News Sentiment Tracker

A Targeted Opinion Mining Interface

Andrew Wesson | Prashanth Rao

1 Introduction

More than 2.5 billion people read online or print news articles on a near-daily basis. Unlike more ephemeral sources of information such as Tweets or videos, the effect of strongly positive or negative news coverage of a target can linger for long periods after a triggering event. Considering this unparalleled global reach and scale of news organizations, the ability to gain insights into large-scale shifts in perception of the press towards a target can have significant commercial implications, and empower relevant decision-making personnel in organizations and governments.

Prior work in this area has focused on large-scale news sentiment modelling [1] or sentiment analysis of financial news articles [2] using binary classification. To our knowledge, there does not currently exist a system that can efficiently target the coverage of specific entities within an article's content for fine-grained sentiment.

2 Problem Statement

Our goal in this project is to build a commercially applicable automated system using NLP that can perform end-to-end data processing and fine-grained sentiment analysis on news articles to identify key events that trigger a change in sentiment toward a target entity. We aim to provide an in-depth breakdown of temporal sentiment trends and provide aggregated information to assist relevant personnel (in marketing, public relations or product management) in taking appropriate action during a period of rapid shift in sentiment.

One of the main challenges in building such a system is that performing a *targeted* search of keyword queries can be very expensive over several hundreds of thousands of news articles (each thousands of words long). Once relevant articles are identified, it also becomes necessary to narrow down on the exact mentions of the target entity within the article itself, following which a sentiment analysis towards the target can be performed.

There are also numerous challenges from an NLP perspective, such as dealing with long, complex sentence structures and ambiguous usage of words (for e.g. words with multiple meanings). News articles are known to wildly vary in their perception of a target (within an article itself), so a simple binary sentiment classification of the whole article (or just its headline) would not provide sufficient depth in meaning to the end user.

3 Data Science Pipeline

To address the above challenges, we divide our data science workflow into three main stages, as shown below.

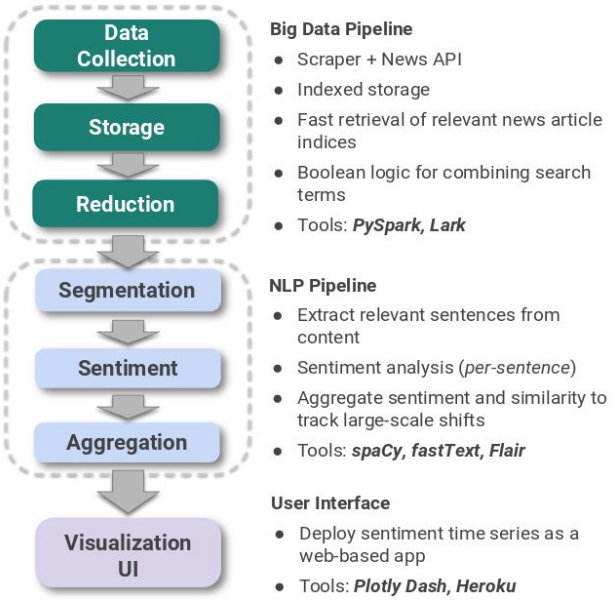

Figure 1: Data science pipeline

3.1 Big Data Pipeline

Our big data pipeline is designed to enable efficient search of the news article database for

relevant content pertaining to our query. To do this, we build an inverted index to map each query's words to the articles they appear in, and a query engine to enable Boolean searches using this index.

3.1.1 Ingestion

As documents get added to the system, we perform some preprocessing in Apache Spark in order to prepare the documents for efficient search. Currently, we store the results of this preprocessing in HDFS, separate from the actual documents themselves. We perform some work upon document ingestion to reduce the amount of work needed to perform the search.

3.1.2 Query Engine

Single-term queries are not powerful enough to properly retrieve documents which pertain to a specific entity. However, by combining these single-term queries with union, intersection, and difference operators, we can create much more useful queries. To facilitate this, we built a query language that allows arbitrarily long combinations of queries using binary operators. We also allow for queries to specify a minimum number of occurrences of a word, rather than retrieving any articles with a non-zero count. We parse the query into a binary expression tree, which we then use to generate Spark code to execute the query.

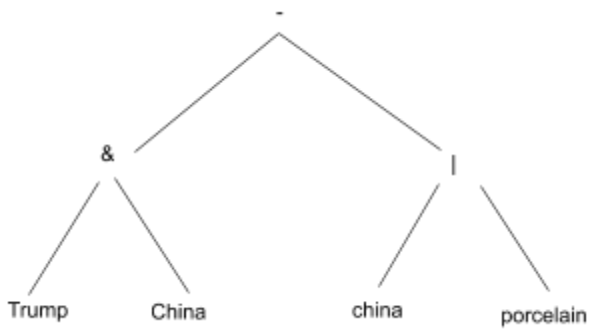

Figure 2: Expression tree for the query "Trump & China - (china | porcelain)"

Queries are performed in batches, multiple queries can be executed at once. The results of the query are returned as a JSON object, with the query strings as keys and the resulting list of indices as values.

3.2 NLP Pipeline

Our NLP pipeline is applied on only the (relatively small) subset of articles that directly mention our target query. To do this, we utilize the article IDs identified from the prior big data pipeline and focus on just the relevant articles (for a particular query) for sentiment analysis.

3.2.1 Sentence segmentation

In this stage, we parse the news article content (which can be thousands of words long) and extract only those sentences that explicitly mention the target query (in part or in full). This is done by identifying "sentence boundaries" using a dependency parse from a trained statistical parser. In our case, we apply the sentence boundary detection model from the NLP library spaCy.

For example, to track mentions of "United Airlines", we extract not only those sentences which mention the exact string itself, but also those that mention just the word "United". In case of a specific product, such as "Samsung Galaxy Note", once we identify the correct articles that mention this product, we also extract sentences that mention just "Samsung", or "Note".

3.2.2 Sentiment analysis

A trained sentiment analysis model is then run on *each sentence* of the content pertaining to the target. To train our sentiment model, we take into account the fact that in many cases, news coverage towards a target topic can express fine-grained sentiment. Rather than treating this as a binary sentiment classification task, our approach uses 5-class sentiment prediction - the sentiment of each sentence is predicted on a scale of 1 to 5, where 1 is very negative and 5 is very positive.

To make aggregating the scores easier, and to compare the scores across different models more effectively, we normalize the sentiment scores to be continuous in the range [-1, 1], with -1 being very negative and +1 being very positive. Consider the below example snippet from our dataset, pertaining to the target entity **Liberia**.

'Monday, Michelle Obama and her daughters, Sasha and Malia, begin their lightning tour of Liberia and Morocco to promote the Let Girls Learn initiative.' 'Massa David, a tall, lanky 15-year-old in the sixth grade, would like to take Obama on a tour to show her some of the problems with Liberia's infrastructure.' 'Poverty, sexual exploitation both in and outside Liberian schools and teen pregnancy are major factors that stop girls from completing their education in Liberia.'

In the above example, each sentence contains a different level of sentiment towards Liberia. Our hope is that a per-sentence scoring approach is better able to disambiguate the overall perception of targets in cases such as these.

3.2.3 Sentiment aggregation

The stored sentiment values (per sentence) for each article are then aggregated in two separate ways *per article*:

- Mean sentiment value (i.e. "score")
- Mean sentiment standard deviation

Articles that possess a high score but a low deviation are likely more polar in their coverage towards a target. On the other hand, articles that possess both high score and high deviation are likely mixed in sentiment towards a target [3].

In periods of large-scale shifts in sentiment towards a target (e.g. in case of a scandal or controversy), there might be multiple articles written by many publications towards the target on the same day. Our system performs an additional aggregation step in such cases, taking the average of all mean scores and deviations for that particular day. All the computed aggregated scores and deviations are stored as a time series (sampled daily).

3.2.4 Similarity aggregation

An additional step in our NLP pipeline is to compute the similarity aggregation between publications. "Similarity" in this case is defined in terms of content vocabulary and keywords pertaining to a target. To do this, we featurize the extracted content using a Term Frequency-Inverse Document Frequency (TF-IDF) method. This gives us feature vectors containing the keywords of the relevant content pertaining to the target.

Using the TF-IDF matrix, we calculate the pairwise cosine distances in the form of a distance matrix. This tells us how similar or different the coverage of the target is, based on its content. Articles that cover very similar content (using similar keywords) towards a target entity have smaller pairwise cosine distances, while articles that cover very different topics can have larger pairwise cosine distances. These distances are then aggregated as mean distances per publication.

3.3 User Interface

To make the results intuitive to the end user, we designed an interactive web-based UI that allows users to inspect the time series and coverage distribution more meaningfully. The sentiment analysis results are output to a tabular format and visualized on a dashboard. An example use case of our dashboard is shown here.

The primary goal of the dashboard is to allow the end user to narrow down on specific periods of heightened news coverage (either positive or negative). In case of a high-profile event such as a scandal, it can be very useful to be able to focus on specific news content from a wide range of publications so that appropriate countermeasures can be taken by a person or organization to alleviate the situation.

4 Methodology

Our application uses the following libraries from the Python big data/data science ecosystem.

- *PySpark*, *Lark*: Data indexing/reduction
- *Pandas*: Data cleaning and analysis
- *spaCy*: Sentence segmentation and lemmatization
- *TextBlob*, *fastText* and *Flair*: Sentiment analysis
- *Matplotlib* and *Plotly*: Visualization
- *Plotly Dash*: UI dashboard
- *Heroku*: Web app deployment

4.1 Exploratory Data Analysis

We used the "All the News" dataset (obtained from Kaggle) to build our NLP modules and train our sentiment classifier. This dataset contains 143,000 news articles from sixteen U.S. publications, the breakdown of which is shown in the below chart (the numbers next to the chart indicate percentage values of article counts per publication).

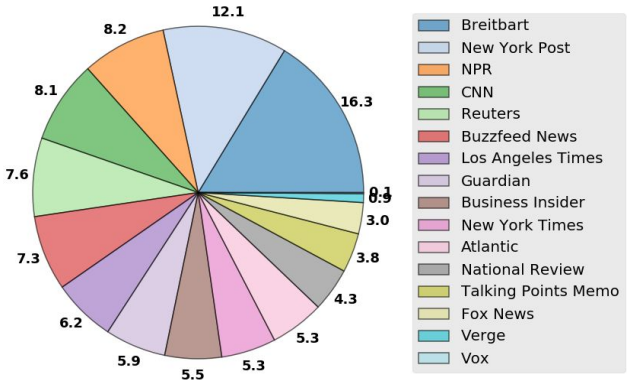

Figure 3: Article breakdown per publication

Most articles in this dataset are from the years 2014-2017, shown below.

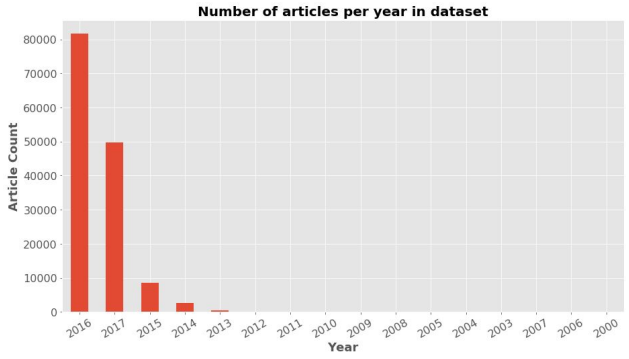

Figure 4: Article counts per year

The median article length is below 400 words in the case of *Talking Points Memo* and *Business Insider*. On the other hand, publications like *Vox* and *New York Times*' median length exceed 1,000 words.

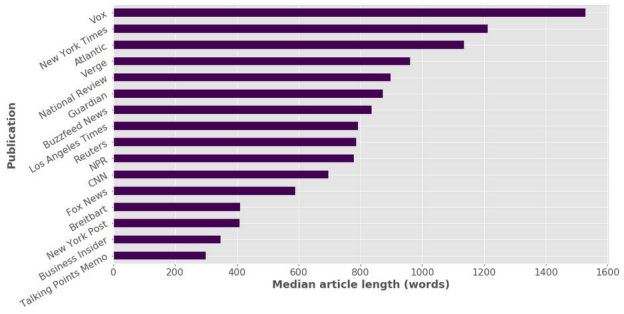

Figure 5: Median article length (words)

4.2 Data Indexing and Retrieval System

4.2.1 Inverted Indexing

In order to enable efficient document retrieval, we want to be able to identify which documents contain specific words. This is done in our system using Apache Spark. For each document in the dataset, we tokenize the document and count the number of occurrences of each token. Then, we pivot each of the individual word counts into a key-value pair such that the word is the key, and the document and the count are the value.

We then use PySpark aggregation operations to group the entries by word, and combine all of the counts and IDs for documents in which the word occurs into a single row of the resulting RDD. This allows for much more efficient retrieval of the IDs of all documents containing a given word compared to doing a text search on every single document.

4.2.2 Query Parsing

To parse our boolean queries, we used the parsing library Lark. We designed a simple context-free grammar to parse queries into binary expression trees in a left-associative fashion, with parentheses available to bypass this associativity. Using Lark, we convert queries into a parse tree, which we can then traverse to generate the Spark code needed to execute the query. Lark allows us to parse the expressions very efficiently with relatively low programming effort. Of all the context-free grammar parsing libraries we considered, it seemed to be the easiest to use without sacrificing performance.

4.2.3 Query Execution

To execute a query, there are two major phases. The first is to fetch the index entries for each term in the query (regardless of what operators are applied to them) and to re-invert the index. The result is, for each article containing at least one of the search terms, a bag of words containing the number of occurrences of each search term within the article.

Then, we transform the query into a predicate that, when given the result of the previous step, returns "true" if the article satisfies the query. To do this, we use Lark's built-in Transformer class to convert the expression tree to a predicate function. For the leaves of the tree (terms), we create a function that checks that the term occurs at least once (or in the case of greater-than searches, more than the specified number of times). Then, for each of the binary operators, we evaluate the two child functions and join their result with the Python equivalent of the specified operator.

Table 1: Query to Python Operator Conversions

Query Operator	Python Operator
&	`and`
|	`or`
-	`and not`

The result is a single predicate function that we then pass to Spark to filter out articles that match the query. Then, we collect the IDs of all the articles satisfying the query, and return JSON objects of the queries/indices as our result.

4.3 Sentence Segmentation

After obtaining the indexing system's output to narrow down our search to just the relevant news articles, we utilized spaCy to break each article's content into its constituent sentences. As part of this step, we also used regular expressions to clean "dirty" text in each sentence, i.e. unwanted symbols and artefacts from the scraping process that could confuse our sentiment model downstream. In our current implementation, we did observe some cases where both the sentence boundaries and text cleaning were not perfect. We identified ways to improve both using rule-based approaches, but did not pursue them in this project due to time constraints.

4.4 Duplicate Article Removal

A common occurrence in news publications is that an article written by a global news agency such as Reuters is reused (with permission) by other news publications. This can lead to multiple occurrences of duplicate articles published on different days, that cover the same topic using the same (or very similar) words. To avoid these articles skewing our mean sentiment scoring metric, we implemented an NLP routine that checks for duplicate content *after* the relevant sentences pertaining to the target are extracted. We used spaCy to *lemmatize* (i.e. reduce to its root form) the extracted sentences for each article. The lemmatization step also removes stop words (common words that don't add value) and punctuation, so minor changes in article syntax are ignored during comparison.

4.5 Sentiment Analysis

4.5.1 Choice of sentiment model

Since modelling fine-grained sentiment of targeted news content is quite a complex task, we were mindful about the inadequacies of typical rule-based sentiment models. To more rigorously analyze our framework's performance, we implemented and compared three separate sentiment analysis models.

- ***TextBlob***: A rule-based approach that utilizes "polar" words from a pre-existing corpus, and considers linguistic aspects such as negation, modifiers etc. to deduce a sentence's overall sentiment.
- ***fastText***: Uses pre-trained word embeddings to fine-tune a classifier on a 5-class sentiment dataset (Yelp reviews) and make per-sentence sentiment predictions on our news content.
- ***Flair***: Uses a sophisticated pre-trained neural language model with contextual word and string embeddings, that is fine-tuned on the same Yelp 5-class

review dataset and then used to make predictions on our news data.

The three approaches above are in increasing order of complexity and run time - both TextBlob and fastText run on CPU, but the Flair model requires a GPU with a significant amount of memory - all our experiments with Flair used an NVIDIA P100 GPU with 16 GB of memory.

4.5.2 Sentiment model training

The Yelp 5-class public dataset [4] is a very large dataset containing reviews of ratings (5.8 million of them) on a scale of 1 to 5. For the purposes of model training, we take the star-rating to be similar in nature to sentiment, since a low star-rating corresponds to a low opinion of a restaurant or person.

To train our fastText model, we used the command line utility provided by Facebook Research (the developers of fastText) [5] along with a preprocessed version of the labelled Yelp review data. Numerous hyperparameters were tried and the model with the best accuracy was chosen to run our predictions on the news dataset.

The Flair sentiment classifier was trained using a modification to the original source provided by Zalando research labs (the makers of the Flair NLP library) [6]. Flair allows the use of "stacked" embeddings, where word embeddings from a pre-trained model (such as GloVe, BERT or ELMo) can be concatenated with Flair's own contextual string embeddings.

In our experiments, we stacked ELMo [7] word embeddings (by Peters et al.) with Flair's forward and backward string embeddings, which we believe is highly relevant to news data. This is because both ELMo and Flair's bidirectional LSTMs were pre-trained on a large news text corpus, so we expect that the language models already encode a significant amount of knowledge from the typical vocabulary seen in news articles.

4.6 Visualization

4.6.1 Time series plots

The mean scores and deviation from each sentiment model are obtained as a sparse time series, which we visualize as shown below.

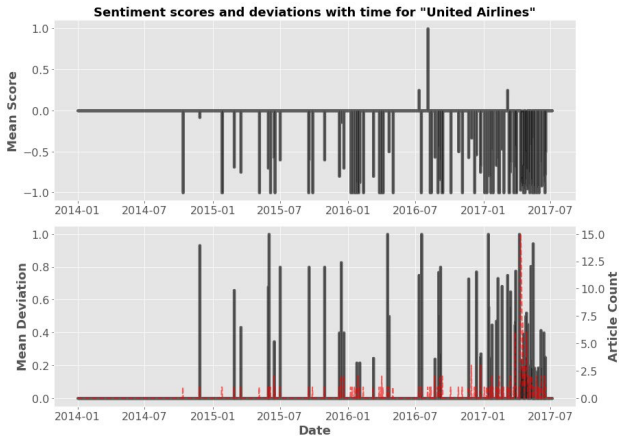

Figure 6: Mean sentiment score/deviation time series for the query "United Airlines"

In figure 6, the top plot shows the mean sentiment score on each day in the period 2014-2017, for the query "United Airlines". The lower plot shows the mean deviation over the same period. There is a period of increased activity (i.e. high density of bars) in the early part of 2017, and the red line shows a spike in the number of articles written about "United Airlines" during this period.

To further inspect these trends, we convert the above 1D time series to a 2D calendar heat map.

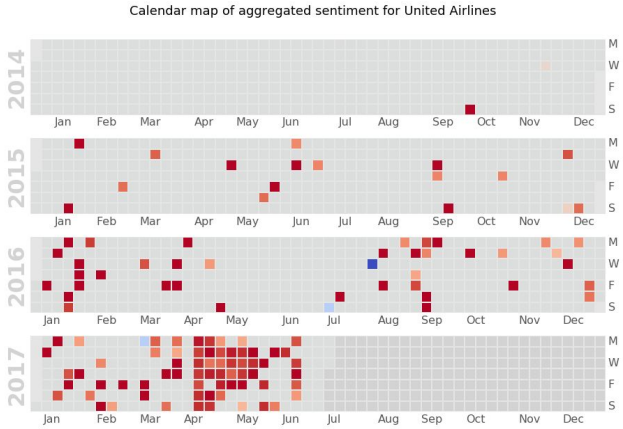

Figure 7: Calendar heat map for "United Airlines"

The calendar heat map allows us to more effectively visualize the distribution of strongly

positive/negative sentiment through a tiled representation spread over a multi-year period. On each day, red squares show strongly negative sentiment while blue squares show strongly positive sentiment (aggregated by the mean value for that day). Based on the above plot, it is clear that the large cluster of red squares in early 2017 means there was a significant amount of negative press coverage of United Airlines during this period.

4.6.2 Distribution plots

We also visualize how the positive and negative coverage towards a target are distributed across publications. This is done by separating the extracted content based on its sentiment score and then grouping by publication.

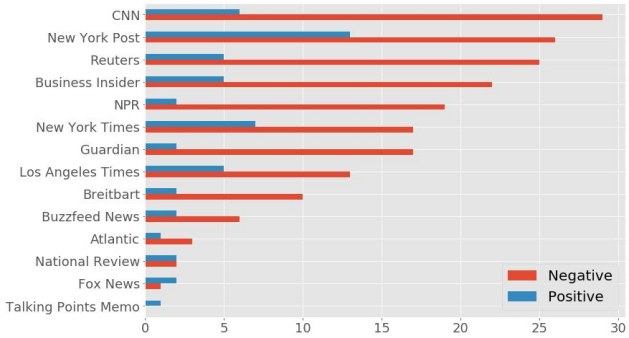

Figure 8: Breakdown of article count per publication

In figure 8, it is clear that CNN, New York Post and Reuters wrote the most negative content about United Airlines, whereas Fox news barely wrote about United Airlines at all.

To judge the similarity between each publication (based on keywords used), we use the computed TF-IDF and mean cosine distance matrices as described in section 3.2.4 and transform them to Euclidean space.

Figure 9 shows these similarities between publications that wrote about "United Airlines". The bubble size indicates the number of articles written while the distance between the bubbles indicates how similar or dissimilar they are in terms of word features. For this example, Atlantic, Fox News and Talking Points Memo are the furthest from the rest - however, this could primarily be because there were very few articles written by these publications in favor of or against United Airlines in our dataset.

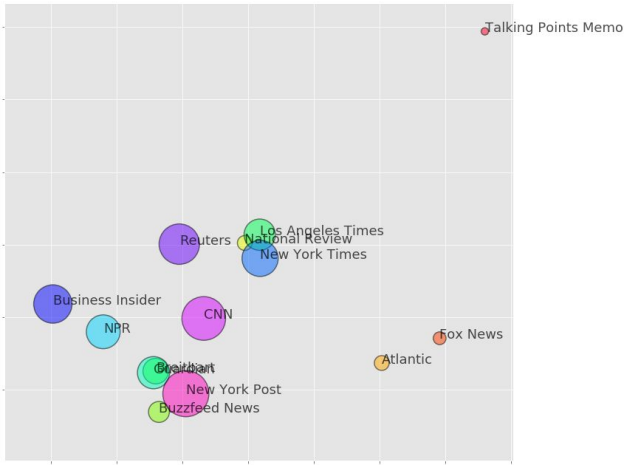

Figure 9: Mean cosine distance per publication

Other observations from the plot agree with logic - Buzzfeed and New York post are spaced close together, as one might expect. Both Reuters and New York Times wrote a similar number of articles as did New York Post, but they are both positioned relatively far from New York Post, meaning that New York Post's content on United Airlines was different (in terms of the language used or topics covered). Following an initial inspection of these similarities, a deeper inspection of the actual content written by each publication can be done as required.

4.7 Interactive Dashboard

We developed a UI dashboard using Plotly Dash to facilitate easy and interactive exploration of sentiment trends per query of interest. Dash is an interactive Python framework for building web applications, written on top of Flask, Plotly.js (which itself is built on top of D3.js) and React.js. By combining the speed and flexibility of ReactJS to manage the flow of data with the power and aesthetics of Plotly/D3, Dash combines the best of both worlds and greatly increases the ease with which data scientists can build complex visualizations.

Once we designed an interactive UI with the relevant visualizations for news sentiment data exploration, we deployed it as a web application using *Heroku*, a container-based cloud platform as a service (PaaS) framework that helps users manage, distribute and scale their applications. An example use case for our app is shown here. We believe that the combination of Plotly Dash

and Heroku make our system easily deployable for a range of use cases, and also convenient to update and maintain for custom tasks.

4.8 Enhancements for Future Versions

In this section we highlight a few features that we were unable to implement in this project, for time and cost reasons.

4.8.1 Data input for continuous news feed

There are numerous news APIs available for a fee that can automatically pipe the latest relevant news articles pertaining to a target topic (for example newsapi.org). To maintain data quality *which is of paramount importance*, it makes sense to rely on a specialized tool optimized to provide curated, targeted news feeds at scale, and use it to feed data to our sentiment tracking system.

4.8.2 Database management

The current implementation of our system uses PySpark HDFS and structured output formats (CSV) during the transfer of data from the retrieval system to the NLP pipeline. To facilitate efficient retrieval and storage of data downstream and to streamline the deployment of the visualizations to a remote web server, we would expect to utilize database systems. The below figure shows an example end-to-end pipeline.

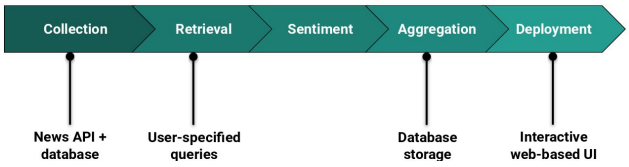

Figure 10: Layout of end-to-end pipeline

For the upstream data handling steps (indexing and retrieval), a NoSQL database system (such as Cassandra or MongoDB) would be apt. The downstream step after aggregation could involve data storage through SQLite (assuming the condensed output is small enough to be stored in memory). In case the output is sufficiently large, a production-grade PostgreSQL service can be utilized. Web deployment services such as Heroku provide ready-to-use production-grade database systems that can power significant amounts of data transfer efficiently, so we foresee our application being scaled up via this framework.

4.8.3 Dynamic updating

Our current system considers the entire time-span for which we have data (from the beginning of time). In a fully productionized system, we expect that we would have our data and visualization pipelines use past-computed results and only update the database dynamically with any new data that comes in. The dashboard would also be plugged into this dynamically updating database, and reflect the trends of any new data on a daily basis. This would greatly reduce the computation time for the sentiment prediction since on average, only a few hundred articles (at most) would arrive per target on a particular day.

5 Evaluation

We devised a two-pronged approach to evaluate the performance of our NLP pipeline.

- A *quantitative* measurement of sentiment classification accuracy and F1-scores on the Yelp review test set.
- A *qualitative* judgment of temporal sentiment trends using queries from real-world scandals/controversies.

5.0.1 Quantitative evaluation of fastText

We ran multiple instances of sentiment model training using fastText, testing a variety of hyperparameters. The biggest benefit of training using fastText is its speed - it can be trained on millions of samples on multiple CPU threads (within minutes). Some of our experiments and their impact on test F1-scores for the Yelp 5-class dataset are shown below.

Table 2: **fastText** training experiments

Training samples	*Max LR*	*Epochs*	*Pretrained model*	*Test F1 score*
40k	0.25	5	None	**0.55**
40k	0.25	10	None	**0.56**
40k	0.25	5	wiki-news	**0.60**
5.8m	0.1	5	None	**0.64**

When comparing test F1-scores across different cases, the size of the test set was reduced (after stratification) to 10% that of the training set. For all training scenarios using fastText, bigram models, i.e. models that consider pairs of neighbouring words for the input vectors, were found to give better F1-scores than unigram models. It was also observed that the model was overfitting when running for many epochs, so the maximum epochs were capped at 5 for most runs. Using a pre-trained model (wiki-news) was not that beneficial in improving the model accuracy, whereas increasing the number of training samples to the full Yelp dataset had a bigger impact on F1-scores and accuracy.

5.0.2 Quantitative evaluation of Flair

When training our Flair sentiment model, we were limited by the framework's inability to read in very large text inputs. This is because deep learning text classifiers are typically limited by the amount of text that can fit into GPU memory. Our experiments with different batch sizes did not yield much success, hence we opted to train the Flair models on a small subset of the Yelp review dataset.

Table 3: **Flair** training experiments

Training samples	*Hidden size*	*Epochs*	*Embeddings*	*Test F1 score*
8k	512	10	ELMo + Flair	**0.55**
40k	512	25	ELMo + Flair	**0.51**

Just like in the fastText case, the size of the test set was scaled (and stratified) to be 10% of the training set in all our experiments for a fair comparison across cases. We used 512 hidden layer dimensions and stacked pre-trained embeddings of ELMo + Flair (forward and backward) for all experiments. Training the Flair model with 40k training samples for 25 epochs took around 3 hours on an NVIDIA P100 GPU.

With Flair, we observed that the model significantly underfit the training data (i.e. the validation errors were still reasonably high) even after 25 epochs. Due to hardware and time constraints, we were unable to train for more epochs. Our expectation is that it could take up to 100 epochs to achieve improved F1-scores with Flair that are comparable with those from fastText. In addition, we believe that the larger the number of training samples we are able to fit into GPU memory, the more the model can "learn" from the 5-class training set to provide better fine-grained sentiment predictions on our news dataset.

5.0.3 Qualitative evaluation of all models

We also performed a cross-domain qualitative evaluation of our models by inspecting how well they identified known scandals or controversies. We chose multiple queries across domains (people, organizations, products, nations) and observed that the Flair sentiment model performed the best across all scenarios.

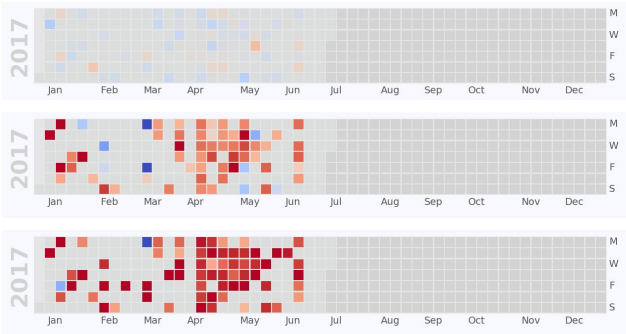

Figure 11: Calendar maps of Textblob (top), fastText (middle) and Flair (bottom) for "United Airlines"

The above plot shows a comparison of the predicted mean sentiment scores (dark red is -1, dark blue is +1) for the query "United Airlines" in 2017. The TextBlob model performed very poorly and failed to identify negative sentiment trends over a broad range of topics, mainly because it was unable to deal with even mild sentiment swings or ambiguous vocabulary within the extracted content. The individual positive and negative per-sentence scores with TextBlob were greatly diluted when aggregated, making this approach unsuitable for our purpose.

In the case of fastText, the broad trends were well captured, but the model had a tendency to "flip" the mean scores (between positive and negative) abruptly between days. We interpret this as the model's inability to disambiguate words that have multiple meanings, or rare, unseen words in the news dataset. This caused wild swings in daily mean sentiment scores,

reducing the model's reliability across a range of scenarios.

In general, we were most satisfied with our Flair model's ability to accurately capture the below scenarios in which there were significant amounts of negative press coverage:

- 2014: Ebola epidemic in Liberia
- 2016: "Lochtegate" controversy with the American swimmer Ryan Lochte
- 2016: Drug pricing controversies with the American CEO Martin Shkreli
- 2016: Samsung Galaxy Note battery fires
- 2017: Airline manhandling and dress code controversies with United Airlines

Appendix A shows some examples of sentiment trends captured by our best model for the above real-world scenarios. By making predictions that correspond with our expectations from reality, we believe that our model does capture real-world phenomena with sufficient accuracy, with room for improvement given adequate training data and resources.

6 Data Product

We envisage our application as a customizable product that scales with larger and larger news datasets. Because data indexing and sentiment prediction are expensive steps, our intention is to deploy the application as a background service (on the end user's servers) that continually scans a news database for sharp changes in sentiment towards a list of targets. The targets themselves are not fixed - they can be customized by the end user to fit their needs.

With regard to usage, our UI/dashboard is designed to allow the end user to explore pre-decided search queries for temporal trends. The time series plots can be interactively probed to dig deeper into metrics such as per-day article counts and most polar (positive or negative) headlines toward a target on a particular day. The UI also provides distribution plots showing which publications wrote the most negative content towards the target, and a similarity distance plot to learn at a glance which publications are similar in terms of content written. We also implemented a dynamic data table that can narrow down on specific time periods of interest per target - the user can update a slider bar to dynamically filter relevant content pertaining to a target within a specific date range.

We believe our targeted news sentiment tracking system is an intuitive tool to help break down the coverage towards entities of interest in a user-friendly and customizable way for numerous real-world scenarios.

7 Lessons Learned

Through our work in this project, we ventured deep into the implementation aspects of data indexing and retrieval, data cleaning, text processing, sentiment analysis and interactive visualization. We learned a great deal about end-to-end product development towards a specific end goal.

During the development of our data indexing and retrieval system, we learned about and applied concepts from parsing, such as context-free grammars, to effectively index queries pertaining to specific keywords. We realized that even curated content from news articles can be *very* messy for NLP purposes, and experimented with numerous text cleaning and sentence segmentation techniques to improve our sentiment predictions. We spent significant time fine-tuning our sentiment analysis pipeline and applied some of the latest developments in NLP, deep learning and language modelling to achieve realistic results.

We also conceptualized novel visualization techniques for viewing sentiment as a time series and brainstormed about how to effectively communicate our findings to the end user in an intuitive manner. By building and deploying an interactive UI as a web service, we believe that we made significant strides in our thought process as data scientists.

8 Summary

In this project, we developed an end-to-end NLP-based application that automatically detects fine-grained sentiment towards a specific target query (such as a person, event, product or organization) in news articles. We applied novel

combinations of techniques from big data, NLP and time series visualization to provide the end user targeted insights into press coverage on a specific entity. Our system was shown to identify large-scale shifts in sentiment in news coverage towards a target reliably, and we foresee numerous commercial applications that could benefit from this approach and help guide the relevant personnel in making data-driven decisions.

References

[1] Godbole, Namrata & Srinivasaiah, Manjunath & Skiena, Steven. (2007). Large-Scale Sentiment Analysis for News and Blogs. ICWSM 2007 - International Conference on Weblogs and Social Media.

[2] Krishnamoorthy, S. Knowl Inf Syst (2018) 56: 373. https://doi.org/10.1007/s10115-017-1134-1

[3] Google Cloud Natural Language API, Interpreting aspect-based sentiment analysis.

[4] Yelp Open Dataset. https://www.yelp.com/dataset

[5] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, *Bag of Tricks for Efficient Text Classification*

[6] *Contextual String Embeddings for Sequence Labeling*. Alan Akbik, Duncan Blythe and Roland Vollgraf. 27th International Conference on Computational Linguistics, COLING 2018.

[7] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. NAACL, 2018.