Positively Newsworthy

Team: CodeCooks

Final Report 21-08-2017

Jaroslaw Diuwe
Edward Hope
Francis Humphreys
Paul McEvoy
Tyson Thangaraj

Part 1: User Scenario

We have built a prototype web-based tool that allows users to filter daily news stories by sentiment and entity, so that they can choose to see more positive and fewer negative stories.

Our target user

Our target audience are users that like to keep up with news by visiting news websites but who are concerned about the quantity of sensationalist negative news, including fake news, and would rather see more balanced or positive news.

Some of our users may have a strong preference for positive news. Some may believe they do, even if they may be more likely to choose negative articles in a given selection (Stafford, 2014; Trussler and Soroka, 2014). Others may just want to control their news diet. Taken together, a significant population may find our app of interest.

The audience for positive news may also be growing due to sharing on social media. There is evidence that 'good news can spread faster and farther than disasters and sob stories' (Tierney, 2013).

Why our target users matter

Well-functioning democracies depend heavily on the subset of the population that follows the news. Those that view more positive news will likely to have a more accurate perception of the general condition of their country and planet than those whose worldview is skewed by dramatic stories of outrage, scandal and disaster. News filtering also has the potential to improve wellbeing (McIntyre and Gibson, 2016).

If we can demonstrate demand for sentiment-filtering, then other news websites may start to implement similar features. If sentiment-filtering for positive news becomes popular, it could have a real impact in democracies on the consumption and production of news.

The problem we solve

We provide users with an app that allows them to keep up with daily news and yet avoid most very negative stories. There is no such solution currently on the market (see Part 2). In addition, we allow users to see selections of positive and negative news on entities of interest that they select.

Like evolutionary adaptations that may contribute to obesity in the modern world (Johnson, 2010), we may have an evolved predisposition to be alert to new threats in our immediate environment that has become dysfunctional in a world of globalised media. Citizens can struggle to prioritise risks as a result. Author Rolf Dobelli compares news to sugar and suggests that "consumption of news is irrelevant to you" (Dobelli, 2013).
Negative news can affect users indirectly by harming democracy and society (see part 2). It can also directly affect their mental health. Writer and cultural critic Lee Siegel argues that political and social

changes "have given the media's habitual negativity unprecedented political consequences." You can be convinced "of impending disaster and all-encompassing danger" derived "solely from what you read, watch, and listen to" (Siegel, 2017). According to British psychologist Dr. Graham Davey, "Viewing negative news means that you're likely to see your own personal worries as more threatening and severe, and …you're more likely to find your worry difficult to control and more distressing than it would normally be" (Gregoire, 2015). The effect may be worse for women. One experiment showed that women remembered negative news better and experienced elevated cortisol for longer (Marin, *et al.*, 2012).

Daily news readers want to be informed. Users of our app will likely perceive the state of the world and their nation more accurately, and be more skeptical of extreme negative stories that they encounter elsewhere, whether real or fake.

Fig 1. Negative news example

Is there demand for good news?

Do users want to filter the news? One of the challenges in establishing demand from user surveys is that the link between preferences and behaviour may be weak (Trussler and Soroka, 2014). However, a market for good news does exist. The Daily Good website claims over 100,000 subscribers and Positive

News has a print circulation of 40,000 (McIntyre and Gibson, 2016). GoodNewsNetwork has over 570,000 Facebook 'likes' and almost as many followers, perhaps helped by its sharing of stories from more popular sites like BoredPanda and LittleThings, which have over 10 million likes each. HuffingtonPost and Inkl have good news sections. However, in other organisations, good news sections can be low-profile. For example, the Fox News good news category has only a handful of stories and does not appear to be reachable from the menus.

Those offerings all suffer from the limitation that they don't allow users to keep up with the main news stories, which may limit their market. The total news market is very large. Including advertising revenue, the global print and broadcast news market is an estimated €1 billion. In Ireland, online news media generate about €80 million in revenue. Since most news reports aim to attract an audience, we would expect a significant demand for the most positive 20-50% of news stories. We cannot infer that users would filter the news (i.e., choose such a subset), given the option. However, in our evaluation (see part 4) we found evidence they would strongly consider it.

Part 2: Technical Problem

The motivation for our system

Our system is targeted at users that want to stay informed. Taken as a whole, negative-biased news is misinformation, making it difficult to argue that it is meeting user needs. Media represent the world as being in constant crisis, even though it has improved beyond recognition since World War II (e.g., Roser, 2017).

Democracy has been found to have a long term positive effect on human well-being (Bellinger, 2017) and on both life expectancy and health policy, even after controlling for income (Besley and Kudamatsu, 2016). An advantage of democracies is that the government can be openly criticised. However, excessively negative media coverage of domestic news and politics undermines public support for democracy and blurs the distinction between democratic countries and non-democracies, to the extent that the US, the UK and France have elected or flirted with potential leaders for whom democracy is not a top priority (Hoefer, 2017; Bhatia, 2017; BBC, 2017a; Polyakova, 2016; Croucher, 2015; Bloodworth, 2015; Reuters 2017). Audience and media bias towards negative news can facilitate internal and external enemies of democracy, including violent extremists. It has helped Al Qaeda and ISIS both to recruit and to spread terror.

The bias towards negative news also primes us to believe and share outrageous fake news. Fake news has recently been identified as part of a wider threat to democracy called computational propaganda (Woolley and Howard, 2017). In Michigan, junk news was shared as widely as professional news in the run up to the 2016 US presidential election (Howard, Bolsover, Kollanyi, Bradshaw, & Neudert, 2017). It may even have altered the result (Dewey, 2016; Parkinson, 2016). Much of that fake news was sensationalist negative fiction about one of the candidates (e.g., Pizzagate). Perhaps people would be less credulous if they were not so conditioned to a news diet of outrage and disaster. If filtering could help people to see more positive news, fake news might be easier to spot and filter out.

Finally, as mentioned in part 1, overly dramatic and negative news can increase fear, anxiety, distrust and pessimism, impacting mental health: it has even been linked to PTSD (Gregoire, 2015).

Hence, we have developed a prototype system that allows users to see news with less of a negative bias. Our system assigns a score to each article according to the degree to which its overall sentiment is calculated to be positive or negative. Users can choose to filter news articles based on sentiment, so that they do not see many of the most negative articles.

We have also obtained scores from IBM's Watson for each article for different emotional tones (sadness, joy, fear, disgust and anger), which we have incorporated into one of our filter models, discussed later. Watson also assigns entity types and subtypes to articles according to how significant they are within it (IBM, 2017). We use these to help users find and filter content by both sentiment and entity.

The core technical problems

There are some key technical problems that we would face when building our proposed system and in this section, we will look at some of the problems and potential solutions to overcome those challenges.

Data Collection

To filter positive and negative news articles first we need to collect them from the news sources. Fetching news articles from multiple news sources might be challenging - this could involve custom code to pull news articles from each source individually using each news source's API. If individual code is needed it might restrict the number of sources we can use, considering the time available and the other technical tasks that need to be done

Data Extraction

Even if we manage to get text from news sources, not only will it have formatting specific to that source but it might also have images embedded in the text, advertisement text, cookies, and links to other stories. Will we be able to get the text from the HTML directly and if so could we recognise the features above that do not relate to the article content? Should we consider the minimum size of an article, the maximum size? What if an article is updated at a later date, do we scrape it again and delete the earlier version?

The difficulty of sentiment

A fundamental problem with making accurate sentiment predictions on articles is that news articles do not have an objective sentiment independent of the user. Interpretation of an news story can vary widely depending on a user's location, income, opinions, beliefs and prior knowledge. It was not surprising for us to find considerable disagreement between sentiment tools when analysing the same articles.

There is also a difficulty about giving an aggregate score to text that might contain a combination of positive and negative content. Should the negative content be given extra weight? That depends on whether users want to avoid negative news more than they want to see positive news. What classification errors are most important to the user? Should we filter out all news that isn't clearly positive. If so, do we risk filtering out too much news to be useful?

Another problem is that individuals might judge the sentiment of articles in different ways. Some might be strongly influenced by the headline.Should the headline be given more weight? Should the 1st half be treated differently to the 2nd?

Article presentation

Another challenge we had to consider was once we had the data how could we present it in a coherent and usable way to the user. Some aspects were:

- Do we link directly to a news article or embed it in some way
- Should display images or videos?
- How do we control the varying sentiment; a drop-down menu, a slider?
- How do present the news; by published date, by highest sentiment?

- How do we handle multiple instances of entities?

Sentiment tools and data quality

In order to decide whether articles were positive or negative, rather than build a sentiment analyser from scratch, we needed to find tools suitable to the task, consider which ones to use, how to interface with them and their cost. It was also crucial to know how accurate they were at analysing news articles.

Predicting a new sentiment score

We set ourselves the challenge of dealing with some of the difficulties with sentiment analysis identified above by using more data to build a better model. Although news articles do not have an objective sentiment, if we can identify articles on which there is some consensus, then we can use these as a ground truth corpus with which to build a model. (We designed an experiment to do that, see part 4).

Another way to achieve consensus is to combine multiple tools – Aylien and NLTK as well as Watson – and scores into a single model as this has worked well in other machine learning contexts. We obtained sentiment scores for headlines and also for the 1st and 2nd halves of articles. We could also include Watson emotions in our model. We learned from contact with IBM developers that Watson's emotion scores do not contribute to its sentiment score. The remaining problem was: **how to create a new model that used some or all of this data effectively**.

Existing products that address the problem of negative news

There are 3 current approaches to providing balanced or positive news. None solve the problem for our target users.

Neutral reporting

One approach to providing balanced news is neutral reporting that tries not to take sides employed, for example, by the BBC, RTÉ and PBS. Neutrality is, of course, impossible but attempting to be neutral may increase public trust. In a recent US survey, 4 of the top 6 most trusted media outlets were public broadcasters (Kearney, 2017).

Public broadcasters tend to be less sensationalist, which helps them to avoid negative extremes. However, they still exhibit the negative bias of other media in their domestic and international reporting. The BBC responded to an accusation by President Trump that terrorist attacks in Europe are under-reported by pointing out a long list of attacks that it had covered (BBC, 2017b). It is not terror that is under-reported but progress in global public health that may affect thousands or millions.

Good news publications

Some have tried to counter negativity with 'good news' publications. The association of Irish Non-Governmental Development Organisations, Dóchas, produces 'The World's Best News' to try to set the record straight. However, there seems to be a limited audience for packaged good news. Although there is also a website (worldsbestnews.org), The World's Best News is only printed once a year and is given away for free. Its focus is on global development news, particularly relating to UN development

goals. There are other websites such as Positive News and religious websites that specialise in positive news but they are either narrow in focus, infrequently updated or both. None of these, therefore, solves the problem for users who want to keep up with the day's news without seeing so much negative news.

Good news sections

The third approach is to provide a special section on positive news within a regular news service, such as those provided by the HuffPost and Inkl. However, these positive sections are infrequently updated and are a poor source of daily news and are separated from the top stories. Therefore, they are only ever going to reach small audiences intermittently. Again, they are not solving the problem for users who want to browse the day's main news without seeing so many negative stories.

Part of the reason that HuffPost and Inkl fail to provide daily positive news may be that their positive sections appear curated. They may not have a sufficiently large audience to warrant the cost of manually curating daily news. Even were they to do so, they would have a limited selection of news articles to choose from because they only select news that is clearly positive. Our alternative approach solves both these problems and the problem for our users, because the selection of positive news is automated and, instead of giving users a binary choice between "normal" and positive news, we provide users with the choice of how much of the negative news they wish to filter out of the daily news.

There is a gap in the market for a product that allows users to view a more positive selection of daily news stories.

Part 3: Technical Solution

Our system is designed to carry out the following steps:

- Extract URLs on the latest news from *newsapi.org*
- Scrape the news article text
- Extract the sentiment using Watson and Aylien APIs
- Post process the data
- Evaluate a new model score

Backend

Our Backend a number of technologies, these are a few of the key ones:

PostgreSQL

We decided to use PostgreSQL for our backend data storage solution due to the small footprint it would have in the ecosystem of our project. Also, a number of our team were familiar with it, and confident in applying it. There is also significant support within the django community. PostgreSQL is the backend of choice for Django projects. When determining the most efficient structure for our database tables we applied normalization to the degree of second normal form (2NF). We decided not to normalise further at this stage, as it might slow the development of the prototype.

GOLANG

Golang (Go) is a modern, open source programming language developed by Google. We considered using Python and Go. Both languages are quite easy to learn, have great documentation and third party libraries that would be required for our application. We decided to use Go for a few reasons.

First, Go code compiles into a **single binary**. This is very important as we can easily package an application into a single file and deploy it on any server without having to worry about satisfying dependencies. This is an area where Python falls short as it requires not only all dependencies to be installed on every server where we run and test our code, but also to have the same version of Python installed. Use of virtual environments can help address some but not all of these problems.

Second, Go offers **excellent performance**, especially when using *goroutines* (lightweight threads) for concurrent processes. We use concurrency to process articles, this saves time as we don't need to process articles in serial. Another reason we chose to use Go is because it doesn't require any Web development framework, it has a Web server and a very powerful *http* and *json* packages included in its standard library. Finally, we really wanted to challenge ourselves and learn a new, modern programming language that is quickly gaining popularity as we believe that knowledge of Go can **benefit our careers** as software developers.

Docker

All Golang applications are compiled to a binary file and deployed in production using Docker containers. Using Docker **helps us manage changes** to our code base, and speeds up the process of application deployment in the production environment. Configuration of each Docker image is defined in a Dockerfile included in the GIT repository of each application.

Docker images include all the files necessary to run both applications with the exceptions of configuration files. The JSON configuration files reside on the virtual machine that hosts our Docker images. When deploying Docker containers from those images we specify a shared volume, a local directory on a virtual machine that is mounted in read only mode in each Docker container. We do that for these reasons:

- JSON configuration files contain a lot of sensitive data such as database username and password.
- We use a single JSON file with text extraction rules for each source. This list is updated over time and by being able to edit it at any time we no longer need to rebuild a Go binary and update containers.

How the Backend fits together

There are two core backend components that are designed to reliably find, download and analyse news articles - **codecooks-newsapi** and **codecooks-watsonapi.** They are part of the backend system and were written entirely in Golang.

codecooks-newsapi

What it is:
Codecooks-newsapi is our tool for getting the latest news articles across a number of news sources. It is not designed to get the article content but instead supply key information to WatsonAPI to extract the text.

What it does:
Codecooks-newsapi application finds information about the current top stories published and stores this information in the database. The application reads a JSON file for a list of news sources. For each news source, a list of latest articles is downloaded from newsapi.org and parsed into a Go structure.

Why we need it:
As identified in Part 2, we need a universal mechanism to source news articles. Our team considered using APIs of several independent news publishers such as *The New York Times* and *Reuters*. However, we decided that using the newsapi.org API, with access to news from over 70 sources in a unified format, is more than sufficient for our purposes and allows us to complete this part of the project much faster.

codecooks-Watson

What it is:
Codecooks-watsonapi application is designed to find recently added articles in the database and call two public APIs that we use for sentiment analysis: IBM Watson and Aylien.

What it does:
WatsonAPI fetches the article text, cleans it, sends the text to Watson and Aylien and returns sentiment data to be stored in the database

What why need it:
WatsonAPI is a key part of the design as the conversion of article text to sentiment values can then be used in model predictions. As identified in Part 2, it is fundamental we have clean text to ensure accurate sentiment values can be obtained.

codecooks-restapi

A third Docker image was built for the purpose of making our data more accessible to others. Codecooks-restapi is our REST API application, also written in Go. It is similar to newsapi.org: it provides API key holders with a list of news sources and articles. Information such as the URL, title, author, date of publication, source and description of articles can be requested by making a simple GET request. In addition to this, we include all sentiment, entity and emotion analysis for each article. This additional information can be quite valuable to bloggers or anyone searching for articles about a particular person or articles having more positive sentiment. More information is available in the *appendices* section.

User starts application in CLI mode (allows Docker container as a daemon)

For each news source a list of current top stories is downloaded from newsapi.org and parsed into a structure.

Extra fields are added to each article, these include:

a. uniqueid – unsigned integer which is a hash of article's URL
b. source – information about news source
c. text – text of article is extracted using *goquery* library

Each article is downloaded using HTTP client GET method.

Application checks if an article with the same uniqueid already exists in the database, if yes, this article will not be processed further.

Save the article in the database
Save information as JSON file in local file system

Fig 2: NewsAPI

Application parses flag parameters.

Application reads JSON configuration file, in case of any errors application exits.

Application verifies that a connection to a database server can be established, if unsuccessful application exits.

Application queries the database to get a list of articles that need to be analysed.

For each article, we make a HTTP request to Watson API requesting text analysis

Fig 3: WatsonAPI

Backend Challenges

Text cleaning

As identified in part 2, one of the challenges for our team was ensuring the text that we send to Watson for sentiment analysis was as clean as possible. In early development, we used the URL of each article (provided by *newsapi.org*) in a CURL request that was sent to Watson. This enabled us to collect results of thousands of articles in a few days. We continued to process the backlog of articles but realised the results of sentiment and emotion analysis were not very accurate. It was identified that the HTML documents, included extra text that could negatively affect analysis. This extra text included java script, HTML tags, cookies, links to other articles, etc.

Our first solution to this problem was to use *goquery* library to extract text that is enclosed within * * tags. This gave a marginal improvement. At around the same time we decided we wanted to actually display the text of articles in our website, instead of displaying the original article in an *iFrame*. It was clear we needed a more robust solution for text extraction. We considered using Python's *Newspaper* library. However, as our text analysis application was written in Go, we didn't want to create an additional layer of complexity. We established that the *goquery* library offered equivalent functionality. Our tests show that it performed as good as *Newspaper,* and for some news sources, slightly better.

Efficient API use (bottlenecks)

As identified in Part 2, we needed to consider the best use of the free sentiment tools available. During the first week of testing our codecooks-watsonapi application, we learned that over 30% of articles, from various sources exceeded 10,000 characters - this is the limit of free NLP (natural language processing) units for one unit using Watson API. To increase the number of articles that we could analyse without a premium account, we reduced the number of characters that we pass in HTTP requests to 9999 characters. We also removed articles that had less than 1800 characters, as these often contained mostly multimedia, which we did not have sentiment tools for. We also removed new line characters to make text more compact. The other limit related to Aylien. With the free API we were limited to 1000 requests a day and only 1 per second. By adapting our code to wait between requests and switching various API keys from multiple accounts we were able to get what we needed.

Front-end

CodeCooks-Django serves as the frontend for our web application, it is used as a bridge to connect our frontend news application to the backend postgres database. Our frontend website can be accessed via https://codecooks.me

Our frontend uses the following technologies:

Apache

Apache is web server software that is used to deliver our web application to end users. Apache is also straightforward to integrate with any python web application like Django using Apache's WSGI module. Our web session is encrypted using SSL provided by *Let's Encrypt* - a free solution provided by Electronic Frontier Foundation.

Django

Django is a web framework written in python that follows the Model-View-Template architecture pattern. We decided to use Django rather than other web frameworks as it provided us a quick and easy way to setup our web application in a very short period of time. It also provides us with built-in tools to mitigate common web attacks like cross-site scripting, SQL injection etc. Django also **works very well with PostgreSQL** - our database.

Bootstrap

Bootstrap is a HTML, CSS and JavaScript framework, we decided to use it as it provided us an easy and **simple way to develop our web application** to run on multiple devices, like smartphone and tablets with ease. By using bootstrap, we were also able to quickly build our frontend application without the need for writing any HTML, CSS or JavaScript code manually. We used **Jinja2** for our html template, this also helped us to rapidly deploy our application with minimal effort. We also used the **Seiyria JQuery** package for some of our UI elements like the slider.

How the Frontend fits together

1. User sends a GET request to our web server of a web page they wish to view.
2. Apache forwards the request to the URL dispatcher
3. Based on the URL in the GET request, the request is forwarded to the appropriate django view
4. The view then processes the request, this includes querying the DB to retrieve any data using ORM
5. Additional application logic is done at this level
6. The view then passes the processed data to the template, which will then be used to render it on the end user's browser
7. In order to speed up the load time for JavaScript, Jquery, CSS and Article images, we use content delivery networks to reduce bottlenecks from network latency issues on our server
8. The request is sent to the user to be rendered by their browser

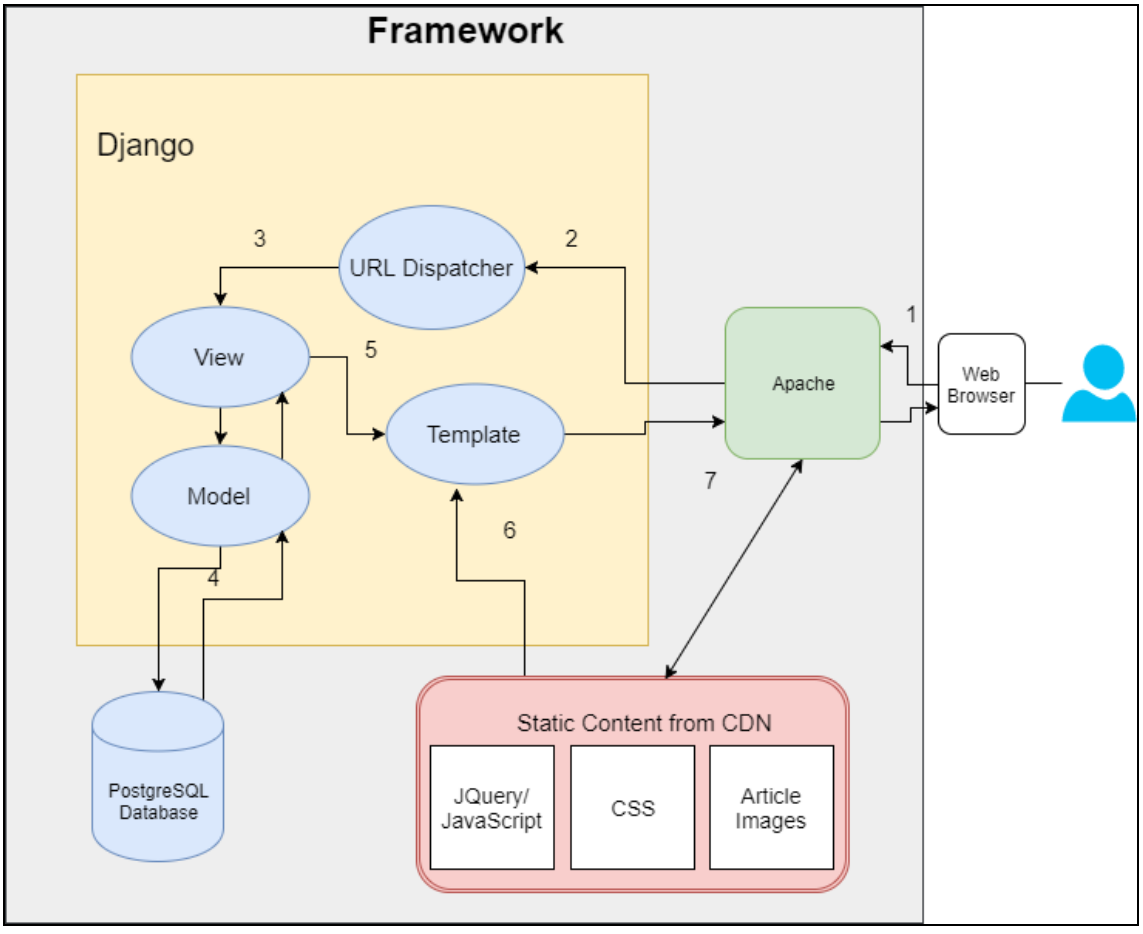

Fig 4: Django Frontend Diagram

Frontend Challenges

During the development of our frontend application we had multiple challenges that we had to overcome. As identified in Part 2 we needed to ensure users could **filter articles based on sentiment**. Initially we had a drop-down box that allowed users to select if they wanted to view all negative, neutral or positive articles. We found that this approach was too restrictive for end-users. We then tried a “range slider”, this allowed users to set the maximum and minimum value of the model score that they wish to filter the articles by, however users found this method too cumbersome. We settled on a single slider which filtered articles according to whether or not their sentiment score was above or below the threshold set by the slider. This worked quite well and was simple to understand for the user.

In part 2, we also discussed the question of linking to an article or presenting the text. We first used an **iframe** to display articles, but some news sites were blocking third party sites (like ours) from displaying articles within iframes. We also received feedback from users that using an iframe to display article text was not very readable, we decided to display the scraped text using the *<pre>* tag instead. As outlined in the *Backend Challenges* section of Part 3 this still required significant text cleaning to make it readable.

Data sources and database structure

Our project relied on a specific set of data: mainly the article text, sentiment and associated features, and then data that we obtained using these – the model scores and user ratings.

News article text

We retrieved articles from the following news sources:

Aljazeera	**Reuters**	**NewsWeek**	**BBC News**
Washington Post	**Economist**	**The Guardian**	**Time**
The Independent	**Bloomberg**	**Huffington Post**	**The Telegraph**
The New York Times			

Table 1: Article data sources

Sentiment Scores

Sentiment scores form the basis for our models. Watson and NLTK provide sentiment values between -1 and +1. Aylien provides polarity; positive, negative and neutral with a *confidence* on that polarity. Each of these are stored in the database and linked to the article.

Emotions and Entities

Emotions: Joy, Anger, Disgust, Sadness, Fear
Entities: People, companies, organizations, cities, geographic features

User ratings

These are the ratings that users provided via the web interface. Each rating is linked to the article and multiple ratings per article from unique users are maintained

Model Scores

The model scores are the predictions made on an article based on all of the sentiment information. The model is also trained on the ground truth given in the form of user ratings

All of this data is stored in our PostreSQL database with a carefully considered table structure to ensure it satisfied the needs of post-processing and article display on our frontend. (Key data is marked **red** below).

Database structure

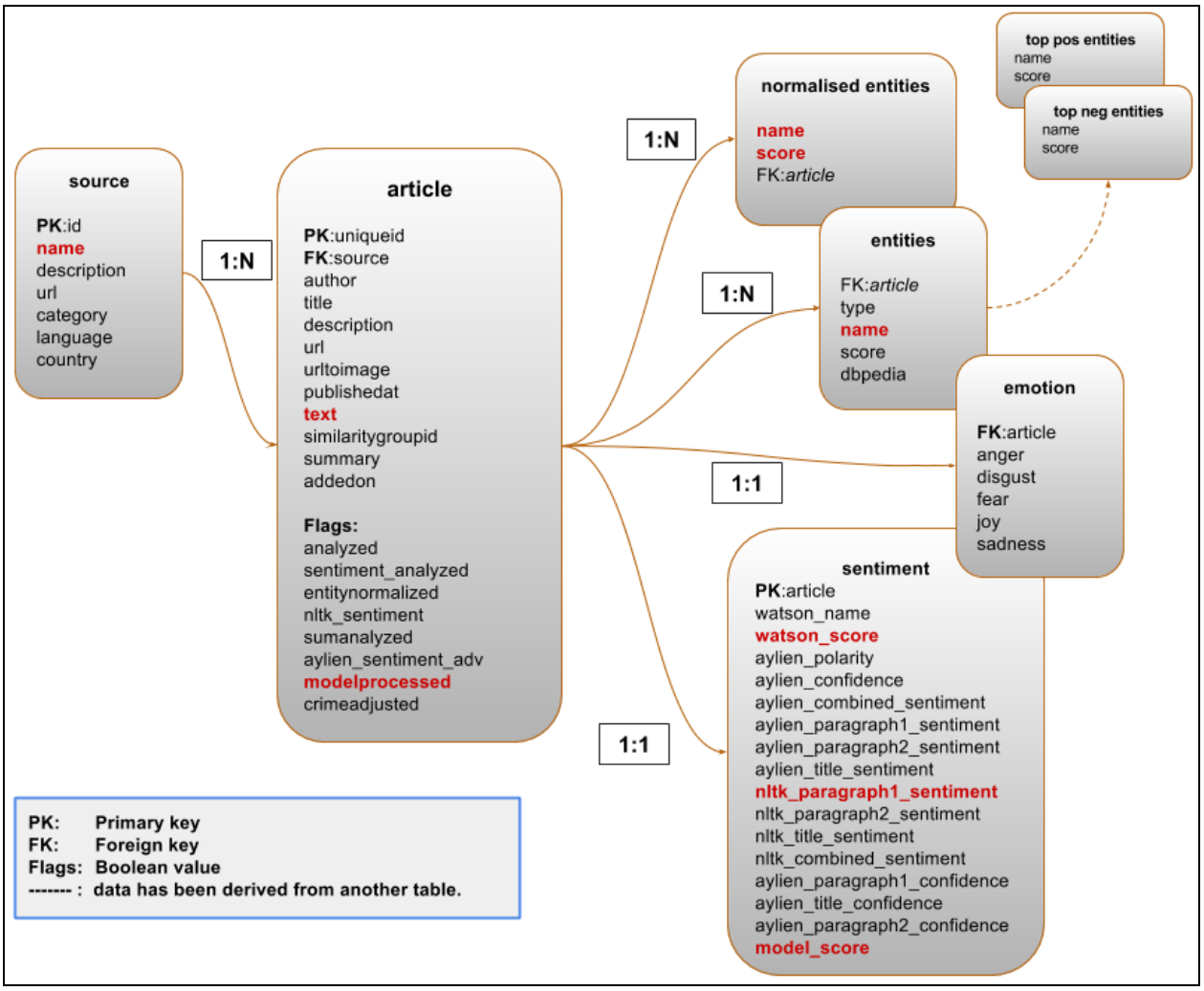

Fig 5: Database structure

Our database consists of six main tables used to store data which we either retrieve from external sources (*newsapi & watsonap*i) or results from our data analysis. We also created several more tables for the purpose of post-processing and also our user evaluations, which contain a subset of pre-existing data.

Post-processing

Once articles have been scraped a number of post-processing steps are carried out. As identified in part 2 we needed to ensure articles were presented cleanly, without repeated stories and an accurate sentiment score. Some to main tasks to achieve that are highlighted below.

Entity Normalisation

Problem: Multiple variations of entities made it difficult to track entities effectively
Solution: Create a dictionary of common entity variations and replace where necessary

Watson would occasionally respond with variations of the same entity (often within the same article) eg:

- Donald Trump
- President Trump
- President of the United States

This would cause difficulty when finding related articles or judging the main entity in an article.

A JSON dictionary was created that recognised the 'key' for a range of variations of the same entity.

```
"Trump": ["Donald Trump", "President Trump", "President Donald Trump",
"Donald Trump", "The Donald"],

"USA": ["US", "U.S.", "U.S.A", "United States", "United States of America"],

"UK": ["United Kingdom", "U.K.", "Britain", "Great Britain", "GB", "G.B."],
```

The normalisation script collected the entities from a single article and checked them with the json dictionary. If a match was found then the entity was replaced with the key. It was then necessary to remove duplicates. However, each entity had a 'score' which indicated the strength of that entity within the article, ranging between 0 and 1. When replacing two entities with scores x and y with a single entity representing both, the following adjustment is made to the score:

$$s(x,y) = 1 - (1-x) * (1-y)$$

This formula ensured that when any 2 entities with scores in the interval (0,1) were combined, the single entity representing both would have a higher score in the same range:

s(x,y) = max (x,y) if min (x,y) =0 or max (x,y) = 1
s(x,y) > max (x,y) otherwise.

The new entities and scores are sent back to *backend_entitiesnormalized* table in the database and the *entitynormalized* flag is set.

NLTK Sentiment & Aylien Sentiment

Problem: We want richer NLTK and Aylien information on text sentiment
Solution: Split text into sections and use an NLTK tool (Vader) and Aylien API to extract sentiment

NLTK sentiment analysis uses the *vader* (nltk.sentiment.vader) tool provided with NLTK. This tool works well for larger text blocks broken into sentences. It relies on locally stored sentiment lexicon that considers

sentiment intensity and context and provides a score between -1 and +1 (negative to positive) for a piece of text. To use this on an article the text is broken into sentences and split into an upper and lower section. The upper and lower sections are fed one sentence at a time to the vader analyser and sentiment result is averaged for all sentences in the respective section. The title is also analysed and all scores for this article are sent to the database to table *backend_sentiment* and the *nltk_sentiment* flag is set. A similar process was carried out using the Aylien API.

Article Similarity

Problem: Articles were displayed by date and similar stories often appeared together
Solution: Do title analysis on the articles and separate common stories

When displaying articles it is important to separate similar stories. To do this some similarity comparison is made by extracting the entities from the titles from every article on a single day. A list of all entities is compiled and ordered by frequency. The top ten entities are assigned an ID value corresponding to their position. Each story is then checked for the presence of that entity in their title, if true the article is assigned that “group” ID. An offset is attached to each ID corresponding to the day which ensures IDs are unique. Keeping the group IDs completely unique across all dates means it is completely agnostic to the day they are from, but know that articles with the same ID are similar and need to be kept apart. We considered (and tried) doing some clustering on the articles based on the text or entities. However this clustering was overkill, we weren't looking for really robust article similarity checking but rather a way to keep very similar articles apart.

Summarisation

Problem: Users might want more information on article content before clicking through
Solution: Use a python package to extract a “summary” and provide via tooltip

The summarisation task uses the Python package 'sumy' to extract the top 5 sentences that represent a summary of the article. These are then used as a tooltip when the article is displayed on the site. Summarisation is only done on articles that have not been summarised by checking the *'sumanalyzed'* flag. This is to reduce runtime as the summarisation can take a few seconds for each article.

Top Entities

Problem: How do we track the most positive and negative entities throughout the day
Solution: Extract all the entities from the last 24 hours and find most positive and negative

All entities from the last 24 hours are extracted and scores summed to give a figure for their overall proportion in the sample. The scores for each story they appear in are summed and the two figures are multiplied. The score at the end gives an indication of their strength and polarity.

Update model scores

See CodeCooks-ML readme (part 8) for more information on model

Scoring article by sentiment

We tried to see if we could deal with some of the difficulties with sentiment analysis identified in part 2 by using more data to build a better model. We implemented 2 models.

Model 1. "Weighted mean"

Here, we took a simple approach: 2 tools (Watson and NLTK) were combined into a single score. The model used an insight obtained from looking at ground truth data that Watson seemed more accurate at scoring negative articles and NLTK at scoring positive articles. If both tools scored negative, Watson was used. If both were positive, NLTK was used. Otherwise, the average was taken. This combined score was weighted by 0.7 and added to the NLTK headline sentiment score, weighted by 0.3. We tested this approach in the 2nd experiment discussed in part 4.

Model 2: Linear regression

A linear regression approach is more general and more flexible than a weighted mean: with new data, we can change both the independent variables and parameters. Since users can rate articles using our prototype, we built a linear regression model that could be updated automatically as we obtained new ground truth ratings.

For the model, we used a large range of potential regressors:

- Watson sentiment and 5 emotions
- NLTK and Aylien scores for whole articles, headlines, and article 1st and 2nd halves.

Bearing in mind the possibility that tools may vary in how accurately they detect positive or negative articles, we split each sentiment tool in 2 parts, one positive and one negative, so that each could be weighted differently.

Given our use of multiple sentiment tools analysing the same text, we would expect some multicollinearity – a linear relationship between multiple variables. Some correlation was clearly visible from scatter plots between Watson and NLTK scores. We managed this by using orthogonal parameters, which helps avoid collinearity and makes for a more robust model than would using principal component analysis, no matter how many regressors there are.

Doornik and Hendry (1994, p244) recommend building a model with **orthogonal** parameters which is **parsimonious**. Therefore, we needed a parsimonious way of choosing between competing models with different numbers of variables. **R-squared** will always go up if new variables are included, therefore we added code to calculate the adjusted R-squared. Before we accepted a model with more variables, we demanded that the adjusted R-squared increased by at least 1% per additional aggressor. In order to automate the process, we created an algorithm to take the input data, split variables into positive and negative, make them orthogonal and evaluate all possible combinations. The code includes an adjustable limit to the maximum number of independent variables to include in the model. Based on the available ground truth data, we obtained a linear model to which NLTK whole article sentiment (negative and positive) contributed most, followed by NLTK for the 2nd half of an article, Watson sadness, and the NLTK headline sentiment (see appendix). We tested this model in experiment 1 (see part 4).

Backend and Frontend together

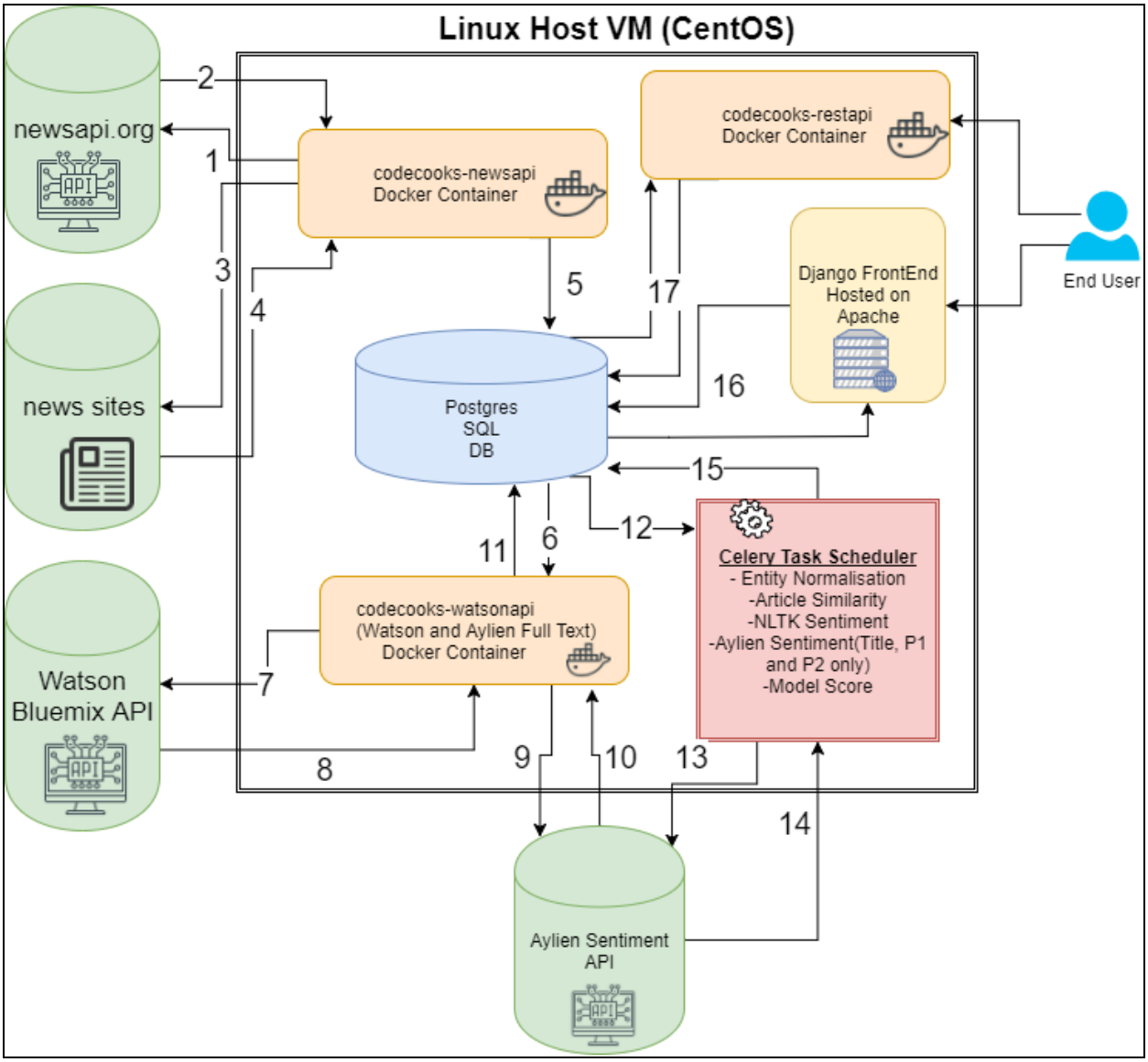

Fig 6: The System Diagram

System Diagram - the steps

1. **codecooks-newsapi** Docker container fetches details about news articles every 2 hours from specific news sources defined in the sources.json file.
2. newsapi.org provides URL about news articles, plus few other attributes likes sources, authors, article published date etc.
3. Using details received from newsapi.org, **codecooks-newsapi** scrapes text from each news article.
4. Text is cleaned - non-article content is removed
5. Clean text is stored in DB and flag is set indicating this article is ready to be analysed.
6. **codecooks-watsonapi** Docker container, extracts un-analysed articles
7. Call is made to Watson Bluemix API and cleaned text is sent for sentiment analysis.
8. Data is received back from Watson Bluemix API (data received includes article sentiment, emotion scores and entities)
9. Call is then made to Aylien sentiment and full cleaned text is sent for sentiment analysis
10. Sentiment data received back from Aylien sentiment analyser
11. Both Watson and Aylien results are written to the DB and the first analysed flag (Boolean flag) is set to True for advanced sentiment
12. Post process scripts start (see appendix for more detail)
13. Aylien sentiment data requested on title and article sections
14. Sentiment data received
15. Sentiment data sent to DB. New model score applied to article
16. Flag set to indicate article is ready for display
17. RESTful API available on demand

The web interface (the product)

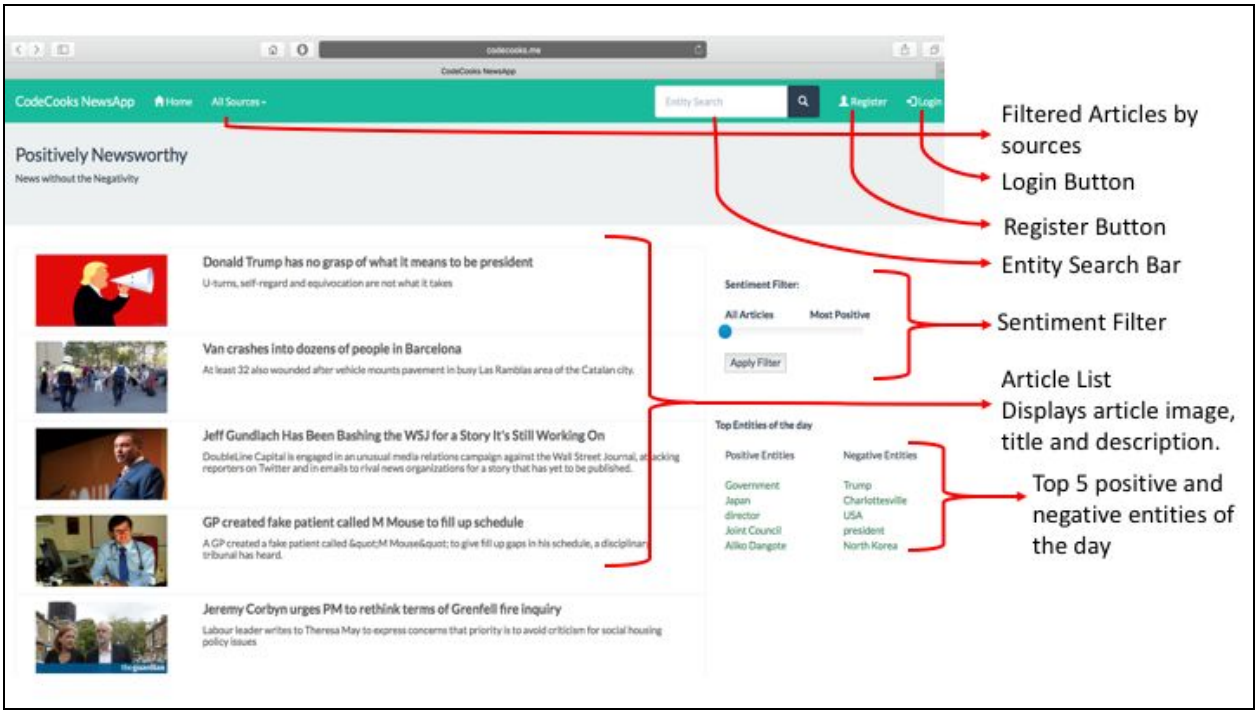

Fig 7: The Front page

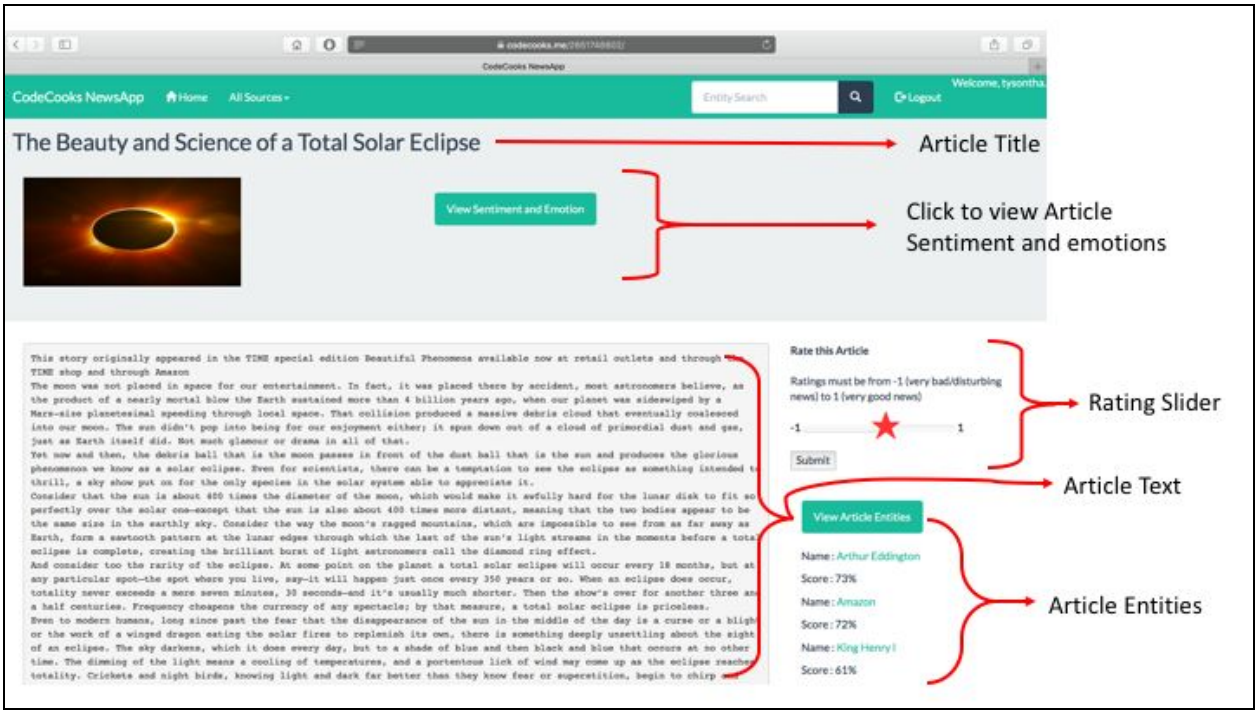

Fig 8: The Article page

Part 4: Evaluation

Evaluation goals

The critical information we sought to discover was:

> "*Do users perceive news filtering to be a real problem that they want to solve?*"
> "*Do we have a viable solution and, if so, how should we implement our solution?*"

We also wanted to find out if our prototype worked as expected and whether or not it would be of value to users. We constructed 6 hypotheses and 2 experiments in order to obtain information on all these areas, to test our filter models and to help build our solution. By testing the news filtering and models, we were also testing a large number of other features on which the experiments were dependent, which amounted to most of the prototype system:

- sourcing news articles via NewsAPI;
- stripping article text;
- scoring articles by sentiment, using multiple sentiment tools, and by emotion, using Watson;
- allowing users to rate articles;
- storage and extraction of articles and scores in the database;
- display of individual stories;
- the news filter slider;
- displaying article collections filtered by sentiment.

We wanted to know what users thought of our whole system. If we asked them directly, they might have told us what we wanted to hear. Instead, we asked users questions relating to whether or not they would find filtering useful on other news websites, since we expected their response to be related to their experience with our prototype.

Hypotheses and experiments

Selected Subjects

Our target users are general online news readers that want to see more positive news. Therefore, we used a selection of adult consumers of online news for our experiment. Almost all were native English speakers and all were familiar with Western media output. They consisted of friends, family and colleagues working in a variety of fields. The group had more males and more computer scientists than the general population but were otherwise likely to be representative in terms of their interest in following current affairs and news in general, and desire to reduce their exposure to overly negative news content. Our target users want to keep up with the news online and want more positive news.

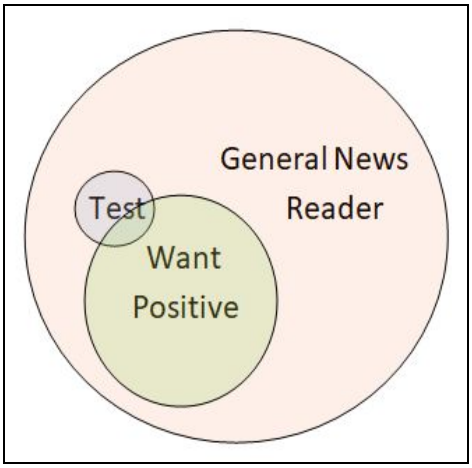

Fig 9: Venn Diagram

Experimental setting

We conducted two online experiments. Running the experiments online allowed us to test a larger number of users more easily than by doing it in a controlled setting. A disadvantage of not running it in a room was that we were unable to control the test environment, such as its duration. However, our experiments were more representative of real-world usage of our app. Google analytics could also help us to monitor our users behaviour to compensate for the fact that users were not directly observed.

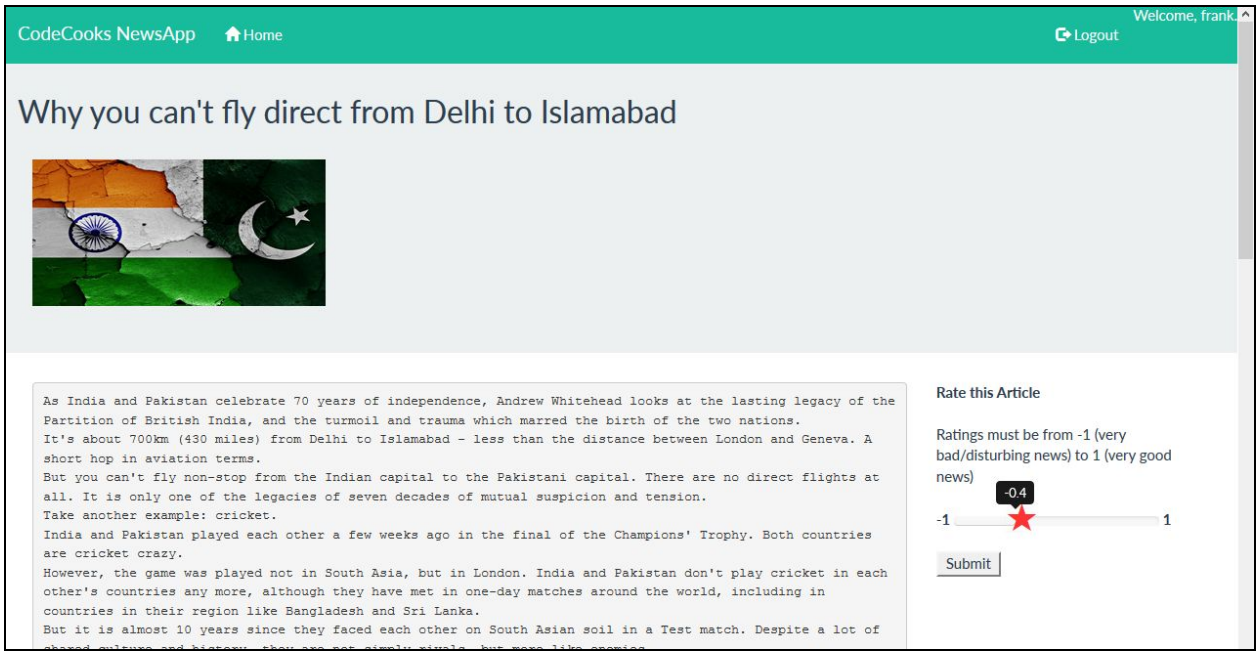

Fig 10: Experiment 1

Experiment 1

We designed our 1st experiment to obtain more ground truth data to:

- build a (linear regression) filtering model and
- test the model and other sentiment tools against the ground truth.

Our team had previously obtained some ‘ground truth’ data by rating 34 articles by sentiment ourselves and with 2 friends, and then taking the mean rating. We also participated in this experiment to obtain more data for test and training, along with friends, colleagues and relations, making a total of 23 users.

Users were given instructions to register, login and rate individual articles on a negative to positive scale of -1 (very bad/disturbing news) to 1 (very good news), with increments of 0.1.

The experiment was set up to be as simple as possible, so we stripped away many of the features of the prototype and hid the sources of the articles, in order to eliminate any bias that might cause.

We randomly selected a set of 200 articles. Each user was presented with a random selection from the 200. Any article with 5 ratings would no longer be displayed. We obtained 498 ratings in total.

Next, we cleaned the data. For each article, we counted its ratings and calculated its mean rating and standard deviation. The mean was used as the ground truth. Any article with fewer than 2 ratings was discarded, as were articles with significant disagreement (a standard deviation > 0.4). The remaining articles were combined with the previously obtained 34 ground truth articles to make a total of 123 articles. These were then split into 2 sets: 82 articles for training the linear regression model and 41 articles to test the model against ground truth in comparison with alternative sentiment tools.

3 hypotheses we used the data to test were that relative to ground truth user ratings:

1 Watson sentiment can identify article sentiment better than random.

2a The linear regression model can identify article sentiment better than random.

3a The linear regression model can identify article sentiment better than Watson.

Experiment 1 results

We tested these hypotheses by taking the test data and comparing the absolute difference between ratings by the sentiment tools and the ground truth ratings using a 2 sided t-test, with the null hypothesis being that there was no difference between different tools or between the tools and random.

The expected mean absolute error (MAE) for a random filter's scores should lie somewhere between 1 and 0.5, depending on how close the ground truth scores were to the extremes or to 0. We generated some random scores and found an MAE of 0.52 (and **RMSE of 0.63**).

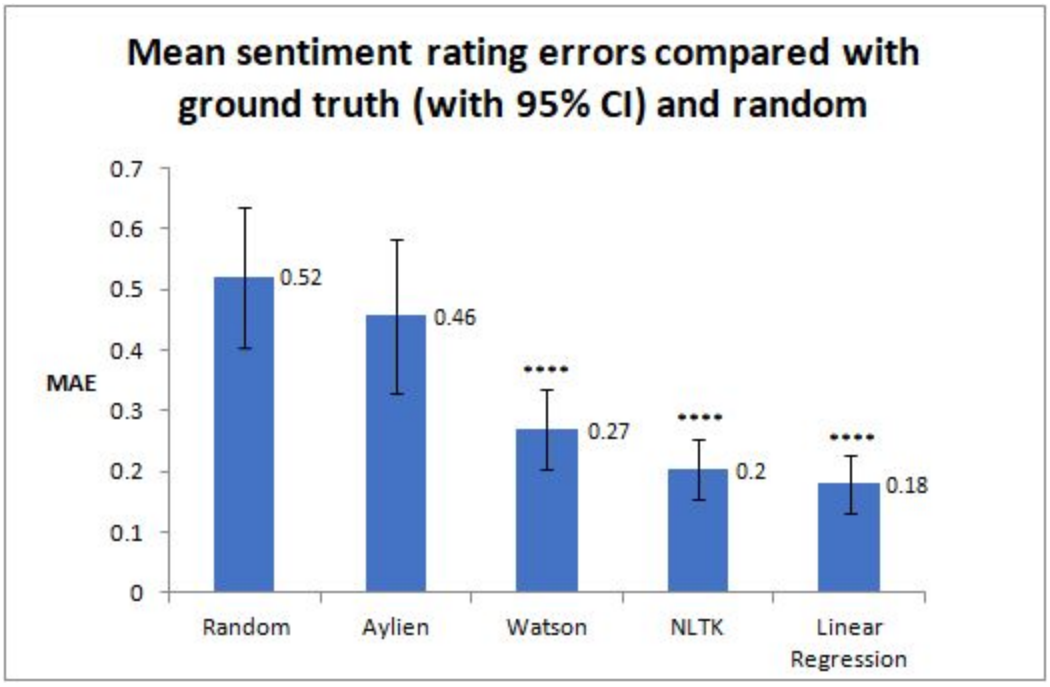

Fig 11. Mean absolute sentiment score error compared with the ground truth.

The linear regression model, Watson sentiment and NLTK were **substantially better** than random at estimating the user-provided ground truths. This validated all the components of the system and data

management that were required to obtain the ground truth and to use and test the sentiment tools (see above). As can be seen in Fig 11, the absolute errors for Watson , NLTK and the linear regression model, but not Aylien, were statistically significantly ($p < 0.0001$) better than random. The linear regression model was also statistically significantly ($p = 0.01$) better than Watson (MAE 0.27; RMSE 0.34). Therefore, we could reject the null for all 3 hypotheses, 1, 2a and 3a.

The MAE of 0.18 for the linear regression model was much lower and was the the lowest (best) of any sentiment or emotion tool, as was the RMSE (**0.23**). It was statistically significantly lower ($p < 0.001$) than all Aylien scores (whole article, title, part 1 and part 2) and emotion scores. It was also statistically significantly lower ($p = 0.02$) than NLTK headline. The difference between the linear regression and NLTK whole article (MAE 0.2), part 1 (MAE 0.23) and part 2 (MAE 0.21) was not statistically significant.

The weighted mean model was developed partly by trying to gain insights from all the user ratings in the experiment. Therefore, we cannot draw strong conclusions by seeing how it performed on the test set (since many of its ratings were studied during the development of the model). Nonetheless, it performed extremely well (MAE 0.19; **RMSE 0.24**), as can be seen visually from the scatter plots (Fig 12). If those results were repeated on an independent test set, then there might be little to choose between it and the current linear regression model.

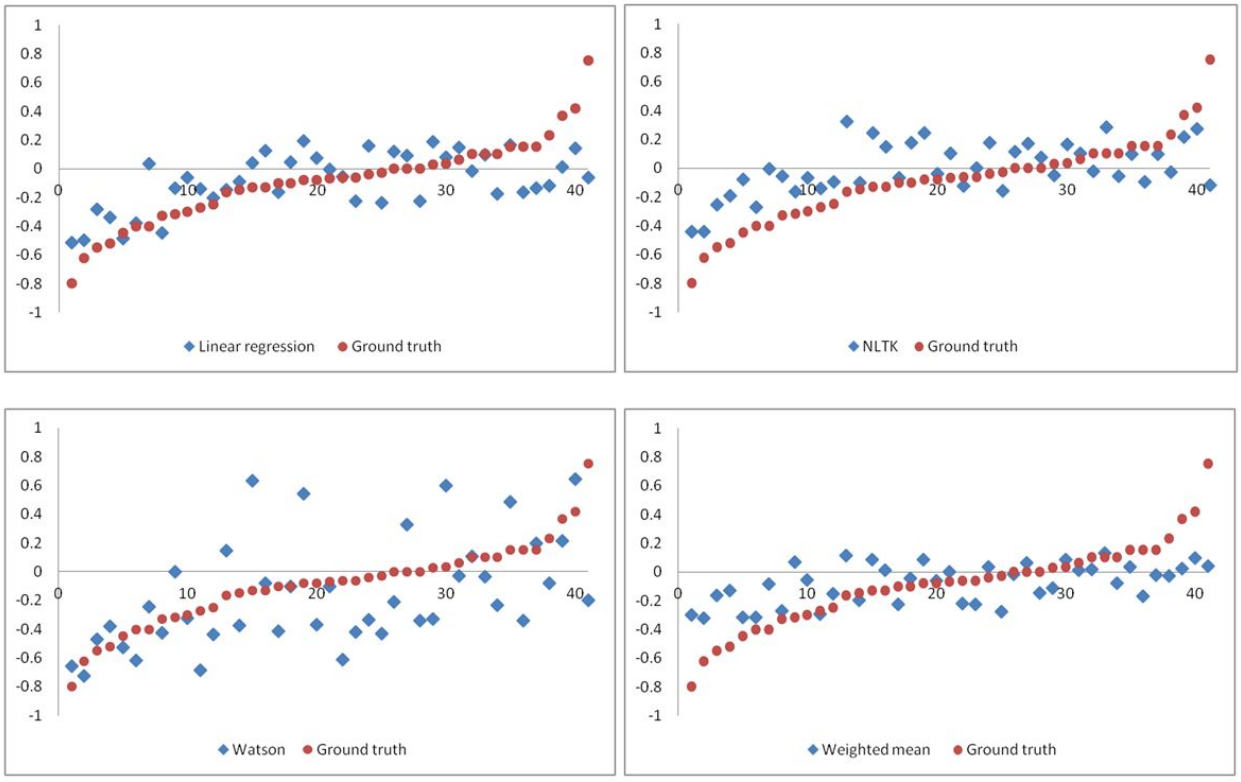

Fig 12. Scatter plots for predicted sentiment by the linear regression model, NLTK, Watson and weighted mean model compared with the ground truth. (See the appendix for Aylien.)

Experiment 2

We designed a 2nd experiment to allow users to filter a selection of news articles to: test the prototype more thoroughly; test the weighted mean model against Watson and a random filter; and find out more about user preferences to improve our system and obtain evidence that we were providing a solution to a real problem. The experiment had 28 users. **We did not participate ourselves.**

We took a set of 175 articles not used in experiment 1. For each user, we displayed 35 articles drawn randomly from the pool, rather than picking a fixed set of 35 articles for all users because the ratings by Watson, the weighted mean and/or random might be unrepresentative for a single small sample.

Users were asked to experiment with each of 3 filters in turn – the weighted mean, a random filter and Watson. **Users were not told which was which**. If they moved any filter to the right and clicked submit, the number of articles displayed would shrink. They were asked to rate each filter on a scale of 5 to -5, according to whether or not the article mix became more positive or negative as they moved the filter to the right.

We aimed to answer hypotheses 1 and variants of the hypotheses 2 and 3 above that, relative to ground truth user ratings:

2b The weighted mean model can identify article sentiment better than random.
3b The weighted mean model can identify article sentiment better than Watson.

We measured the performance of the filters by averaging their ratings across all users. The baseline was that the filters, on average, did not change the positive/negative mix. We compared the performances of the 3 filters with each other and with the baseline to obtain evidence relating to hypotheses 1, 2b and 3b.

We also asked users a number of other questions* relating to additional hypotheses designed to validate and improve our system:

4. More than 5% of consumers of online news would use a sentiment filter, given the option. *Is a news filter viable?.*
5. More users care more about avoiding extremely negative news than about seeing only positive news. *Is detecting strongly negative news accurately more important than accurately detecting positive news?*
6. More than 20% of users will judge the effectiveness of the sentiment filters by looking at headlines only for 50% or more articles. *How important is headline sentiment?*

**Full list of questions asked can be found in Appendices section*

We conducted a pilot version of this experiment in which users were presented with a different mix of 10 articles depending on the filter setting. However, from feedback we discovered that some users found this confusing, so we switched to the approach described above, which received better feedback. We combined the results of the 2 experiments.

As with the 1st experiment, the data collected from users was on a subjective *Likert* scale. These scales were explicitly numeric or could be converted to a numeric scale. We also obtained quantitative data from Google analytics on how many articles users clicked on during the 2nd experiment.

With subjective scores or scales, as Susan Jamieson points out, *"the intervals between values cannot be presumed equal" (2004)*. She argues, therefore, that one cannot use means or apply parametric tests on the results. However, others disagree with this absolutist position. Geoff Norman argues that the most commonly parametric statistical techniques are robust to changes in underlying assumptions about the data (2010). While the mean of ordinal responses is imprecise and should not be taken *"to the 100th decimal place" (Sullivan and Artino, 2013)*, numerical analysis of the ordinal data is sufficient for the purpose of validating and improving our model.

Experiment 2 results

The users rated the random filter close to neutral (**0.19** on a scale of 5 to -5) at making the article mix more positive or negative. This is what we would have expected and is evidence that users were giving honest ratings for the filters. The users found that the weighted mean and Watson filters did make the article selection more positive (see Fig 13). Both were statistically significantly better than random ($p = 2x10^{-4}$ and $3x10^{-5}$, respectively). There was no statistically significant difference between the 2 models although the weighted mean model had a higher average rating (**2.5 vs. 2.4**). Therefore, we could reject the null for hypotheses 1 (again) and 2b but not for hypothesis 3b.

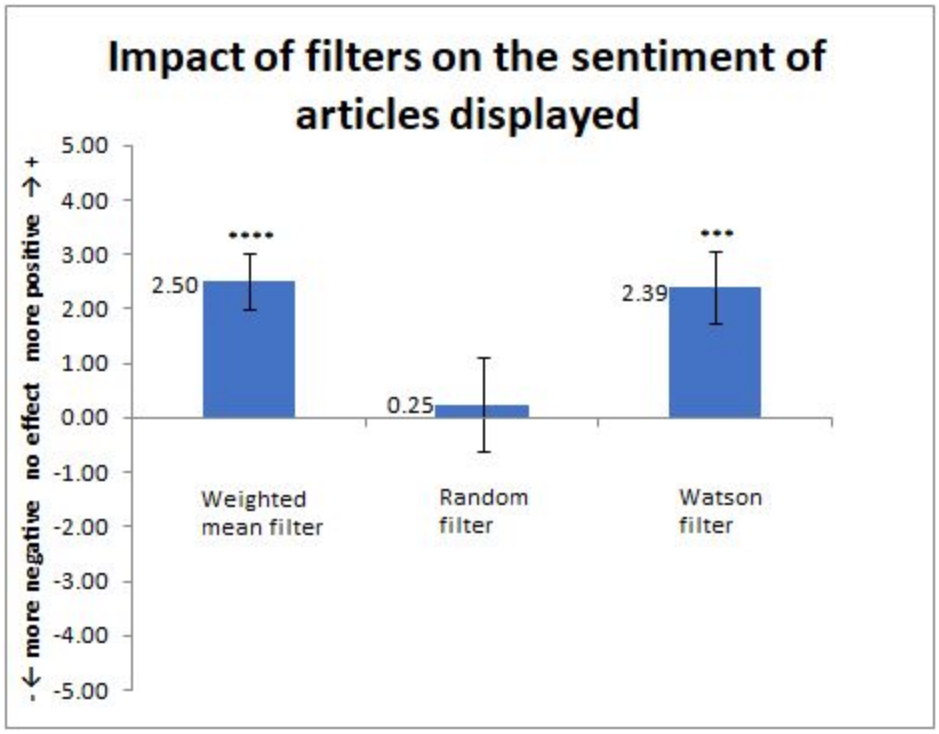

Fig 13. Average user rating for different filters according to how much each made a displayed collection of articles more positive.

We asked users, if filtering was available on another website what setting would they choose, what default setting they would like and whether or not they would adjust the default setting if it did not match their preference. We anticipated that if they liked our prototype they would be keen to use a filter on other sites. We found that 50% would use a filter, 43% would like the default to be positive and 75% would adjust the filter if it did not match their preferences (see Fig 14).

This seemed clear evidence that hypothesis 4 was correct, even allowing for the fact that the users were likely biased, since they knew us.

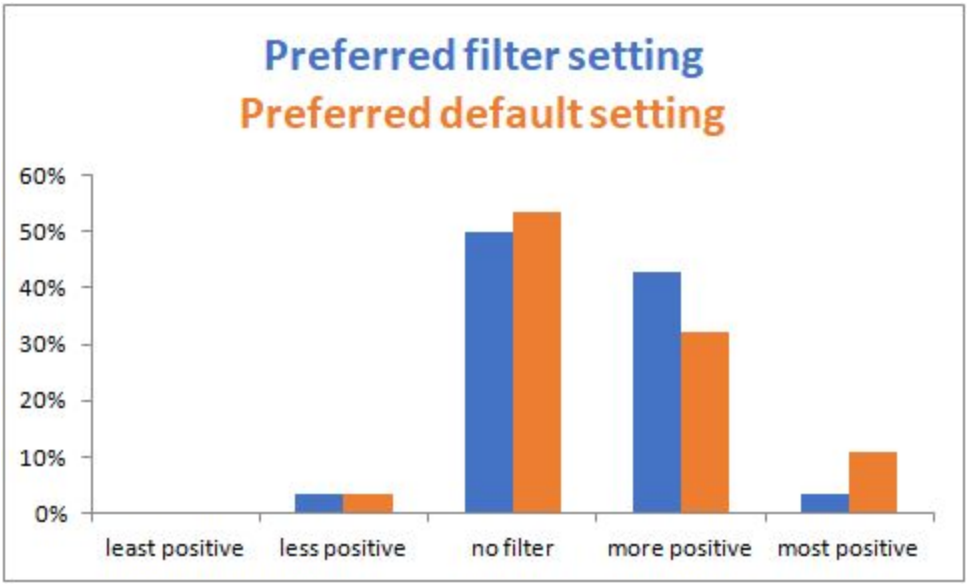

Fig 14. Preferred filter settings for survey participants' favourite news website and preferred default settings on a news website.

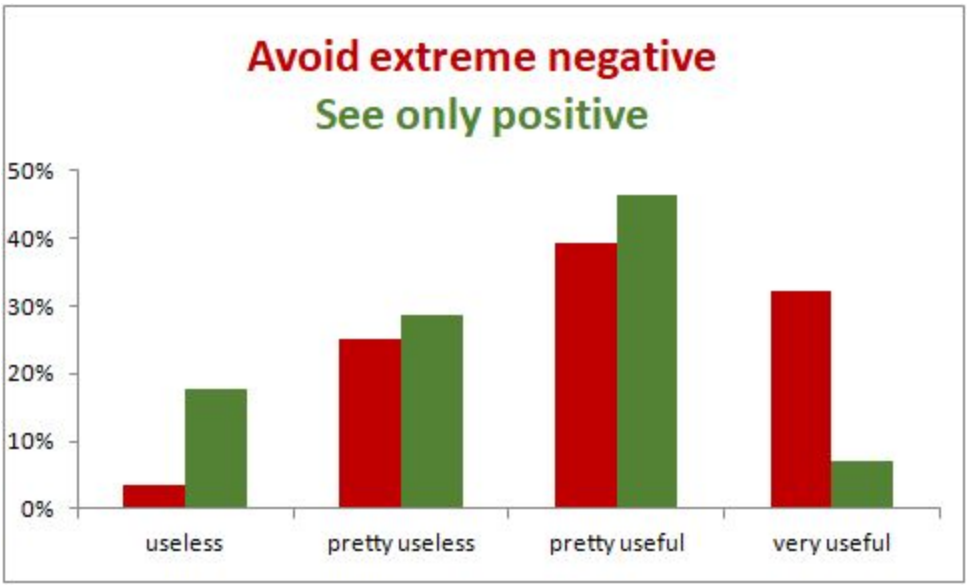

Fig 15. How useless/useful survey participants would find a tool that could help them to control how much extremely negative news they see or help them see only positive news.

We also asked users whether it would be useless, pretty useless, pretty useful or very useful to see only positive news or (in a separate question) to avoid extremely negative news (see Fig 15). The results were that 54% said it would be useful (46%) or very useful (7%) to see only positive news, while **71% said that avoiding extremely negative news would be useful** (39%) or very useful (32%). When we converted

the answers to numbers, there was a statistically significant difference ($p = 0.04$). Therefore the null was rejected for hypothesis 5.

Finally, we asked users whether they rated the filters after reading mostly headlines, mostly articles or 50-50. **All but one** of the 28 users answered mostly headlines (57%) or 50:50 (39%), which was clear evidence for hypothesis 6. We found further evidence that this was true by examining user behaviour with Google analytics. The average user read just 6 articles.

We used Google analytics to track the flow of users to the site and determine their behaviour. This user spent on average **2.5 minutes** to read 6 articles (interactions displayed below). As each article was at least 1800 characters, it suggests that most articles were skimmed or only partly read. The evaluation answers indicated that users read many more headlines than articles, which the Google Analytics results seem to support.

The figure below shows the flow of interaction on our evaluation site for a user.

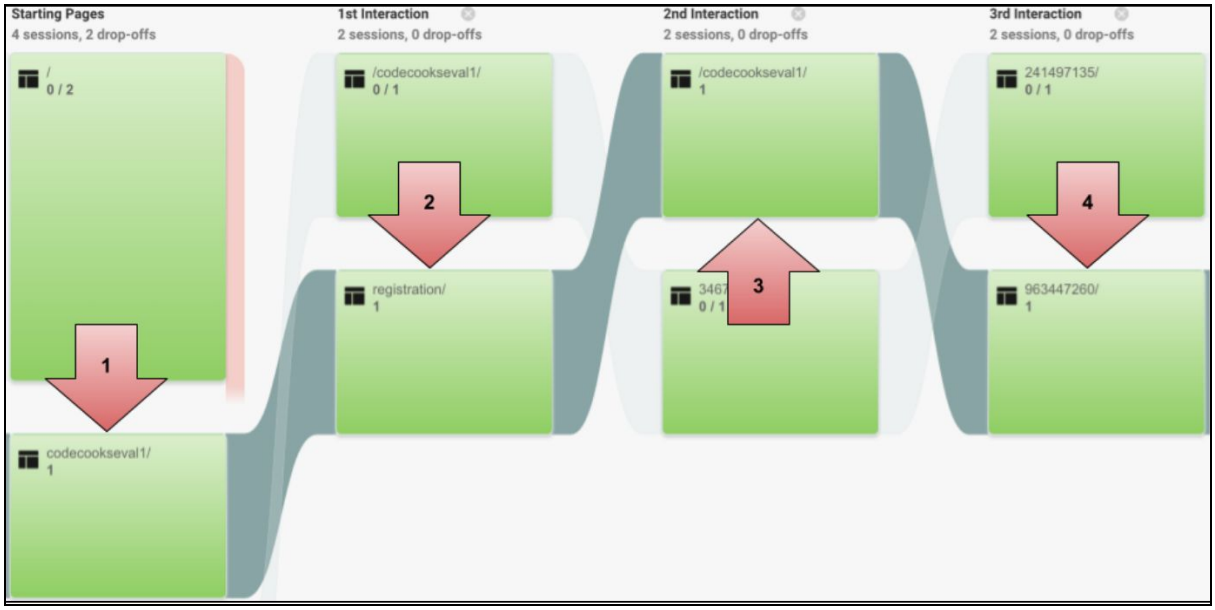

Fig 16. Evaluation User Flow

1. Shows that our user arrived at our evaluation site.
2. The user has clicked on our registration page to participate in the evaluation a user must be registered to allow us to accurately track article ratings.
3. Shows that the user was redirected to the articles page.
4. Shows that our user clicked article with a uniqueid of 963447260.

Evaluation conclusions

Our evaluation experiments helped us to learn several things about our users and system.

Evidence there is a problem to be solved

Half our users said they would use a filter, **75% said they would change the default setting if it didn't match their preferences** and 32% said it would be 'very useful' to avoid extremely negative news. All this suggests that there is a real market for a sentiment filter for daily news.

Evidence a viable solution exists

We found that Watson, NLTK and both models were **significantly better than random at filtering** news articles by sentiment, when compared with user ratings. The linear regression model had smaller mean average errors than any other sentiment tool in testing and was statistically significantly better than Watson and Aylien. This supports our decision to build our own models, combine existing sentiment tools, take headline sentiment into account and use emotion scores.

Evidence to assist with model design and implementation to meet user expectations

We found users were more concerned about avoiding extremely negative articles than seeing only positive articles, which suggests that Type I errors when detecting very negative articles may be less important than Type II errors. Therefore, we may wish to consider additional ways of weighting potentially negative articles, such as by looking at similar articles, taking entities into account, or using a dictionary of key negative words.

We obtained headline sentiment data and used it in both models because we expected few users to read entire articles and many to just read headlines (Manjoo, 2013; American Press Institute, 2014). The experiment results validated our approach.

Evidence that our solution works and is valued

Our app is related to a recommender system, since it is recommending articles based on their sentiment. Therefore, it is instructive to look at evaluations of such systems. Knijnenburg, et al. (2012.) found that perceptions of recommendation quality was important in "predicting three components of user experience: process (e.g. perceived effort, difficulty), system (e.g. perceived system effectiveness) and outcome (e.g. choice satisfaction)." This supports our decision to focus on user perceptions of sentiment filtering quality.
In the context of recommender systems, it has been observed that "reduced browsing indicates higher system effectiveness" (Knijnenburg, et. al., 2012). The results from Google analytics for the 2nd experiment, suggest that it did not take long for users to judge how well a filter worked. More significantly, the fact that Watson and both models were found to have performed well suggests that all the aspects of the system (discussed above) that contributed to the experiments worked correctly.

Finally, the strong interest expressed by the users in using filters and the results for the filters suggest that **a substantial number found the Watson and weighted mean filters to be useful,** easy to use and effective.

Part 5 Conclusions

Project management and strategy

As a team working part-time on the project while managing full-time jobs we needed to be efficient in how time was spent. We used Slack, Skype and Whatsapp to communicate. On Slack we sent over 22,000 messages alone over the 10 weeks we used it (about 300 a day). As the project progressed, though, we often met face to face in groups of 2 or 3 to work through a particular challenge.

We made early use of the task tracking tool *Trello*. This enabled us to break the technical challenges into individual tasks, assign a team member and set a deadline. We could also comment on progress if needed.

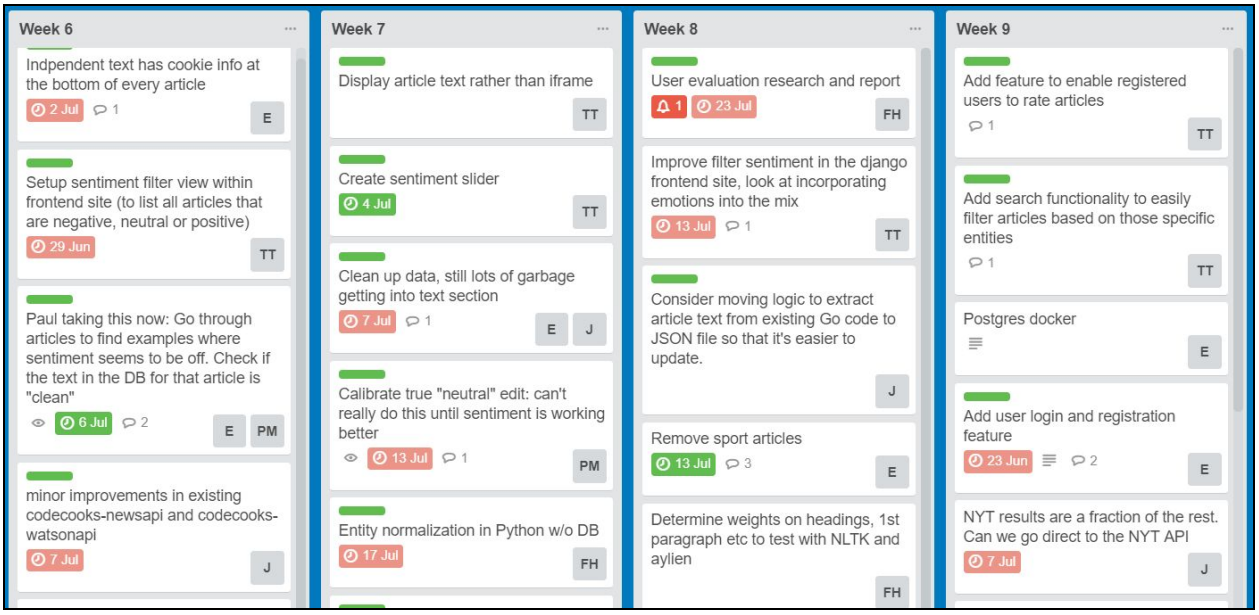

Fig 17: Trello cards

Github allowed us to separate technical areas into repositories. When working on a repository team members were **required to make a fork** of the repo and then commit changes for approval by another team member before they were available in the master. This worked well - broken code rarely made it to the Master.

Project challenges

Decision on direction

An early idea formed that tackled the area of context-based news presentation. We recognised we might not execute it successfully in the time allowed - it was high risk. We decided to use sentiment analysis to provide positive news instead. We lost some time here but **identified the risk early**.

Timing of deliveries

We recognised that we needed ground truth to develop a model but we needed a working site to get ground truth. To get up and running on a model to predict sentiment we created an article list to rate within the group. Within 2 weeks of this we had an evaluation site ready and moved from inputting user ratings manually into a spreadsheet to harvesting all of this data from friends and colleagues via the web interface. This was an example of how **ensuring when tasks fed into each other** at the right time was key.

Opinions

We also faced occasions where opinions differed on how to tackle certain challenges and varying styles of making contributions. Some of us preferred to work alone, others benefited from collaboration. Some situations arose that created some strain amongst the team. Once recognised and resolved we were able to move on to develop an excellent product.

What did we learn from this project

Learned that our system worked (and users valued it)

The difficulty with developing a body of work that can only reveal its performance near completion is the risk that if some link in the chain is broken it may only be discovered at the very end. What if our approach was flawed? What if we had analysed 14,000 articles incorrectly, applied dirty text, stored or linked the database tables in the wrong way? Reading the sentiment results from the model on various articles indicated it was working to some extent. But a team of 5 can only evaluate sentiment performance on narrow window of results.

However the evaluation results were a clear confirmation that **no gross problem existed** in how each step functioned; from backend to sentiment analysis, data storage and frontend. The Model filter performance was clearly distinguishable from the random slider and even slightly ahead of Watson's performance. This was a relief, as the user evaluation could only be done once, if we found a technical problem there was little opportunity to ask users to test again. From the evaluation we also found that users valued using a filter and would default that filter to positive rather than neutral or negative.
The **key contributions of our project** were that we found a **real problem**, examined a **viable solution**, **implemented that solution**, discovered **user preferences** and that we possibly have a **better filtering method** than Watson. The statistically significant results were a strong indication here.

News is negative

Only **23%** of the 14,232 stories we collected over a 9 week period were scored by our model above 0, or positive. Additionally no news source had a mean sentiment that was positive with Al-Jazeera noticeably more negative than any other source. This shows positive stories can be lost in the sea of negative ones.

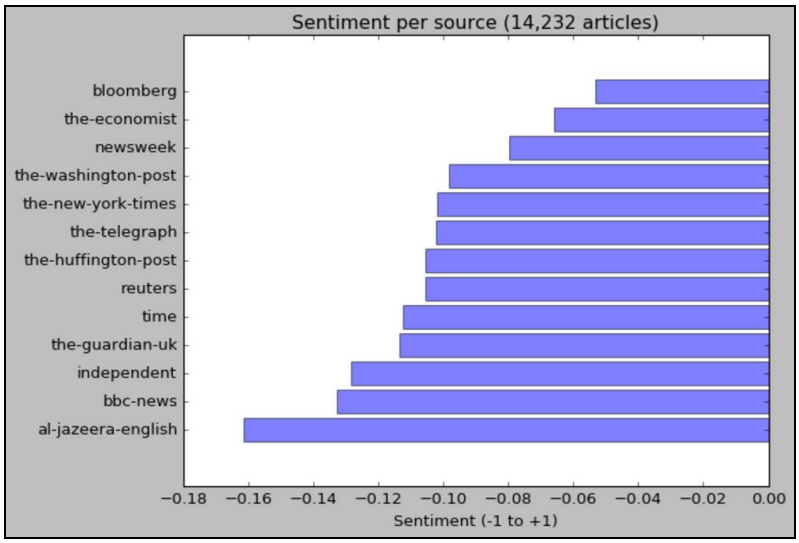

Fig 18: Average sentiment per source

The black box of sentiment tools

When we trialled the idea of using IBM Watson sentiment analysis to recognise news articles that were either positive or negative we could see that it worked reasonably well. But we were never sure exactly *how* it was working. We contacted IBM, they were understandably guarded in supplying trade knowledge:

Response from IBM:

> *"There is a temporal component, in that we attempt to keep our public services reflecting the current world. It is real-time (i.e. changing public sentiments are not reflected instantly). Emotion is not taken into account in calculation of Sentiment scores Unfortunately ….. I cannot offer detailed explanations of how we compute our responses as that is considered a trade secret and IP"*

Once we included NLTK and Aylien as input data for the linear regression, though, we were able to significantly improve (statistically) on Watson.

The importance of (early) ground truth

Sentiment can have widely varying interpretations - there isn't always a right answer. We needed to gauge how a human set of ratings would compare against our machine approach. However our frontend was not ready to use for article ratings. We distributed a list within the team that gave a starting point for our model. We learned early that humans did not consider the text quite the same way as a sentiment tool. It was clear that title and initial paragraphs played a major part. This data helped but we could have done with a lot more and probably sooner. From that point, we prioritised collecting more ground truth and

we applied more resources into getting an evaluation site working that gave us more to work with. This paid off; we managed to get valuable information from users about the direction to take with our model.

Future work

We succeeded in creating a web-based prototype that aims to allow users to filter news articles by sentiment and entity. From this a number of potential expansions to the idea can be explored.

Using user ratings directly

Our site is setup to accept user ratings for any article. Currently the sentiment scores are provided via the model calculation. A potential exists to recognise when user ratings should take the place of the model prediction. The backend infrastructure supports an easy transition to using a human rating by setting a threshold on the number of human ratings required to replace the model value.

Using user ratings with the model

The biggest source of improvement for our model is obtaining more user data. Our linear regression is fully-automated to use more ground truth and adapting the sentiment variables, training a new model and adjusting model scores. Adjusting the model and updating the model scores on all 14,000 articles in the database can be done within 30 minutes. As user scores come in we can recognise the increase and automate the regression to run once new ratings pass a threshold. Our metric for improvement is the RMSE between the new scores and user ratings.

Customised sentiment scores

The fundamental problem with sentiment is that it is subjective. However, if we monitor users and gain insight to how they rate articles, if at all; we may be able to tailor sentiment filtering to individuals.

Using negative entities with the model

User data is the best source of model improvement, but over time, other features may appear. One example is the temporal recognition of entities that may play a part in a sentiment score. An example of this is the presence of *Donald Trump* in the articles we collected. Anecdotally, it appeared that Trump was a notable entity in stories that were often scored negatively. Unsurprisingly, when articles where Trump featured prominently were removed, the results indicated that on average sentiment improved in 12 out 13 sources. In one case, Bloomberg news stories benefited from a **7% sentiment boost** when Trump was removed.

REST API for research

We added support for a RESTful application program interface that allowed any 3rd party with a valid key to make requests and extract data they found useful. Our project is continuously running, collecting and scoring news articles every day. This body of work may provide another research group or media entity the resource to develop on the sentiment challenge further, or present it to a wider audience. With REST infrastructure we have simplified this knowledge transfer and have put the tools in place to extract new data as news articles are created around the world.

Part 6: References

Alan A. A. Donovan and Brian W. Kernighan, The Go Programming Language, *Addison-Wesley Professional; 1 edition*, November 5, 2015

William Kennedy, Brian Ketelsen, Erik St. Martin, Go In Action, *Manning Publications; 1 edition*, November 26, 2015

Matt Butcher, Matt Farina, Go In Practice, *Manning Publications; 1 edition*, October 1, 2016

Sau Sheong Chang, Go Web Programming, *Manning Publications; 1 edition*, July 22, 2016

Ian Miell, Aidan Hobson Sayers, Docker In Practice, *Manning Publications; 1 edition*, May 14, 2016

American Press Institute (2014). How Americans get their news. *American Press Institute*, March 17, 2014.

Besley, T. and M. Kudamatsu (2006). Health and Democracy. *American Economic Review*, Vol. 96, No. 2 (May, 2006), pp. 313-318.

Bhatia, R. (2017). Quietly Erasing Democracy Promotion at the U.S. State Department. *Freedom House*, August 8, 2017. Retrieved from https://freedomhouse.org/blog/quietly-erasing-democracy-promotion-us-state-department.

BBC (2017b). Trump says terror attacks 'under-reported': Is that true? *BBC*, February 7, 2017. Retrieved from http://www.bbc.com/news/world-us-canada-38890090.

BBC (2017a). France's Marine Le Pen urges end to Russia sanctions. *BBC*, 24 March 2017. Retrieved from http://www.bbc.com/news/world-europe-39375969.

Bloodworth, J. Why is no one asking about Jeremy Corbyn's worrying connections? *The Guardian*, August 13, 2015. Retrieved from https://www.theguardian.com/commentisfree/2015/aug/13/jeremy-corbyn-labour-leadership-foreign-policy-antisemitism.

Croucher, S. (2015). Jeremy Corbyn charge sheet: From Russia's useful idiot to his 'friends' Hamas and Hezbollah. *International Business Times*, August 15, 2015. Retrieved from http://www.ibtimes.co.uk/jeremy-corbyn-charge-sheet-russias-useful-idiot-his-friends-hamas-hezbollah-1515529.

Dewey, C. (2016). Facebook Fake-News Writer: 'I Think Donald Trump is in the White House Because of Me.' *The Washington Post*, November 17, 2016. Retrieved from https://www.washingtonpost.com/news/theintersect/wp/2016/11/17/facebook-fake-news-writer-i-think-donald-trump-isin-the-white-house-because-of-me/.

Dobelli, R. (2013). News is bad for you - and giving up reading it will make you happier. *The Guardian*, April 12, 2013. Retrieved from https://www.theguardian.com/media/2013/apr/12/news-is-bad-rolf-dobelli.

Doornik, J. A. and D. F. Hendry (1994). PcGive Student 8.0: An Interactive Econometric Modelling System. *International Thompson Publishing*, 1994.

Fernández J., E. Boldrini, J.M. Gómez a this nd P. Martínez-Barco (2011). Evaluating EmotiBlog Robustness for Sentiment Analysis Tasks. In: Muñoz R., Montoyo A., Métais E. (eds) Natural Language Processing and Information Systems. NLDB 2011. *Lecture Notes in Computer Science, vol 6716*. Springer, Berlin, Heidelberg. DOI 10.1007/978-3-642-22327-3_41.

Gregoire, C. (2015). What Constant Exposure To Negative News Is Doing To Our Mental Health. *Huffington Post*, February 19, 2015. Retrieved from http://www.huffingtonpost.com/2015/02/19/violent-media-anxiety_n_6671732.html.

Gujarati, D. N. (1995). Basic Econometrics. *McGraw-Hill*, 1995, 3rd edition.

Hoefer, H. (2017). America = Russia? *US News*, February 6, 2017. Retrieved from https://www.usnews.com/opinion/views-you-can-use/articles/2017-02-06/donald-trump-defends-vladimir-putin-by-comparing-america-to-russia.

Howard, P. N., G. Bolsover, B. Kollanyi, S. Bradshaw and L.-M. Neudert (2017). Junk News and Bots during the U.S. Election: What Were Michigan Voters Sharing Over Twitter? *Project on Computational Propaganda, University of Oxford*, Data Memo 2017.1. Retrieved from http://comprop.oii.ox.ac.uk/2017/03/26/junk-news-and-bots-during-the-u-selection-what-were-michigan-voters-sharing-over-twitter/

IBM (2017). Entity types and subtypes. *Watson Developer Cloud*. Retrieved from https://www.ibm.com/watson/developercloud/doc/natural-language-understanding/entity-types.html on June 30, 2017.

Jamieson, S. (2004). Likert scales: How to (ab)use them. *Medical Education, 38*, pp. 1217–1218.

Johnson, R. J., P. Andrews, S. A., Benner and W. Oliver (2010). Theodore E. Woodward Award: The Evolution of Obesity: Insights from the Mid- Miocene. *Trans Am Clin Climatol Assoc*, 121, 295–308. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2917125/.

Knijnenburg, B.P., M. C. Willemsen, Z. Gantner, H. Soncu and C. Newell (2012). Explaining the user experience of recommender systems. *User Model User-Adap Inter 22*. pp. 441 – 504. DOI 10.1007/s11257-011-9118-4

Manjoo, F. (2013). You Won't Finish This Article. *Slate*, June 6, 2013. Retrieved from http://www.slate.com/articles/technology/technology/2013/06/how_people_read_online_why_you_won_t_finish_this_article.html.

Marin, M.-F., J.-K. Morin-Major, T. E. Schramek, A. Beaupré, A. Perna, R.-P. Juster and S. J. Lupien (2012). There is no news like bad news: women are more remembering and stress reactive after reading real negative news than men. *PLoS ONE*. 2012;7(10):e47189 DOI 10.1371/journal.pone.0047189. Retrieved from https://doaj.org/article/ad04f346a4b54eca9d1529631aa71be5.

McIntyre, K. E. and R. Gibson (2016). Positive News Makes Readers Feel Good: A 'Silver-Lining' Approach to Negative News Can Attract Audiences. *Southern Communication Journal* Vol. 81 , Iss. 5,2016

Norman, G. (2010). Likert scales, levels of measurement and the 'laws' of statistics. *Adv in Health Sci Educ., Vol. 15, Issue 5*, pp. 625 – 632. DOI 10.1007/s10459-010-9222-y.

Parkinson, H. J. (2016). Click and elect: how fake news helped Donald Trump win a real election. *The Guardian*, November 14, 2016. Retrieved from https://www.theguardian.com/commentisfree/2016/nov/14/fake-newsdonald-trump-election-alt-right-social-media-tech-companies.

Polyakova, A. (2016). Why Europe Is Right to Fear Putin's Useful Idiots. *Foreign Policy*, February 23, 2016. Retrieved from http://foreignpolicy.com/2016/02/23/why-europe-is-right-to-fear-putins-useful-idiots/.

Pu, P., L. Chen and R. Hu (2012). Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Model. User-Adap. Inter. 22*. DOI 10.1007/s11257-011-9115-7.

Reuters (2017). British opposition leader Corbyn declines to condemn Venezuela's Maduro. *Reuters*, August 7, 2017. Retrieved from https://www.reuters.com/article/us-venezuela-politics-britain-corbyn-idUSKBN1AN1QM

Roser, M. (2017). *Our World In Data*. Retrieved from https://ourworldindata.org/, June 30, 2017.

Siegel, L. (2017). Journalists must enlighten, not just inform, in a world darkened by Trump. *CJR*, July 17, 2017. Retrieved from https://www.cjr.org/covering_trump/journalists-trump-news-negative.php.

Stafford, T. (2014). Psychology: Why bad news dominates the headlines. *BBC*, July 29, 2014. Retrieved from http://www.bbc.com/future/story/20140728-why-is-all-the-news-bad.

Sullivan, G. M. and R. A. Artino Jr. (2013). Analyzing and Interpreting Data From Likert-Type Scales. *Journal of Graduate Medical Education*, December 2013, Vol. 5, No. 4, pp. 541 – 542. DOI 10.4300/JGME-5-4-18.

Bellinger, N. M. (2017). Why democracy matters: democratic attributes and human well-being. *Journal of International Relations and Development* 33.

Tierney, J. (2013). Good News Beats Bad on Social Networks. *New York Times*, March 18, 2013. Retrieved from http://www.nytimes.com/2013/03/19/science/good-news-spreads-faster-on-twitter-and-facebook.html.

Trussler, M. and S. Soroka (2014). Consumer Demand for Cynical and Negative News Frames. *The International Journal of Press/Politics* Vol 19, Issue 3, pp. 360 - 379, March 18, 2014, 10.1177/1940161214524832

Woolley, S. C. and P. N. Howard (2017). Computational Propaganda Worldwide: Executive Summary. Samuel Woolley and Philip N. Howard, Eds. *Project on Computational Propaganda, University of Oxford*, Working Paper 2017.11. comprop.oii.ox.ac.uk. 14 pp.

James, B. (2008). Practical Django Projects, 1st edition.

Daniel, R. G. Audrey G. Two Scoops of Django 1.11: Best Practices for the Django Web Framework, 1st Edition

Appendix

Github repositories

Backend

codecooks-watsonapi
Codecooks-newsapi
Codecooks-restapi

Frontend

CodeCooks-DJango

Post-processing

CodeCooks-Repo-ML

General scripts

CodeCooks-scripts-and-general

Docker

The three public Docker repositories with our images can be found here:

- https://hub.docker.com/r/jdiuwe/codecooks-newsapi/
- https://hub.docker.com/r/jdiuwe/codecooks-watsonapi/
- https://hub.docker.com/r/jdiuwe/codecooks-restapi/

RestAPI

Current version of our API offers two simple request types:

1. List of all news sources that we have articles from
http://codecooks.me:9876/api/a71h46fk99/news-sources
2. List of the 10 most recent articles for a specified source
http://codecooks.me:9876/api/a71h46fk99/news-sources/independent

Example output of the sample requests can be viewed in README file in GIT https://github.com/jdiuwe/codecooks-restapi.

Text extraction and cleaning
JSON file with text extraction rules can be viewed in codecooks-newsapi GIT repo:

Linear regression model details

Readme here.

Evaluation questions (experiment 2)

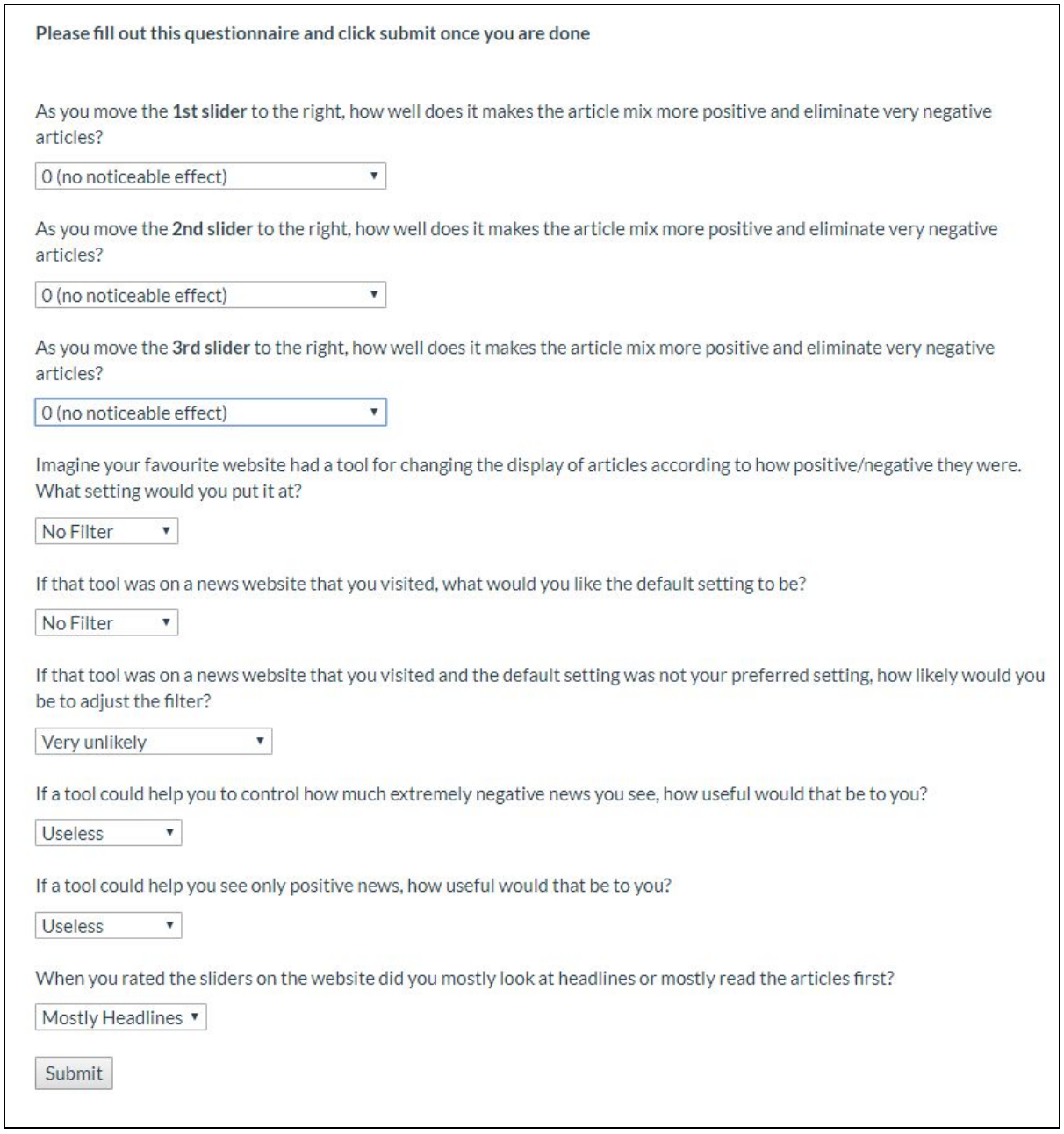

Fig 19: Evaluation questions asked in experiment 2

Aylien

Aylien was found to be less useful than other sentiment tools. However, this may partly be due to its different design. Instead of providing ratings on the scale of -1 to 1, it rates articles as negative, neutral or positive, with a confidence between 0 and 1. To translate its ratings to a -1 to 1 scale, we assumed that articles between -0.2 and 0.2 would be rated neutral, therefore we use the confidence scored to convert positive ratings to the range 0.2 to 1 and negative ratings to the range -1 to -0.2. Further investigation might find a better way to use Aylien scores to contribute to our filtering. We were missing some whole article Aylien scores for the test data, so we performed its t-tests only on data for which scores were available. The following scatterplot shows its accuracy relative to the ground truth.

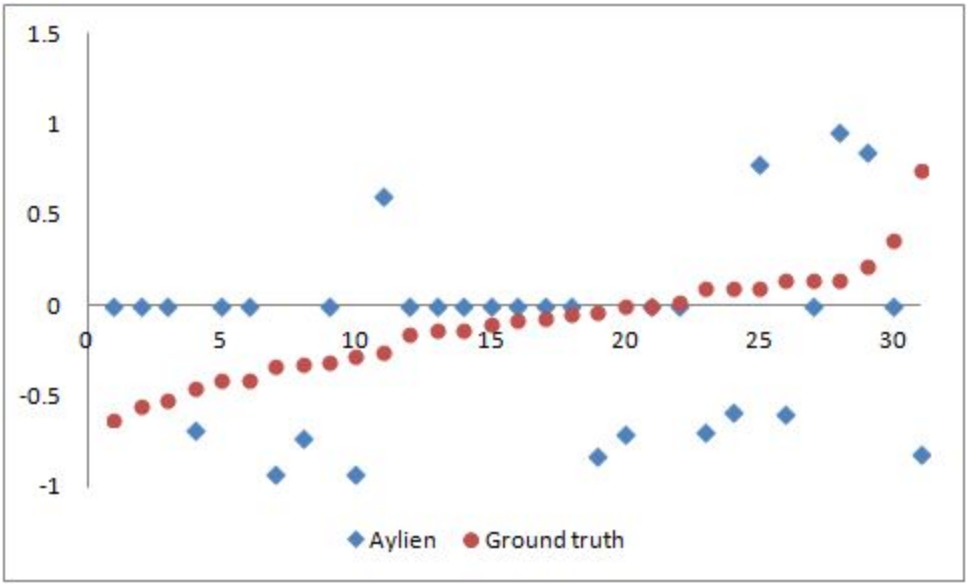

Fig 20: Aylien vs ground truth