectorization process described in the next section, texts of articles are split into tokens. We need to carry on additional filtering to get rid of tokens which are irrelevant or of little importance to the clustering analysis.

Firstly, we use NLTK, a popular library for natural text processing, to split text into tokens with the word_tokenize method.

Secondly, we don't want to consider tokens which are just punctionions marks. We use a simple regular expression to filter those out.

Thirdly, we want to avoid clusterization based on name of places or people. We're interested in similarities of the language used and the clusterization algorithm would put too much emphasis on proper nouns. NLTK has the part of speech capabilities which would allow us to identify proper nouns. Unfortunately, this process is far too time-consuming and we have to use much more imprecise, but way more efficient, method of simple removing tokens starting with an uppercase letter.

Lastly, we want to consider only the root form of each token. The process of reducing inflected and derived words to the base form is called stemming. We use the Snowball stemmer which is part of NTLK.

Most of the steps necessary for preprocessing data are contained in included in the tokenizer module.

2.4 Vectorization

To be able to cluster articles, we need to assess their similarity based on their shared vocabulary.

Using the TfIDf vectorizer from the sklearn library, we create a matrix where columns are created from the entire vocabulary of all the articles. One column represents one word which appeared in at least one of the articles. Each row corresponds to one of the articles. For each article, TF-IDF marks down how many times each word from the entire vocabulary appears in the given article. Since some of the words appear in most of the articles, making them less unique, TF-IDF weights individual words based how frequent they are in the given article, comparing to all the articles. If the word appears often in a given article, almost never in other articles, it receives higher weight as it’s can be considered to be more characteristic of the given document.

The process of splitting documents into tokens is defined by a tokenizer function we pass into the tokenizer. We do an additional filtering on the tokens which is described in the 2.3.3 Tokenizing and stemming section.

As part of the vectorizer process, we also define that english stop words should not be considered as tokens. Additionally we specify that we’re interested only in tokens which appear in at least two articles and we want to use only single n-grams.

Most of the functions related to TF-IDF and vectorization are contained in the vectorization module.

2.5 Clusterization

After we have our TF-IDF matrix, we can run clusterization algorithm which groups the articles based on similarity of the set of tokens contained within them.

The initial approach used the sklearn implementation of the K-means algorithm. However, due to the amount of articles, the clusterization took several hours. As we needed to run clusterization multiple times, to find out the ideal number of clusters, the final approach uses the sklearn implementation of the Mini-Batch K-Means algorithm which is optimized for larger datasets.

Since k-means clusterization is an unsupervised method, we don’t know how many clusters are there in the data. To find out the most appropriate number of clusters, we rerun the clusterization pipeline with different number of clusters and evaluate the value of the silhouette score for each result. All of the results yield values close to 0, meaning no significant structure has been found. This might be due to relative similarity of all the articles, as they are all related to the topic of terrorism. Based on further analysis of the resulting visualisations, the 10 clusters has been selected as an the best appropriate number of clusters.

Most of the functions related to K-Means and clusterization are contained in the clusterization module.

2.6 Decomposition

The vectorization process yields a matrix with thousands of columns. Since we can imagine each column as one dimension, it's possible to visualize these results to be able to manually evaluate similarity of the articles and the assignment to clusters.

The classic method of reducing dimensionality of the multidimensional dataset is the Principal Component Analysis (PCA). Sklearn provides an implementation of the PCA algorithm.

However, the matrix we get from TF-IDF contains mostly zeros. Most of the words are contained only in few articles. This type of matrix is called sparse, and they are usually use stored differently in memory. Because of that we cannot use the standard PCA implementation and reduce the dimensionality using sklearn implementation of the TruncatedSVD algorithm.

We have to provide the algorithm with the number of components we want our dataset reduced to. Initially, we decompose the dataset into 2 dimension, so that we can create 2D scatter plot visualisation. Later on, we decompose the dataset into 3 dimensions, so that we can create the same scatter plot visualisation in 3D.

Either way, the resulting components are stored so that they can be used later for visualisations.

2.7 Topical modelling

Since the clusterization process failed to provide many insights into similarities between articles, we additionally carry out topical modelling analysis using LDA. The goal is to run extract topics from group of articles from different years, to be able to tell whether there's a significant trend in topics throughout years.

We are using the Gensim implementation of the LDA algorithm. We're looking to extract 20 topics for each year. Visualisation of extracted topics is made by pyLDAvis.
Most of the functions related topics to the topical analysis are contained in the topics module. The visual analysis of topics from different years is done within a Jupyter notebook.

2.8 Document hierarchical clustering

As a last form of analysis, we run a sklearn implementation of the Ward algorithm. The ward algorithm builds a hierarchy of clusters represented as a tree by successively merging clusters with minimum between cluster distance.

2.9 Performance optimization and cloud computing

The initial runs of the clusterization pipeline were taking up to 5 hours. Since we wanted to rerun the pipeline several multiple times, to explore an effect of different parameters, it was crucial to find a way to optimize the performance and set up the processing scripts so that they could be run automatically on a cloud computing parameters.

2.9.1 Architecture

We define the parameters for all the steps of the clusterization algorithm in the main clusterization script. From there, we call the clusterization pipeline passing the parameters object.

2.9.2 Caching results of individual operations

When rerunning the clusterization script, it might happen that we've run a particular step of the pipeline with exactly the same parameters before. For instance, when testing a different number of clusters to find the best silhouette score. The vectorization step of the pipeline, which returns the TF-IDF matrix, will always return the same result.

To save processing time, a result of each operation is stored into a file, using sklearn joblib library. To be able to retrieve the cached results, a document is stored in the cache collection of the MongoDB database. This document contains a path to the object with stored results of the operation, and also query objects with parameters used to run particular operation. This way we make sure, we don't retrieved an incorrect cached results computed with different parameters.

Each module running processing-heavy operations then first checks whether there's a cached result for the same operation and parameters. If there is, it loads the file with serialized results and returns it, instead of rerunning the computation. Thus saving a substantial amount of time.

The caching mechanism is implemented in the caching module.

2.9.3 Storing clustering results

Rerunning the clusterization algorithm multiple time in a time frame spanning days means we have to store results of each run so that they can be reviewed retrospectively.

For each step of the clusterization algorithm, we store the results in human readable form into a csv file. Similarly to the caching module described above, the document with the path to a file and parameters is stored in the results MongoDB Collection.

Unlike the caching module, the dumping of the results into a csv file is done with the Pandas DataFrame to_csv method.

The storing of results is implemented in the results module.

2.9.4 AWS

It proved impossible to rerun many iterations of the clusterization algorithm consecutively on a personal computer. The script takes most of the machine memory and CPU processing, thus rendering it useless for several hours.

For these reasons, a virtual machine on Amazon’s EC2 has been rented for few days. This instance has been of type m4.large, as its specification corresponded closely to a MacBook PRO the clusterization has been developed and tested.

MongoDB instance has been hosted on Mlab.

To be able to upload the latest version of the script into the rented machine, we’ve installed git and created a new remote to push to. The additional setup was then similar to setting up the project on a development machine and is described in the project readme.

The generated csv files with the results of the clustering algorithm were then downloaded from EC2 using the unix SCP command.

3. Findings

3.1 Clusterization

3.1.1 Generated groups

The following groups have been found and interpreted:

a) N. Ireland / Orange Group

Contains only two articles about the Orange Volunteers, a terrorist group targeting catholics in N. Ireland in the 90s.

b) 02. N. Ireland / I.R.A

3,337 articles mostly concerned with N. Ireland and IRA related activity.

c) 03. Israel / Palestine

1,301 articles mostly concerned with the Israeli-Palestinian conflict.

d) 04. Global Terrorism

The largest group of 9,309 articles with no distinguished topics other then terrorist attacks across the globe.

e) 05. War on Terror / Al-Qaeda

1,905 articles mostly published before 2012, mostly reporting on Al-Qaeda inspired attacks and War on Terror.

f) 06. Europe / ISIS Inspired Attacks

713 articles mostly reporting on attacks from 2016 and 2017 with emphasis on the ISIS connection.

g) 07. Syria / Middle East / ISIS

2,346 articles mostly reporting on Syrian war, continuing war on terror in Afghanistan and Pakistan and rise of Isis

h) 08. Afghanistan

Single pre-9/11 of the US administration article asking Taliban to stop funding Bin-Laden.

i) 09. U.S. Attacks / U.S. Politics

484 articles mostly covering attacks happening in the US and US politics.

j) 10. Europe Attacks

2,738 articles mostly covering attacks in the UK and across Europe.

Two main visualisation have been created to allow for exploration of the resulting clusters. 2D scatter plot made with plot.ly and 3D scatter plot made with deck.gl.

3.2.2 Silhouette score

The resulting silhouette for the final parameters is: 0.00455. Rerunning the clusterization pipeline with different number of clusters resulted in even poorer results, sometimes negative.

3.2 Topical modelling

The topical modelling analysis has been carried out for multiple years, always extracting 10 topic groups. Here are top 3 resulting topic keywords found for some of the years:

3.2.1 Topic keywords from years 1990 to 2000

a) arrest, bail, crash, connect, truce, driver, embassi, deton, murder, link, group, attack, shot, jail, flight, factori, bomb-mak ...
b) decommiss, agreement, parti, paramilitari, peac, process, unionist, power-shar, assembl, republican, ceasefir, execut, de, deadlin, disarma ...
c) lawyer, trial, court, cleric, imam, hear, judg, confess, mosqu, consul, appeal, request, asylum …

One of the most prominent topic in the 90s seems to be still ongoing conflict in Northern Ireland.

3.2.2 Topic keywords from years 2002 to 2003

d) burden, neo-conserv, hawkish, imperi, diplomaci, re-elect, injustic, democrat, multilater, resolut, sooner, theguardian.com, forum, soften, legislatur ...
e) debri, bystand, pick-up, holiest, homemad, crater, synagogu, pullout, neocon, blast, noos, militiamen, mourner ...
f) extradit, aysir, courtroom, remand, acquit, magistr, remors, sentenc, ex-pat, clemenc, appel, malici, treason ...

The most prominent of the early 2000s seems to be the neoliberal political movement in the US and related War on terror.

3.2.3 Topic keywords from years 2016 to 2017

g) metro, blast, attack, airport, bomber, suicid, explos, deport, injur, raid, deton, manhunt, blew, wound, accomplic ...
h) educ, religion, cultur, footbal, revolutionari, faith, centuri, basebal, statu, feminist, gender, museum, generat, devout ...
i) firearm, knife, mental, counter-terror, constabl, shooter, polic, plot, mall, commission, vulner, offic, republican, stab, maraud, motiv ...

One of the most prominent topic of the mid 2010s seems to be Islamic terrorism and related conversations about religion and human rights.

4. Conclusions

4.1 Evaluating results

4.2.1 Clusterization

Clustering analysis failed to uncover surprising insights. Seems like the articles are mostly clustered based on place of the attack.

Resulting clusters are not very well defined, with the exception of the Israel / Palestine cluster. This might be due to the general similarity of all the documents, since they all cover the same topic of terrorism.

4.2.2 Topic modelling

The analysis of the evolution of topics throughout years yields an interesting overview of the themes discussed.

While the raw result of the topic modelling analysis is hard to interpret, with the use data visualisation tools, the changing topics and different keywords associated with them are easily explored.

4.2 Conclusion

The document clusterization methods used don’t work very well for documents which are very similar.

Topical modelling proved useful in uncovering general themes across the documents.

Methods of data visualisation proved critical in evaluating results of the all the method of the information retrieval.