MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT REPORT

JAMES LUCAS BAKER

September 19th, 2018

DEFINITION

PROJECT OVERVIEW

The project applies a series of machine learning and natural language processing techniques to analyse news articles. The objective is to take a large body of articles, group them into related stories, then further analyse those groups to attempt to produce a metric for how well the set of articles covers multiple perspectives on the story.

A key inspiration for conducting this project is the present state of journalism, social media, politics and social discourse. The news media in the US are accused of biased reporting. By focusing exclusively on the media outlet whose agenda and biases most appeals to them, individuals can (and often do) subsist on a diet of news reporting that reinforces their own views. As a result, people increasingly both (a) believe their own views ever more strongly, and (b) disdain and dismiss as unsupported the views and opinions of others. This has led to a widespread sense of discord.

In a truly enlightened world, people might engage more fully with a broader set of views and could be more likely to (a) respect other people, and (b) be aware of any biases within the news that they consume, even from their preferred sources.

For example, if one wants to get a clear picture on the current trade war, one should probably read the NY Times, Wall Street Journal, local newspapers from industrial areas in the US that will be directly impacted, and some coverage from Asian/European newspapers. One way to consider this is to say that the text of the news article inevitably reflects the sentiment of either the author or the publisher of that article. It is useful to refer to the dictionary for a definition, even if that definition is far more broad than the meaning in which we use the word here:

Sentiment *from The American Heritage® Dictionary of the English Language, 4th Edition*

n. A thought, view, or attitude, especially one based mainly on emotion instead of reason: An anti-American sentiment swept through the country. See Synonyms at feeling, opinion.

n. Emotion; feeling: Different forms of music convey different kinds of sentiment.

n. Tender or romantic feeling.

n. Maudlin emotion; sentimentality.

n. The emotional import of a passage as distinct from its form of expression.

n. The expression of delicate and sensitive feeling, especially in art and literature.

For our purposes, 'sentiment' will refer to the views of an article's publisher – without implying anything about the merits of the basis for those views. Specifically, we are in interested in the extent to which the article reflects the author having a (stronger/weaker) positive or negative feeling about the events of the underlying story. Conveniently, the Natural Language Processing field of Sentiment Analysis seeks to extract the underlying sentiment from a document.

Discourse on the general topic of media sentiment and bias has been subjective and speculative. This project aims, in part, to use machine learning algorithms to try to define a metric that evaluates how systematic such biases are.

The project makes use of a large set of new articles available via public APIs from 15 media institutions. Conveniently, the data have previously been aggregated and made available on Kaggle:

https://www.kaggle.com/snapcrack/all-the-news/home

Note that this is a dataset and not part of a Kaggle contest. The dataset contains the full text content of over 140,000 articles along with associated metadata. Those metadata will NOT be used in generating the evaluation metrics.

PROBLEM STATEMENT

The easiest way to think of the problem is with a single trivial use case: After reading a news article, the objective is to propose to the user another article for them to read. This article will cover the same story, but from a different journalistic perspective.

This implies the following conceptual steps:

1. Identify a series of new articles covering the same story, but from different media outlets.
2. Rank those articles according to their perspective.
3. Propose the article which has the score which is most different to the original article's.

While the use case mentioned above is very specific, the linkage between sentiment, subjective perspective and ranking has much broader potential applications. Indeed, armed with such a metric, it could prove possible to:

- Compute a score for someone to understand how biased/uniform an article they are reading is.
- Propose additional articles to read on the same story, but with differing perspectives, in order to achieve a more balanced view on the specific story.
- Compute aggregate scores to enable someone to understand whether they are systematically reading from one perspective.
- Measure the nature of media coverage on a specific topic. This might be of use both to media and to political campaigns.
- Apply the solution to other media, such as Twitter feeds, and use the computation to drive marketing and trading insights.

The execution of the project is broken into two main parts:

- Analyse news articles and group them into inferred related stories – this may be thought of as topic mining/clustering. This is implemented in Python and makes use of various Natural Language Processing techniques to pre-process the text in order to increase the amount of significance that can be inferred. It utilises publicly available NLP libraries such as NLTK and spaCy. The implicit clustering operates over a TF-IDF vectorization from scikit-learn.
- Rank and score the sentiment of a series of articles relating to a single story – this is also implemented in Python and uses a different set of NLP libraries (Vader, Google Cloud Platform, and Stanford CoreNLP). The comparison between the various libraries informs the decision on which one to prefer. The results of the sentiment computation are then fed into the newly proposed metric calculation.

METRICS

The nature of this project is such that:

- There is no existing metric by which its success may easily be measured.

- The analysis is significantly subjective – indeed it is an attempt to measure subjectivity.
- Part of the objective is to propose a new metric which might be used for this purpose.
- This score is introduced and covered in detail, later in the report. It is called the **Neutrality Score**.

That said, it is possible to correlate the computed sentiment values with some publicly available data at the level of the publication. In particular, the report will look at how the relative rankings of the articles compare with the Pew Research Center's extensive survey of the perceived ideological placement of each of the major media institutions. While this will provide some insight, it will never be more than an approximation since the institution's perspective may tend to influence the sentiment of the article in one direction or another depending on whether it views the underlying story as good news or bad news. At its most basic, something that is bad news for the political career of Donald Trump might be reported as very bad news by Fox News, or as very good news in the New York Times.

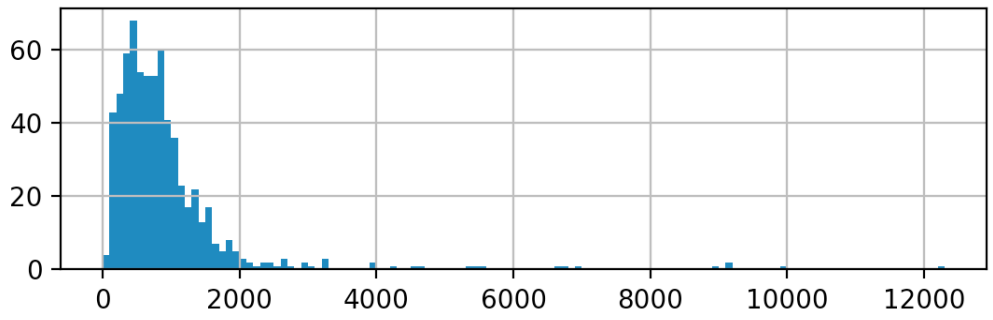
ANALYSIS

DATA EXPLORATION

The following table shows the publishers of the articles in the data set, and the dates to which they pertain.

Article counts by publisher:		**Article counts by date:**	
Breitbart	23575	2017-02-03	399
NY Post	17466	2017-04-13	398
NPR	11966	2017-04-14	383
CNN	11245	2017-04-07	382
Washington Post	11061	2016-08-22	378
Reuters	10710	2017-04-06	378
Guardian	8667	2017-04-05	376
NY Times	7620	2017-01-13	372
Atlantic	6707	2017-03-09	368
Business Insider	6503	2017-02-10	368
National Review	6153	2017-02-08	366
Talking Points Memo	4988	2017-01-12	366
Vox	4942	2016-12-02	361
Buzzfeed News	4528	2017-02-24	360
Fox News	4329	2017-04-12	357

There are many other dates in the dataset covering a multi-year period – although the history is far from complete. Focusing on 2016-09-01 (which much of this report will focus on) the word counts for individual articles are shown in the histogram below. Their mean length is 935. Median is 715. The distribution looks vaguely log-normal. Lengths are generally within 100-2000 words, although there are some outliers. The minimum is just three words (a Washington Post article that was apparently corrupted (it contains only 'Jahi Washington Post'). Conversely, the longest article contains 12,248 words. This is from the Atlantic. They were maintaining a list of where prominent Republicans stood on Trump's candidature – and appended to the article, republishing it, whenever more opinions became known.



The complete corpus comprises 142,570 articles. But this includes articles which are not usable for a variety of reasons. The first step in the process is to programmatically remove articles which are either:

- "Daily Briefing" type articles – these are published by many institutions and contain barely a sentence on each of several stories. Because they include multiple stories they bring noise into the clustering process. And because they contain such a small amount of text on each story, they also don't carry much useful sentiment. Hence they are excluded.
- Extremely short articles – some of the publications send out "articles" which are effectively just notices that they have written an actual article. As such they are essentially just the title/headline of the article and hence communicate very little information or sentiment. These, too, are excluded programmatically.

Once these two categories have been filtered out, 140,460 articles remain. At origin, each article entry contains the following columns:

- ID – a unique numeric identifier for the article. This will serve as the reference index for most of the code.
- Title – the title of the article. This field is not necessarily used by all publishers, and is not used at all within this project as it doesn't provide any useful information.
- Publication – the publisher of the article – e.g. NY Times. This is not used in any of the processing, but is shown for information in some of the reports.
- Author – not relevant for this project.
- Date – publication date of the article. This is used to focus the clustering for specific articles.
- URL – not relevant for this project.
- Content – the text of the article upon which all the processing in the project is based.

An example of a single article is show below:

> ID: 214888
>
> Publication: Washington Post
>
> Content: The Romanian hacker who first revealed that Clinton used a private email address while she was secretary of state was sentenced to more than four years in federal prison Thursday by a U. S. district judge in Alexandria, Va. Marcel Lehel Lazar, 44, known online as "Guccifer," was extradited in 2014 to the USA and pleaded guilty in May to one count each of aggravated identity theft and unauthorized access to a protected computer. Lazar admitted to victimizing about 100 Americans from his home… *(truncated for inclusion in report)*

It is worth noting that there are many encoding errors in the dataset, presumably from articles having been historically moved between systems with different default settings. As a result, there is a degree of corruption which cannot at this stage be undone. For example, any pair of hyphenated words in an original NY Times article has both words completely missing in the corpus. (apparently triggered by them being hyphenated with en dashes which weren't supported in one of the intermediate character sets – such errors and omissions turn out to have only a marginal impact on the results, although there can surely exist cases where their impact would be more profound)

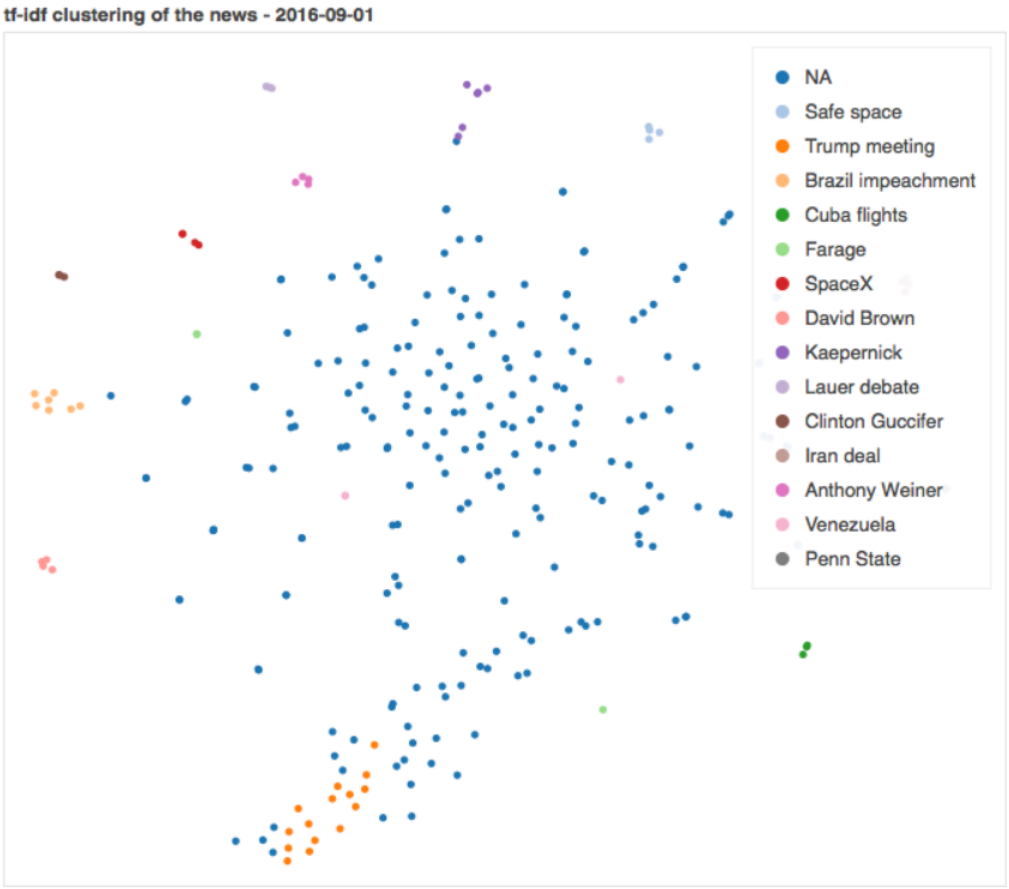
EXPLORATORY VISUALIZATION

The graph below is actually a result of the analysis performed for the clustering, so it may be premature to mention it in a discussion of the data. It will be returned to in more detail later, but it is useful to look at it at this time as it gives a sense of how the topics of the various articles are distributed (in this case for 2016-09-01). Here, each point represents an article from that day's reports. The closer together two points are,

the more likely the articles relate to the same story. This is the result of a high degree of dimensionality reduction in a TF-IDF vector space, which will be explained later.

To be clear, the project's objective is to automatically infer that the illustrated clusters exist (without prior information), and then to select those stories which belong to the clusters and rank them. (the coloring shown here has been added automatically from the output of this analysis)

While the two dimensions here are somewhat of obscure meaning (since they are derived from a dimensionality reduction from a much higher dimension space), they do hint that the clusters should indeed be found.



ALGORITHMS AND TECHNIQUES

CLUSTERING ARTICLES INTO STORIES

Ultimately the clustering will be performed on TF-IDF vectors generated from the articles. TF-IDF produces some reasonable results on the raw articles, but there remains a large degree of noise. To improve the likelihood that the clustering is correct, it is necessary to refine the TF-IDF process. This is done in several steps (which can be switched on/off by the user of the Python implementation):

- Remove corrupt data – as discussed in the Data section above.
- Parts-of-speech filtering – as a general concept it is better if the article text provided for TF-IDF vectorization contains only high value information. The more low-value information it contains, the greater the likelihood that noise may distort the results of the clustering. One way to improve the amount of signal relative to the noise, is to strip out certain parts-of-speech. For example, we might consider that adjectives tell us very little about an article's the topic. The algorithm as implemented

makes use of the NLTK and spaCy NLP libraries to allow for the specification of a list of parts-of-speech which will be included (e.g. Verbs, Proper Names, Common Nouns, Adjectives, etc)

- Lemmatization – words exist in many derivatives of their root. For example, 'walking', 'walks' 'walked', 'walk', etc, all refer to the verb 'walk'. If the words are kept in the form in which they are found, they will not help match the articles in their TF-IDF vectorization. Lemmatization is the process of substituting words with their root – and thus increasing the chances of getting correct matches between documents. The NLTK and spaCy implementations of lemmatization can optionally be applied in the algorithm as implemented for this project.
- N-grams – this refers to the process of grouping consecutive words and including them in the representation of the article fed into the TF-IDF computation. For example, in an article on NAFTA, the phrase "free trade agreement" would normally be represented by the series of three individual words, "free", "trade" and "agreement". If we looked at the DF value for each of those words, we would likely find something small – as they are all quite common words that might occur separately in many different contexts throughout the corpus. By including the 3-gram "free trade agreement", we would have a term that would have a far greater value, in terms of it appearing to contain far more discriminating information when compared with the corpus – and this should help ensure that articles pertaining to free trade agreements are properly associated. The algorithm implementation allows for the specification of N, which may be any integer value from 1 up.
- Constraining length of text considered – it is a journalistic norm that an article's first paragraph – its "lead" or "lede" – should contain the who, what, where, when and how of the story. This is statistically likely to contain the verbs and nouns that most easily capture a reference to the story to which the article pertains – without the noise that increases from including increasing amounts of the articles. Naturally, journalistic traditions are not followed universally, so some care is required in using this option in the implementation of the algorithm.
- Stop word removal – a frequent step in natural language problems is to simple remove words that are known to not contain much discriminating information. Such words comprise a list of Stop Words. They typically include extremely common words such as 'a', 'the', 'Saturday', 'man', etc. The list is encapsulated in an input file.

It's worth noting that while some of the concepts treated here are clearly true in theory, in practice their implementations may be not perfect in all cases – and those imperfections may introduce noise in the clustering results. As a result, it is always better to consider, from practical experience whether or not to make a specific adjustment, and with which library. To facilitate that discovery process, a basic grid search feature has been built into the implementation of the algorithm.

TF-IDF VECTORIZATION AND CLUSTERING

The pre-processed articles are next passed to the TF-IDF vectorizer. It is the values of these vectors that will ultimately be used in the clustering. The TF-IDF vectorization gives, for each article, a single vector with one dimension per term in the dictionary of terms (that dictionary being constructed from the union of the terms across the articles resulting from the preprocessing). The values of the article *d*'s vector in the dimension of a term *t* corresponds to its TF-IDF value as follows:

$$\text{tf-idf(t,d)} = \text{tf(t,d)} \times \text{idf(t)}$$

$$\text{idf}(t) = log\frac{1+n_d}{1+\text{df}(d,t)} + 1$$

Here, *tf(t,d)* represents the number of times the term *t* occurs in document *d.* This is scaled according to how important *t* is in the entire corpus – which is dictated by the inverse document frequency, *idf(t)*, where n_d is the number of documents and *df(d,t)* is effectively the number of documents containing the term.

Several parameters are required in order to run the clustering process (also note that the clustering itself is non-standard).

- DF constraining – this is elaborated on elsewhere in this report. It is a parameter to the vectorization that can be controlled in the implementation of the algorithm. For the purpose of finding other articles on the same story, it may be better to use a value other than 1 – potentially 2, given that it requires at least two documents in order to have a meaningful cluster.
- Article affinity measurement – clustering is typically based on the Euclidean distance between points in the *n*-dimensional space (of article scores per dictionary term). However, the experience from this project suggests that this approach doesn't place sufficient weight on the terms which carry truly pertinent information. As a result, small differences in the score for a non-discriminating term are treated as identical to small differences in the score for a highly important term. To work around this problem, the Euclidean distance between two vectors was replaced with the dot-product of the two vectors. This creates the behavior where different values for non-important terms have extremely small significance for the clustering. And conversely, similar values for important terms have a huge impact. A consequence of this is that the algorithm is effectively seeking to maximize this measurement of affinity, rather than seeking to minimize a measure of distance.
- Threshold for clustering – because the measurement being used for comparing two articles is non-standard, it is not easy to use standard clustering methods. Moreover, those methods turned out to be difficult to use given the naturally-occurring distribution of stories/articles in the real world. Specifically, there is an extremely large number of single-member clusters – a situation which appears less amenable to the clustering approach as most algorithms end up seeking to combine unrelated articles into the same cluster. The best approach thus far has been to provide a single threshold value for the affinity measurement of a pair of articles. Above that threshold the articles are considered related (and part of the same story). Below it, they are considered not related. (experience has shown that a good value lies around 0.25 – with affinity itself being on a scale from 0 to 1, since the TF-IDF vectors themselves are each normalized to unit value 1 in Euclidean distance).

NEUTRALITY / SENTIMENT SCORING

Once the story clusters have been determined, the individual articles within each story are analyzed for sentiment. Those article sentiment values are then fed into a Neutrality Score that expresses how well-balanced the overall coverage of the story is.

There are multiple steps to this process:

- Restore the complete version of the article (from before lemmatization, etc).
- Convert the article into a list of sentences. This is implemented using the NLTK library.
- Optionally restrict the analysis to the first *n* sentences of each article, where *n* is a user input. From experience this seems to work best with a relatively small value, perhaps around 5 or 10.
- Compute the sentiment of each article. This is done with the user's choice between Stanford CoreNLP, Google Cloud Platform's NLP library, and the NTLK implementation of Vader.
- Stanford CoreNLP requires the Java server from Stanford to be installed and running on the network.
- Google Cloud Platform requires the necessary user credentials to exist and to be referenced in the environment variable GOOGLE_APPLICATION_CREDENTIALS.
- The results are translated to lie in -1/+1 based on the meaning of the scale of each of the libraries.

- They are further scaled to take into account the likelihood that the NLP library can return values of the full range. For Google, an analysis of several thousands of news articles suggested that the appropriate scaling was to divide by 0.86.
- Finally, the scaled standardized sentiment values of the articles comprising a story are taken as inputs to compute the Neutrality Score for the coverage of the story.

The Neutrality Score is computed using a new formula that was designed specifically for this project. It is shown below and its motivation and behavior characteristics are then explained:

$$Neutrality = \frac{1}{2}\left(\left(1 - \frac{1}{N}\left|\sum_{i=1}^{N} Sentiment(i)\right|\right) + \frac{1}{N}\sum_{i=1}^{N}\begin{cases} 1, if\ count\left(-1 + \frac{2(i-1)}{N}, -1 + \frac{2i}{N}\right) > 0 \\ 0 \\ 0, otherwise \end{cases}\right)$$

This concept is very loosely inspired by Wikipedia's editorial guidelines, in particular its definition of a "Neutral Point of View" to mean the inclusion of all verifiable points of view without undue weight being assigned to any of them. The proposed formula's main parts are:

- A term capturing the idea that sentiment in the aggregate coverage of the story should not skew either positive or negative. This derives from the mean of the article sentiments. Zero is the no-bias value for the mean. But its range is -1 to +1. So, 1-abs(mean(x)) gives a value between 0 and 1, where 1 corresponds to a perfect lack of skew, and 0 is highly skewed.
- A term rewarding 'balance' of coverage where the score is higher when a greater variety of points-of-view is contained in the articles covering the story. To find such a measure, we consider that each point on the sentiment scale represents a distinct POV. The maximum number of points-of-view that can be represented in the set of article is thus the total number of articles. This term counts the number of articles in each bucket (i.e. range of sentiment values) and punishes the set for each empty bucket. The term has a maximum value of 1.
- The two terms are averaged to yield the Neutrality Score.
- The Neutrality Score is computed in the code via *computePopulationBalanceScoreHistoMean()*.

The objective in including two elements in the Neutrality Score is to avoid giving similar scores to situations such as [-1,-1,+1,+1], [0,0,0,0,0], and [-1,-0.5,0,+0.5,+1]. Sets of articles with those scores should clearly aggregate to very different types of coverage, and do so when the second term of the formula is included.

A perfect score for the aggregate coverage of a story would be 1. This would indicate that all possible 'perspectives' have been reported, and that the net sentiment (across the series of articles) is zero. The worst possible score for the coverage of a story is 0. This would indicate that the only articles reporting on the story all take an identical (and extreme) perspective.

BENCHMARK

Since the project is effectively splt into two parts, the question of a benchmark applies somewhat independently to each of those parts.

TOPIC CLUSTERING

There is no data which identifies a specific real-world event, then lists the articles which recount its story. For example, on the day the Brazilian Senate voted to impeach President Dilma Rousseff, many articles were published about the vote. But there is no pre-existing way to link those articles to indicate that they recount the same story/event. They are simply part of an extremely large, free-floating mass of independent articles. This fact is part of the motivation for the project.

Lacking a ground truth for this article clustering, the only option is to review a large number of news articles and assign them to their relevant stories. This is a manual and surprisingly time-consuming process (which is part of why it is valuable to build the tool in the first place). The following "story map" has been manually built from news reports of 2016-09-01 and will be used in the validation of the inferred clustering:

```
Trump meeting : [151832, 110126, 172078, 48306, 57365, 190512, 26536, 71335, 21499,
23872, 142033, 110133, 23888, 71336, 57366, 71339]
Brazil impeachment : [120639, 80103, 25225, 21502, 57362, 120636, 110141]
Kaepernick : [40617, 40543, 39520, 80109, 80101, 47403]
Clinton Guccifer : [214888, 85803, 47979]
Farage : [37252, 37468, 46175]
Anthony Weiner : [49480, 110144, 142300, 214934]
SpaceX : [38658, 134545, 172095, 214894]
Safe space : [21448, 78169, 78171]
Lauer debate : [43447, 47078, 138709]
Venezuela : [172079, 57375, 190522]
Iran deal : [158005, 48823, 57373, 76343, 120634]
Penn State : [80094, 157527, 214892]
David Brown : [172085, 80096, 141886]
```

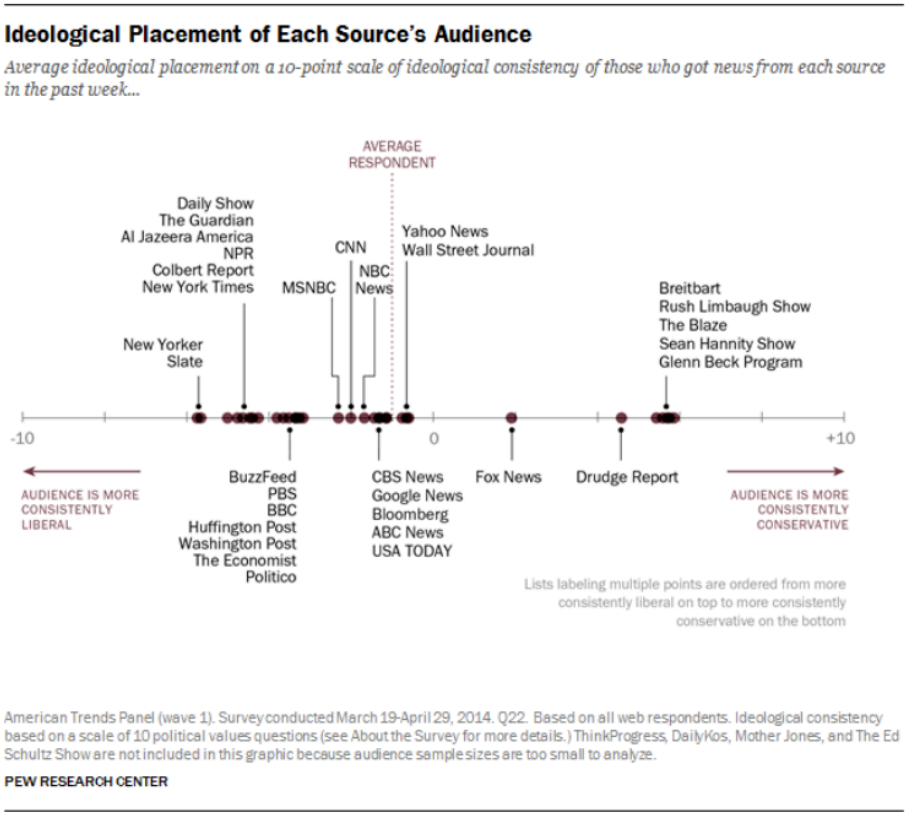
SENTIMENT RANKING / BALANCE SCORING

As previously explained, this is not labelled data from which supervised learning can be performed with its results evaluated directly. However one looks at it, there is no absolute truth for the results. In fact, the whole point is to study relative subjectivity, and ideally to find a way to measure a dimension of it.

The best we can do for a benchmark is to refer to this extensive survey performed by the Pew Research Center:

http://www.journalism.org/2014/10/21/political-polarization-media-habits/

Their report produces the following graphic to summarize the ideological placement of the various news publishing institutions[i].



Thus, the hope is to be able to take news stories (with each story being presented in several articles) and compare the relative computed sentiment of each of those stories, via its publisher, to the graph above.

METHODOLOGY

DATA PREPROCESSING

The data were in the form of three large csv files, totaling nearly 700MB in size. The project loaded them into a Python Pandas Data Frame which greatly facilitated their management and use. Much of the pre-processing mentioned was covered in the existing section. It was implemented using simple operations on the contents on the data frame.

In addition, there were several earlier steps.

- The three separate files were merged into one. See *mergeInputFiles()* in the source code.
- And a series of substitutions was performed for common terms in the corpus. These substitutions were based on a manually compiled file of synonyms that seemed relevant to the articles – e.g. 'Federal Bureau of Investigation' and 'F.B.I.' were replaced with FBI. This has the effect of concentrating the value of references to the FBI in any article, and thus increasing the likelihood of good pairing of articles that referred to the agency. This was implemented in the function *applySynonymChangesToFile()*.
- Punctuation was simplified to the most basic ASCII variants (basic, straight apostrophes, etc) to work around some constraints that were encountered[ii].

The other (previously-described) steps that might be considered pre-processing and are supported by the developed application are:

- Parts-of-speech filtering – utilizing either NLTK or spaCy.
- Lemmatization – utilizing either NLTK or spaCy.
- N-grams – utilizing sklearn.
- Constraining length of text considered – implemented locally.
- Stop word removal – implemented locally.

As a general comment, it is worth noting that while the quality of the data is extremely important, the pre-processing of that data also has a significant impact on how well the clustering performs and on how cleanly sentiment can be extracted.

IMPLEMENTATION

The project is implemented in Python. The key source files are jlbCapstoneClustering.py and jlbCapstoneSentiment.py. These can be run with arguments from the command line. In addition, each is abstracted into a Jupyter notebook, from which it can be run with presentation and explanation of intermediate results. The notebooks are contained in the files JamesBakerCapstonePartOne.ipynb and JamesBakerCapstonePartTwo.ipynb.

CLUSTERING ARTICLES INTO STORIES

The clustering can be followed from the PartOne notebook – or it can be run from the command line as follows:

```
python jlbCapstoneClustering.py --input-file ./data/articles.csv --lemma-
conversion=False --tfidf-max=0.5 --tfidf-min=2 --ngram-max=3 --nlp-library nltk --
pos-list PROPER VERB --max-length 50 --tfidf-norm l2 --process-date 2016-09-01 --
display-graph=True --story-map-validation ./storyMapForValidation.csv --stop-words-
file ./data/stopWords.txt --grid-parameter-file ./gridParameterRangesClustering.csv
--story-threshold 0.26
```

Many of those arguments are optional. The full list can be seen via the help. Most of them refer specifically to parameters required by the NLP preprocessing steps or within the TF-IDF vectorization and hence have been defined earlier. Some others merit clarification:

- input-file – path and name of file containing the news articles to be processed. The file data/articles.csv contains an example (on a far smaller set of dates than the full file).
- process-date – used as a key in the article clustering. It was effectively found that using articles from the full history overwhelmed the clustering and produced bad results. Constraining the articles to a single date for the clustering produces far superior results – and makes sense given that we are processing news articles about specific stories of that day.
- display-graph – used to request that the results of the clustering will be mapped. If requested, a graph is created using Bokeh.
- grid-parameter-file – the clustering can optionally be submitted to a grid search for help in parameter selection. This file contains a list of the ranges over which each of the parameters should be run in order to find the 'optimal' set. The file gridParameterRangesClustering.csv contains an example.
- story-map-validation – this should contain a map of how the articles relate to specific stories. It is a manually created file and is used in the validation of the results of the clustering. (via the creation of output generated at the end of the run) The file storyMapForValidation.csv contains an example.
- stop-words-file – a file containing the list of stop words. See data/stopWords.txt.
- article-id-list – rather than working to see whether the clustering accurately reflects the validation story map, the code can accept a list of article IDs and report the series of articles in the corpus that relate to the same story.
- pos-list – this list uses a small set of custom names for the various parts of speech. It is then automatically mapped to the corresponding NLTK or spaCy names.
- story-threshold – the threshold for the affinity score above which articles are considered to be part of the same cluster, and hence refer to the same story. This is in the range 0 to 1, and 0.26 seems empirically to be a reasonably effective value.

The content of the Python source file has been copied into the .ipynb file and broken down in a way that it can be run in individual steps. The only significant difference between that file and the .py file is that the main( ) function has effectively been deconstructed into steps and distributed to various stages in the notebook. This is to enable the notebook to be more usefully run and understood.

Note that there is scope for ambiguity between the command line parameters and the parameters contained in the referenced files, this is usually well-mitigated.

EVALUATING SENTIMENT AND NEUTRALITY OF STORY COVERAGE

The sentiment analysis stage of the project is covered in the PartTwo notebook. Alternatively it can be run from the command line:

```
python jlbCapstoneSentiment.py --input-file ./data/articles.csv --sentiment-sentences
5 --sentiment-library google --story-map-validation ./storyMapForValidation.csv --
grid-parameter-file gridParameterRangesSentiment.csv
```

The full list of possible arguments can be seen via the help. Some clarifications are provided here:

- sentiment-library – name of NLP library used inside the sentiment calculation – either vader, stanford, or google.
- input-file – same file as the clustering program, see data/articles.csv.
- article-id-list – a list of article IDs comprising a single story, and for which the Neutrality Score and sentiment will be computed.

- grid-parameter-file – as before, but with a different set of parameters, for an example, see gridParameterRangesSentiment.csv.
- story-map-validation – performs the main analysis independently on each of the stories in the map. See the file storyMapForValidation.csv.

Note that some setup is required before this can be run (unless only Vader results are required). First, the Stanford server needs to be downloaded and running locally. It can be downloaded here: https://stanfordnlp.github.io/CoreNLP/.

Once it's available, cd to its directory, then run the following command to launch the server:

```
java –mx4g –cp "*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer –timeout 1000000
```

Before Google Cloud Platform can be used, credentials need to be created and referenced – with the name of the JSON file created in the Google registration process.

To register: https://cloud.google.com/free/.

To generate credentials, follow these instructions: https://www.programmableweb.com/news/how-to-start-using-google-cloud-natural-language-api/how-to/2016/09/01.

Then reference the credential file at the commend prompt:

```
export GOOGLE_APPLICATION_CREDENTIALS=<json key file path>
```

Finally run jlbCapstoneSentiment.py using the command above. The comments previously made for clustering (about the relationship between the .py and .ipynb files, and about the interaction between command line arguments and arguments embedded in input files) apply to the sentiment calculation.

REFINEMENT

The very nature of this work means it involved constant refinement in search of superior results. Due to the inherent subjectivity of the results, it was not possible to maintain a simple, linear log of what was tried and what value resulted. However, it is important to note that each of the NLP processing steps was added progressively to enrich the clustering process, or to enrich the sentiment computation.

The values of the parameters applied in those steps were honed over time, first by trial and error, then when the number of combinations became too great, by implementing the grid search (implemented via sklearn's ParameterGrid). The range of clustering and NLP parameters that were considered include the following (expressed in a grid parameter file):

```
ngram_max,1,2,3
story_threshold,0.2,0.25,0.26,0.27,0.3,0.5
tfidf_maxdf,0.5,0.4,0.6
tfidf_mindf,1,2
max_length,50,100,200
parts_of_speech,VERB+PROPER,VERB+PROPER+COMMON,PROPER,ALL
lemma_conversion,True,False
tfidf_binary,False,True
tfidf_norm,l2
nlp_library,nltk,spaCy
```

For the sentiment/neutrality analysis, the parameterization is less varied:

```
sentiment_library,stanford,vader,google
sentiment_sentences,5,10,20,50
```

Having a good pipeline for experimenting with permutations and generating results was clearly key – but the analysis of those was always to some degree subjective.

RESULTS

MODEL EVALUATION AND VALIDATION

As in prior sections this will be taken in two steps as while related, the two are functionally independent.

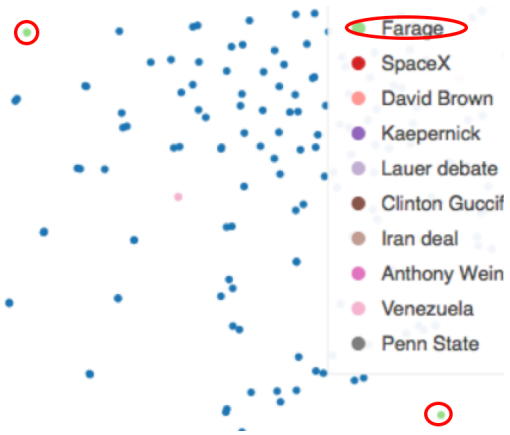
CLUSTERING ARTICLES INTO STORIES

The results of the clustering are contained in the graph shown earlier in this document. It is easier to view this graph interactively in a web browser (use the file *jlbCapstoneClustering.html*).

Recall that the graph is the result of a large dimensionality reduction. It occasionally appears to generate different orientations and projections between seemingly identical runs. Some of the points in the graph may appear behind the legend, so it may prove necessary to drag the graph to uncover those points.

Each point on the graph corresponds to an article in the dataset, in two dimensions (reduced from the high-dimension TF-IDF vector space). Hovering over the point will show its ID, publisher, a little of its content, and the story to which it has been assigned. This works best if the point is to the left of the graph, so it can be helpful to drag the point to the left before hovering over it. A few points can be made:

- Central swarm of NA articles: a large swarm of points exist in the center, marked as NA (meaning that they have not been assigned to a story). Inspection of these articles tends to show that they are indeed unique, in the sense that they are the only articles among the dataset treating an underlying story/event. They could be rendered as individual stories, each with their own color, but that would make the graph harder to read. It is interesting to note that this large swarm dominates the center of the graph. On investigation, the cause of this appears to be that in being the sole articles covering specific stories, they have very few words that carry a high level of information for clustering purposes. In fact, we explicitly chose to exclude TF-IDF terms for document counts less than two. Thus, a term won't even be considered if it only occurs in one document in the dataset. This sounds restrictive, but since the objective is to find articles that pair with other articles, terms only appearing in a single article are effectively noise. Removing them ensures neater clustering. Further, since these sole articles have had their highest TF-IDF terms deliberately ignored, they are left with TF-IDF vectors which contain only small values. This causes them to swarm around the center – where the *x* and *y* axis are close to zero.
- Overlapping points: in some instances, hovering over a point on the graph will result in a list of multiple articles. (in the example html file, look at the Penn State story) Zooming in repeatedly will reveal that the points are typically slightly different, although in some cases there truly are two articles in exactly the same point. This largely reflects the success with which the pre-processing is working – focusing on the *lede*, removing uninformative parts-of-speech, translating to root words (lemmatizing) and removing stop words can actually result in an identical set of words being passed to the vectorizer for two quite different articles. This makes sense if the two articles truly are about the same story.
- Distant neighbors: in some instances, two articles will be colored to indicate that that they are in the same story, even though they are not proximate on the graph. While this appears to be an error, the assignment is technically correct. The points are close in the high-dimensional space, but some artefact of the dimensionality reduction has forced them to appear remote on the graph. (on a prior run, the Farage story contained two points on one side of the graph and one on the other – despite them all being linked as close neighbors in the un-reduced space)



- Ambiguities and errors: there are indeed some articles where it is difficult to cleanly determine if they pertain to the same story – sometimes because the articles cover more than one real world story/event. This was a common feature in the run up to the US Presidential election, and its impact can be seen in the Trump Meeting story (on the right of the sample graph, relating to candidate Trump's meeting with President Peña Nieto). This reflects the fact that many articles about Trump's Phoenix rally mention his trip to Mexico City earlier that day. In this case it is hard to know what the correct clustering should be – it is perhaps an argument for considering that stories can belong to multiple articles, and using Latent Dirichlet Allocation, but my experiments with LDA generally produced inferior results.

The easiest way to evaluate the quality of the results numerically is to physically read all the articles then try to manually and meaningfully assign them to stories. This turns out to be highly laborious. My best attempt (which is surely incomplete) is contained in the story map shown above. This can then be used as a ground truth for the article assignments and the accuracy of the clustering can be testing against this. As can be seen at the end of the Part One notebook, this results in a value around 94%.

EVALUATING SENTIMENT AND NEUTRALITY SCORE OF STORY COVERAGE

Since notions of sentiment and neutrality are subjective, this analysis is difficult. We will consider here the articles pertaining to three stories:

1. The announcement of General Mattis as Trump's Secretary of Defense.
2. The impeachment vote in Brazil.
3. Candidate Trump's meeting with the Mexican president.

The clustering can be run to give the list of articles, those articles are then passed through the sentiment analyzer. The following results were found:

MATTIS APPOINTMENT		
Article	Publication	Sentiment
121883	National Review	0.76
40826	Breitbart	0.22
87167	Fox News	0.22
72790	Business Insider	0.083
193938	Reuters	0.06
113412	Buzzfeed News	0.06
193886	Reuters	0.02
49199	Breitbart	0
43869	Breitbart	0
174613	NPR	-0.083
NEUTRALITY SCORE: 0.633		

BRAZIL IMPEACHMENT		
Article	Publication	Sentiment
25225	NY Times	0.1
57362	CNN	-0.04
110141	Buzzfeed News	-0.12
80103	Atlantic	-0.24
21502	NY Times	-0.32
120639	National Review	-0.325
120636	National Review	-0.38
NEUTRALITY SCORE: 0.548		

TRUMP MEXICO MEETING		
Article	Publication	Sentiment
48306	Breitbart	0.133
57365	CNN	0.05
172078	NPR	0.017
190512	Reuters	0
26536	NY Times	-0.04
110126	Buzzfeed News	-0.05
71335	Business Insider	-0.083
151832	Guardian	-0.12
23872	NY Times	-0.22
21499	NY Times	-0.54
NEUTRALITY SCORE: 0.657		

MATTIS APPOINTMENT

National Review is not covered in the Pew data shown earlier, but is defined by Wikipedia as a conservative publication. Business Insider and Reuters are not known to have specific bias, so we will place those in the middle of the spectrum, with Business Insider slightly to the right simply because it is a business publication, and like the Wall Street Journal, might naively be assumed to lean in that direction.

If we look at the publishers of the Mattis stories, we can sort them according to the Pew report's scale (shown previously): [NPR,NYTimes,Buzzfeed,*Reuters*,*BusinessInsider*,FoxNews,Breitbart,*NationalReview*]

Clearly some of the positions here have been chosen for convenience. But the general sense is that the order of sentiment appears to closely match the list from Pew (and augmented). The exception is the two lower Breitbart articles. These will be commented upon later.

BRAZIL IMPEACHMENT

Here we compare with: [NYTimes,*Atlantic*[iii],Buzzfeed,CNN,*Reuters*,*BusinessInsider*,FoxNews,Breitbart, *NationalReview*]. The main discrepancy to note is that 21502 puts the NY Times out of place. Such complications often exist, either because publications cover multiple perspectives (particularly through the use of Op-Ed pieces) or because the sentiment extraction fails to correctly analyze the text.

TRUMP MEXICO MEETING

The 'Pew benchmark' here is: [NYTimes,Guardian,NPR,Buzzfeed,*Reuters*,CNN,*BusinessInsider*,Breitbart]. The usual caveats apply. Apart from the fact that the NY Times has one article out of place, the other articles seem to conform plausibly.

ANALYSIS

In all illustrated cases Google sentiment values were used. These appear to discern a wider range of sentiment than either of the other NLP libraries supported. The Neutrality Scores seem encouraging – certainly suggesting that a series of perspectives are covered – but it would require much more testing over a larger set of data in order to see how well things truly correlate.

JUSTIFICATION

Google likely produces better results than Vader/Stanford because it is trained on a broader and larger set of text data – and is hence more likely to capture nuance. Vader in particular is trained on Tweets – which is not particularly relevant for journalistic articles, but could be useful if the algorithm was applied to other domains.

Further testing would reveal whether overfitting is a problem – given that the dataset is small and the parameters have been tuned by experience (and validated with Grid Search), this risk is likely non-zero. See comments in the conclusion for how the dataset could be expanded.

The valence of the Neutrality score remains somewhat undetermined. There is surely significant scope for testing and improvement. More rigorous testing could be performed, given more labelled data – see conclusions. It is also possible that the results work well when averaged over many articles, but less well on in individual cases. It is further possible that it works extremely accurately on some stories and yet poorly on others.

Despite these caveats, the results of the algorithm as presented appear germane to the problem posed, and can safely be used – after all, we are doing nothing more dangerous than making news reading recommendations. No car will crash. No illness will go undiagnosed. And anyone sufficiently self-aware to consider further reading to be useful is likely to be aware enough to understand if recommendations are not helpful.

CONCLUSION

FREE-FORM VISUALIZATION

The most effective visualization of the algorithm is the clustering scatter graph shown previously. Its implications have been discussed at length in the preceding sections.

REFLECTION

The project has applied many different concepts to try to answer the original question of 'how to find and propose an article which covers the same story differently'? The clustering of the articles into stories appears to work very well, but the testing would need to be expanded in order to validate this perception. Time and resources are, however, always the main constraint for this type of work.

The Neutrality score does something that appears to have meaning but is clearly neither robust nor foolproof. It would be interesting to advance this work in the future in order to make it more sophisticated.

The end-to-end pipeline was roughly as follows:

STEP 1: CLUSTERING

- Import articles from file into Python.
- Perform synonym simplification using a custom synonym file.
- Suppress corrupted articles and articles likely to contain no/conflicting information.
- Using either spaCy or NLTK perform a series of NLP pre-processing steps on the contents of the articles – including Parts-of-speech filtering, Lemmatization.
- Remove stop words using another custom input file, then restrict article length according to input.
- Using sklearn generate a TF-IDF matrix with the requested TF-IDF term value constraints.
- Process the resulting vectors, ranking their affinity.
- Group articles whose affinity is above the given threshold into "stories".
- Graph the clustering and story space using Bokeh.
- Score the accuracy of those groupings against an input story map file.

STEP 2: SENTIMENT AND NEUTRALITY

- Import a group of articles (pertaining to a story) from file into Python.
- Convert articles into sentences using NLTK.
- Constrain the number of sentences based on the given parameterization.
- Using either Vader, Stanford CoreNLP or Google Cloud Platform, compute the sentiment of each article.
- Scale the sentiments to a standardized scale.
- Compute the Neutrality Score for the coverage of that story and generate results.

A grid search was embedded in order for experiments to be conducted on the parameter values.

IMPROVEMENT

The solution presented in this project is a first step. There are many ways in which it could be enhanced.

- Larger data sets for testing. This is probably the biggest and most important next step. It is also difficult because the work of grouping articles into stories and to ranking them is highly manual and inherently subjective. One possibility would be to have the process guided by the program. It could start by randomly choosing an article, clustering it, then computing Neutrality and sentiment. Those results could be presented to a tester for sanity checking. Even then, though, the amount of work involved remains very large.

- One way of scaling up would be to engage the Mechanical Turk for ranking and grouping articles. This work might be a little laborious and specialised for that to work well, but if it proved possible to divide the problem into a series of trivially comprehended tasks, the results of which might then be aggregated.
- It would be interesting to investigate following an evolving story through time. This would logically be done by taking the TF-IDF vectors of the original article, then clustering the following day's articles and looking for near neighbours there. If any are found, we can then look for near neighbours, too, of these articles – both on that day and on subsequent days. Such an approach has some appeal as stories evolve – e.g. the name of a murder suspect may not be known in the initial reports, but may be an important feature of later reports.
- Some analysis could be done to look for a way to make the sentiment of an article a function of whether the underlying story is good/bad. This might shed more light on any bias in the reporting.
- Republishing of articles is a problem that requires consideration. One of the anomalies in the results discussed above came from Breitbart republishing a Reuters article. Does that represent the sentiment of Reuters? Or Breitbart? Or both? Not managing this correctly could introduce noise and bias
- 'Burying the lead' – the algorithm is currently tuned to look for topic-related information in the first sentences of the article. This corresponds to the lead (or *lede*) and traditionally contains the Who, What, Where, When and How of the story. This is perfect information for our purposes. But when the journalists fail to follow this convention (and "bury the lead"), then the problem becomes harder. More work could be put into studying that case.
- The threshold constant could be replaced with something more sophisticated – although it worked well on the given data. There is intuitive appeal in varying the threshold by story/cluster. This could require de-normalising the TF-IDF vectors.
- The sentiment analysis itself could be made richer by analyzing at a deeper level – for example, excluding reported speech, since that seems to be factual rather than necessarily representing the sentiment of the article itself. However this is germane to another of the problematic Breitbart articles where most of the sentiment was actually coming from the reported speech of one of its journalists – since the article was effectively a series of quotes from the journalist.
- It is worth putting a little though into the scope for the Neutrality score being gamed. In the situation where this concept was built into a major news reading app/site, one could imagine that publications might seek to deceive the algorithm into showing a more neutral score for an extreme article – perhaps by burying the lead, etc. This might be mitigated by comparing the sentiment of the lead with the sentiment of the entire article and discarding articles for which the two diverge too far.
- One further avenue to consider is how user feedback could be incorporated in the ongoing tuning of the algorithm.

---

[i] This is not a measurement of bias per se, more of affiliation.

[ii] In retrospect, it's unclear that this particular step was necessary, but it was part of an attempt to manage the various encoding issues that were encountered.

[iii] The Atlantic Monthly is not classified in the Pew report. Per comments on this page - *https://www.quora.com/Is-the-literary-magazine-The-Atlantic-socially-politically-biased-If-so-to-what-side-left-or-right-does-it-tilt* – the Atlantic is regarded as slightly left of center, but less so than the NY Times.