News Domain Database

The first step of this project was creating a database of news domains. I knew of several resources that publish information about news domains - for example, OpenSources (http://www.opensources.co/), a curated resource for assessing online information sources, uses a combination of a dozen tags (such as 'fake news', 'satire', 'extreme bias' etc.) to assess domains, and this information is available for public use. In addition, Guy DePauw sent me a similar list of sources and tags from PolitiFact, though the tags were slightly different. Finally, Tom De Smedt pointed me to MediaBiasFactCheck.com, another resource that contains 1600+ media sources and categorizes them into classes such as 'questionable', 'right', right-center', 'least-biased', 'left, 'left-center', 'conspiracy', 'pseudoscience', etc.

Naturally, there was a bit of overlap between these three resources, and while having information from more than one of them would be helpful, I still wanted a centralized database for all of the domains. The challenge was that the three resources used different categorization schemas and sometimes referred to the same domains differently (for example, some only had the URL and not the name, some only the name and not the URL, and even then it wasn't always an exact match (ex. www.cnn.com vs cnn.com, The Daily Buzz vs. daily buzz). I normalized the URLs and used various string similarity/distance measures (such as Levenshtein distance) to group together duplicates. In the end, I aggregated all of this categorization in one place and put together a CSV of all domains from the three sources above (~2k domains), along with the categories assigned by each, any additional comments, etc. For those sources that lacked URL information, I made sure to get that information by running the name of the source through the Google API - typically the first returned result was the correct URL, but I did check double-check this manually.

The resulting file has been used for all of the tools I built during the summer. It is available on Github under /data/crawled_data/domain_information.csv. In order to port the tools to other languages, it would be instrumental to find, or create, analogous resources with information about news domains.

Data Collection

My mentors and I agreed that while matching against this database is a simple way of checking the credibility of an article if the source is known, we do eventually want to build one or more text classifiers that would classify a news article based on its content (rather than the source where it was published). The first step for such an endeavor would be, of course, to collect data.

In mid-June, I started to crawl the domains from the compiled file mentioned above. My approach was to tread carefully and thoughtfully in order to ensure "clean", cohesive datasets, rather than to try to automatically crawl all domains and gather as much data as possible. I hand-picked each domain to be crawled, based on information from *MBFC*, *Open Sources*, and *Politifact*, as well as my own judgement - only picking those domains that clearly exhibit characteristics of a potential category (ex. sensationalist, objective, right, least-biased etc.) For some larger news domains, I only crawled certain subcategories, such as “politics” and “world news”. Eventually, I was crawling about 200 sources and collecting over 1k articles daily. By the end of the summer, I have aggregated almost 40k articles. This data is available on Github under /data/crawled_data/crawled_data.csv. Before using this data for training, I cleaned the articles, removing articles that contain video files (these usually have “WATCH” or “VIDEO” in the headline), and also removing metadata, such as share buttons and comments, from the texts.

Source Checker

One of the tools that I developed this summer is a source checker that takes a text, chops into pieces, runs the pieces through web search, and returns the sources that publish this text, as well as a warning if one of the sources is not reputable, along with a graph visualization. For the web search component, I used the Google API that is available through Pattern. The problem with this approach is that the Google API is not free. I experimented with using other, open-source, search engines, such as YACY, but got poor results from them. Therefore, I developed the tool using Google, but took measures to minimize the number of search queries. Below is a detailed description of the tool.

Usage:

python source_checker.py "text snippet” (language)

What it does:

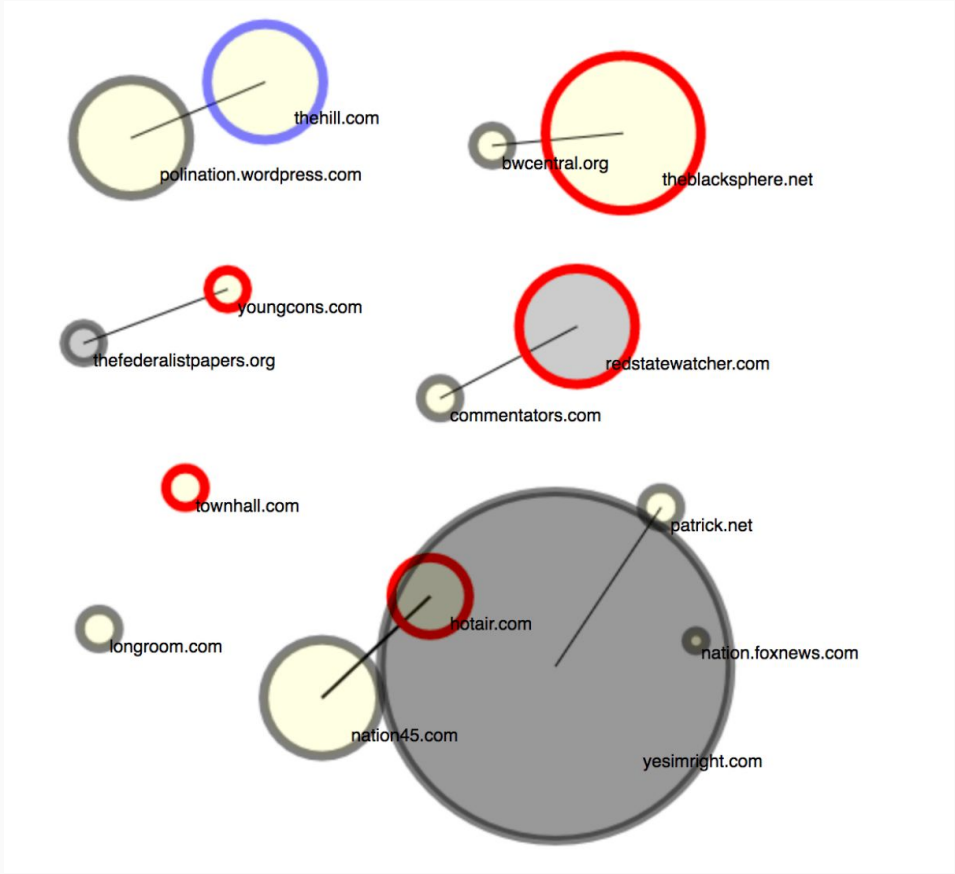
Input:

Snippet of text (as a command line argument)

Command line output:

Domains where parts of the snippet appear, by amount of overlap. If the domain is in our database (i.e. the compiled file of domains from MBFC, Politifact, and OpenSources), the category or categories assigned by these is also returned.

Graph output (rendered through the Pattern Graph module):



- The circles in the graph correspond to returned domains.
- Circle size corresponds to amount of overlap between the input snippet and the domain.
- Circle border color corresponds to bias: blue = left, red = right, green = neutral, grey = unknown.
- Circle fill corresponds to unreliability: black circles are classified by one of the lists as either fake, unreliable, clickbait, questionable, or conspiracy. The blacker the circle - the more unreliable it is.
- Edges that connect circles correspond to overlap of statements - the thicker the edge, the bigger the overlap.

Example:

Input:

"It doesn't get much more liberal than Michael Moore. The guy was speaking at women's marches for goodness sake, and he's so liberal that the women let him. He seems to have built his entire career on career solely on criticizing people (especially conservatives) and for some reason, people love him for it. Unlike most liberals though, Moore isn't discriminatory about who he gets fed up with, and this time, it's his own party. Apparently, the Michigan-born Democrat was ready for a win in the much-anticipated Georga special election and he's a little ticked off that he didn't get it, which caused this ultra liberal to say what the rest of us have been thinking about the Democratic party. Liberal activist Michael Moore lashed out at national Democrats on Wednesday after the party fell short in a special House runoff election in Georgia despite spending tens of millions of dollars on a race that was viewed as a referendum on President Trump."

Output:

HIGH OVERLAP:

- yesimright.com: *questionable, fake, bias*

SOME OVERLAP:

- nation45.com
- polination.wordpress.com
- redstatewatcher.com: *right, bias, clickbait*
- theblacksphere.net: *right*
- thehill.com: *left_center*

MINIMAL OVERLAP:

- bwcentral.org
- commentators.com
- hotair.com: *right*
- longroom.com
- nation.foxnews.com
- patrick.net
- thefederalistpapers.org: *questionable*
- townhall.com: *right*
- youngcons.com: *right*

Architecture:

The text snippet is broken down into n-grams using the Pattern n-gram module. N-grams that consist primarily of stop-words or named entities are discarded (the threshold was determined through experimentation). A sample of the remaining n-grams is reconstructed into the original strings (by re-inserting punctuation) and run through the Google API as an exact phrase (in quotation marks) . Currently, I am using 15-token n-grams (continuous, i.e. across sentence boundaries), and I take 10 evenly-spaced n-grams regardless of the size of the text snippet - so if there are a total of 30 n-grams, every 3rd n-gram is selected, if there 50 every 5th, etc. The reason for doing this is that the Google API is not free, and thus by minimizing the number of queries I minimize the amount of money it costs to use the tool, and my keeping the number of queries constant I ensure that the tool doesn't get overloaded with a very lengthy input that drains the query usage. The returned domains are then rated by the amount of queries that returned that domain (more than 6 out of 10 = "high overlap", 3 to 6 = "some overlap", less than 3 = "minimal overlap"), and matched against our database of news sources. The graph is rendered using the Pattern Graph module.

Multi-lingual feature

This tool has the ability to handle non-English text. I modified the architecture so that the language can be specified as an optional second argument. If NLTK provides stop-words for the language, the n-grams get filtered by percentage of stop-words, but not by percentage of named entities, since NLTK does not provide named entity recognition for non-English. If it's a language for which NLTK does not provide stop-words, n-grams do not get filtered at all. This may result in slightly lower accuracies for those languages, but since I only had English data, I had no way of evaluating the accuracy. One way of improving this tool for non-English data would be to incorporate named entity and stop-word filtering for those langauges where the tools were not available.

Evaluation:

For evaluation, I used 300 article snippets from the data that I have crawled. The snippets are from varied sources (right-biased, left-biased, least-biased, unreliable, etc) and of variable length (100-500 word counts). Since I know where I crawled a given article, there's a true label for each snippet. If the true label appears in the list of domains returned by the Source Checker, the output is considered correct. By this metric, the current accuracy is around 74%. Generally, the longer the snippet the better the accuracy (88% on 500-word snippets). The reason for the misses seems to stem from the Google API: I have noticed that for many queries, the API output doesn't match the output I get when I run the same exact query through regular Google

web search: there are usually much fewer results through the API. In every case that I tried to debug, this was the reason for the miss. I am still in the process of figuring out why this happens and if there's a fix.

However - there seems to be quite a bit of statement overlap between the categories of sources - especially between unreliable/fake sources (which is something that might be worth exploring in more detail). This means that even if we don't catch the exact source that published a given piece of text, the output will still provide a reasonably good idea of the types of sources that publish it (or parts of it).

Sensationalism Classifier

Once I had enough data collected to use for training, I began to develop the text classifiers. The first classifier I developed was for detecting sensationalist vs. objective news. The Sensationalism Classifier directory on Github contains the trained model, as well as Python code to train a model and to classify new data, the data used for training, and a sample of test data (a held-out set) with TRUE labels.

General Approach:

My mentors agreed with me that the best approach to modeling sensationalism would be to avoid bag-of-words lexical features, so that the model is not as sensitive to changes that occur over time in the content of the articles, which are of course natural for news/recent events data. Because sensationalism is more about the style of writing than about content, using features that have been traditionally used in genre detection, such as part of speech tags, in addition to features that could capture the subjectivity of the writing, could potentially yield promising results.

The model:

The final model is a non-linear SVM (implemented through scikit-learn) trained on 8000 sensationalist articles and 8000 objective articles , and it uses the following features (extracted from both headline and article):

- POS tags (unigrams and bigrams)
- Frequency counts of each punctuation mark
- Average sentence length
- Number of all-cap tokens (excluding common abbreviations, and normalized by length of text)

- Number of words that overlap with the Pattern Profanity word list (normalized by length of text)
- Polarity and Subjectivity scores (obtained through the Pattern Sentiment module)

Usage for the Python code:

python SensationalClassifier.py -args

The arguments are:

-t, --trainset: Path to training data (if you are training a model)

-m, --model: Path to model (if you are using a pre-trained model)

-d, --dump: Dump trained model? Default is False

-v, --verbose: Default is non-verbose

-c, --classify: Path to new inputs to classify

-s, --save: Path to the output file (default is 'output.csv')

Dependenices:

- Scikit-learn for modeling
- NLTK for POS-tagging
- Pattern for Polarity, Subjectivity, and Profanity features

Results:

This final model achieves an F-score of 92% (obtained through 5-fold cross-validation).

	precision	recall	f1-score
objective	0.91	0.93	0.92
sensationalist	0.92	0.91	0.92

Below are f-scores for models trained on subsets of features (*text statistics* refers to the combination of *sentence length*, *all-caps*, *profanity*, *polarity*, and *subjectivity* features):

Features	**F1-Score**
Text Stats	0.71
Punctuation	0.85
POS unigrams	0.79
POS unigrams + bigrams	0.86
Text stats + punctuation	0.86
Text stats + POS ngrams	0.88
Punctuation + POS ngrams	0.91
Text stats + Punctuation + POS Bigrams	0.92

This model was trained on features from both the headlines and the articles. Training the same model ONLY on headlines or ONLY on articles results in lower f-scores (82% and 89% respectively).
I took some measures to prevent overfitting - i.e. taking out the name of the source from the articles, not using bag-of-word features, cross-validation, etc. but nevertheless - training and testing data do come from the same sources. One other way to test for overfitting that I have not tried is to separate data by source - training on one subset of sources, and testing on another (the data comes from roughly 140 different sources).

Observations:

These are some observations about sensationalist news, based on most predictive features. These align pretty well with my intuitions.
- Sensationalist headlines and articles more likely to contain exclamation marks
- Sensationalist articles more likely to contain question marks
- Comparative and superlative adjectives/adverbs, interjections, and conjunctions more present in sensationalist news than in objective news
- Sensationalist news more likely to contain words in all-caps

- Sensationalist news have more words indicating subjectivity as well as words indicating negative sentiment.

Resources necessary to port this tool to other languages:

- POS tagger
- List of common abbreviations
- Profanity list
- A module to calculate subjectivity and polarity of text

Bias Classifier

Developing the Bias classifier has proven to be quite a challenging endeavor. The precision and recall numbers are definitely lower than desirable, thus this classifier needs further development.

The Bias Classifier directory on Github contains the trained model, as well as Python code to train a model and to classify new data, the data used for training, and a sample of test data (a held-out set) with true labels and the classifier scores. In the training data, label '0' corresponds to the 'least-biased' class, '1' corresponds to 'left', and '2' corresponds to 'right'.

Just like the sensationalism classifier, this classifier takes as input a 2-column CSV file, where the first column corresponds to the headlines and second one corresponds to the article texts.

Usage for the Python code:

python SensationalClassifier.py -args

The arguments are:

-t, --trainset: Path to training data (if you are training a model)

-m, --model: Path to model (if you are using a pre-trained model)

-d, --dump: Dump trained model? Default is False

-v, --verbose: Default is non-verbose

-c, --classify: Path to new inputs to classify

-s, --save: Path to the output file (default is 'output.csv')

Output:

The output is a number between -1 and 1, where -1 is most left-biased, 1 is most right-biased, and 0 is least-biased.

Data:

The articles come from the data I have been crawling - a hand-picked subset of sites that were labeled as "right", "right-center", "left", "left-center", and "least-biased" by mediabiasfactcheck.com. I used one subset of sources for the training data and a different subset of sources for the testing data in order to avoid overfitting. I also trained a separate model on all of the sources I had available - since it is trained on more data, it may perform better. This model is also available in the Github directory under the name "trained_model_all_sources.pkl".

It is worth noting that articles from 'right-center' and 'left-center' sources often exhibit only a subtle bias, if any at all. This is because the bias of these sources is often not evident on a per-article basis, but only on a per-source basis. It may exhibit itself, for example, through story selection rather than through loaded language. For this reason I did not include articles from 'right-center' and 'left-center' sources in the training data, but I did use them for evaluation.

Architecture:

The classifier has a two-tiered architecture, where first the unbiased articles are filtered out, and then a second model distinguishes between right and left bias. Both models are Logistic Regressions based on lexical n-gram features, implemented through scikit-learn.

Features:

Both models rely on bag-of-word n-gram features (unigrams, bigrams, trigrams).

Results:

The output is a number between -1 and 1, where -1 is most left-biased, 1 is most right-biased, and 0 is least-biased. For evaluation purposes, scores below 0 are considered “left”, above 0 are considered “right”, and 0 is considered “least-biased”.

As previously mentioned, along with the 3 classes that are present in the training data, there are two addition in-between classes that I used for evaluation only.

In order to be counted as correct for recall, right-center can be predicted as either 'right' or 'least-biased', and left-center can be predicted as 'left' or 'least-biased'. In addition, when calculating the precision of the 'least-biased' class, 'least-biased', 'right-center' and 'left-center' true classes all count as correct.

Class	**Precision**	**Recall**
Right	45%	82%
Left	70%	71%
Right-center	N/A	70%
Left-center	N/A	60%
Least-biased	96%	33%

Note:

Unlike the Sensationalism classifier, this classifier relies on lexical features, which may be specific to the current political climate etc. This means that the training data might "expire" and as a result the accuracy could decrease.