Report

*Please download Eluvio_DS_Challenge and GoogleNews-vectors-negative300.bin in the ./**DS_challenge** directory before running the code.

Eluvio_DS_Challenge.csv:
https://drive.google.com/file/d/15X00ZWBjla7qGOIW33j8865QdF89IyAk/view?usp=sharing\

GoogleNews-vectors-negative300.bin:

https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz

The results involve 4 sections:

1. Most used words for each year-month segmentation
2. Words similarity from top voted titles per year
3. Title similarity from top/bottom voted records per year
4. Prediction on voting based on titles

1. Most used words for each year-month segmentation

Method:

It would be interesting to see the high frequency words for each year-month segmentation, those highly used words give us some brief ideas on the hot topics for certain months. To count the frequency of hot words in one particular month, I tokenized each topic, removed stop words and words that contain digits (such as 5k), and counted with 'Wordcloud' package. Top 100 words are selected for each month, words are capitalized for better visualization.

Code access:

To retrieve most popularly used words for each month in certain year, function 'Most_used_words' in WordingTrending.py file plots word boards for the need. Simply set ‘True’ in the corresponding operation request under ‘if __name__ == '__main__'’ (line 254) in the same python file to save the plots under the directory ‘Images/MostPopularWords’.

Results:

12 months/plots for each year are obtained. From the results, we are clearly seeing that the word ‘President’ is highly used in each month of election year (2018,2012,2016) [Fig.1, Fig.2, Fig.3]. Other words like ‘Russia’, ‘China’, ‘Attack’ are often appeared in each plot regardless of the timing. Some hot words are very time-sensitive, for example, ‘Bush’ appeared in all months of 2008 and disappeared in the following years.

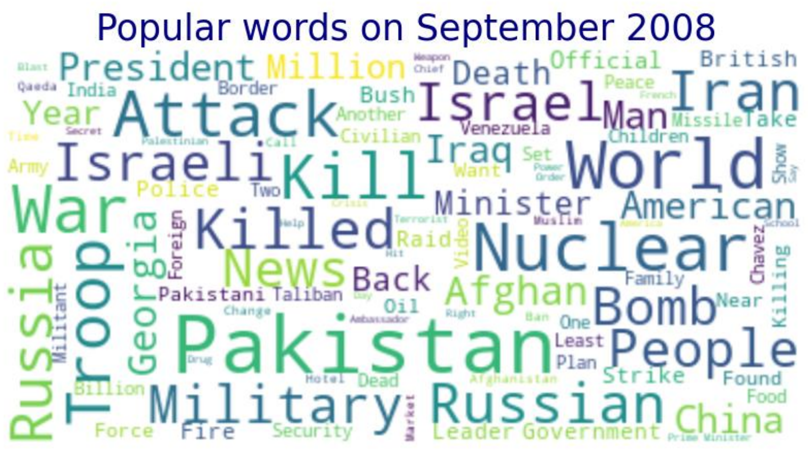

Fig.1

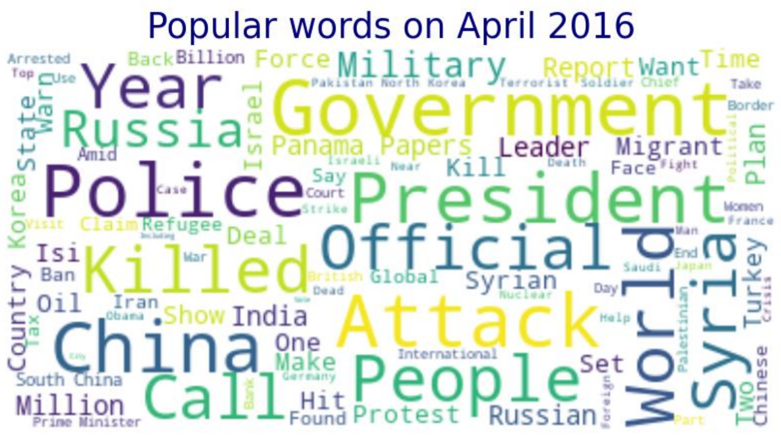

Fig.2

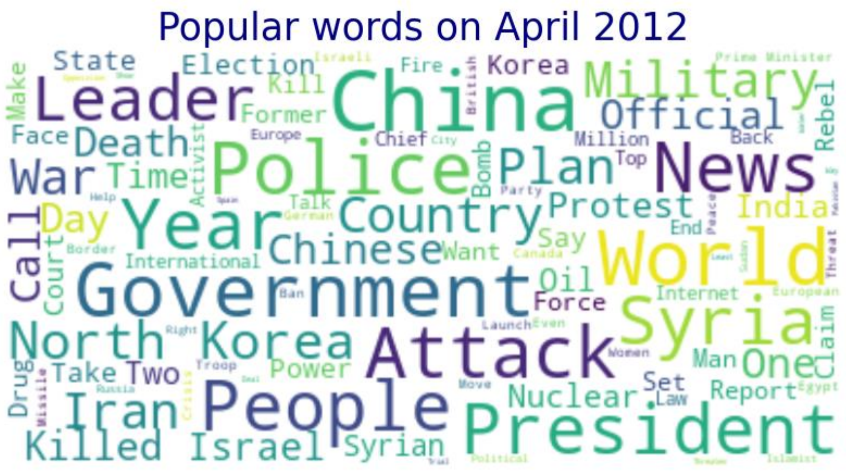

Fig.3

2. Words similarity from top voted titles per year

Method:

To see if words in highly voted titles are clustered together, here we plotted the words used in top 5% up-votes titles for each year. Firstly, I selected titles whose upvotes stands in top 5% of the year, and tokenized words that are contained in those titles. To obtain vectorized words representation, Google pre-trained word embedding (GoogleNews-vectors-negative300.bin) were used to generate word features. For better visualization, I reduced the word representation to 2 dimensions using TSNE package in Python.

Code access:

To retrieve top titles' words similarity for each year, function 'Google_embedding_analysis' in WordingTrending.py file plots for the need. Simply set 'True' in the corresponding operation request under 'if __name__ == '__main__'' (line 256) in the same python file to save the plots under the directory 'Images/ TopVotesWordsSimilarity.

Results:
From the below result example, I observe that words that are used in highly voted titles are generally clustered together. [Fig4]

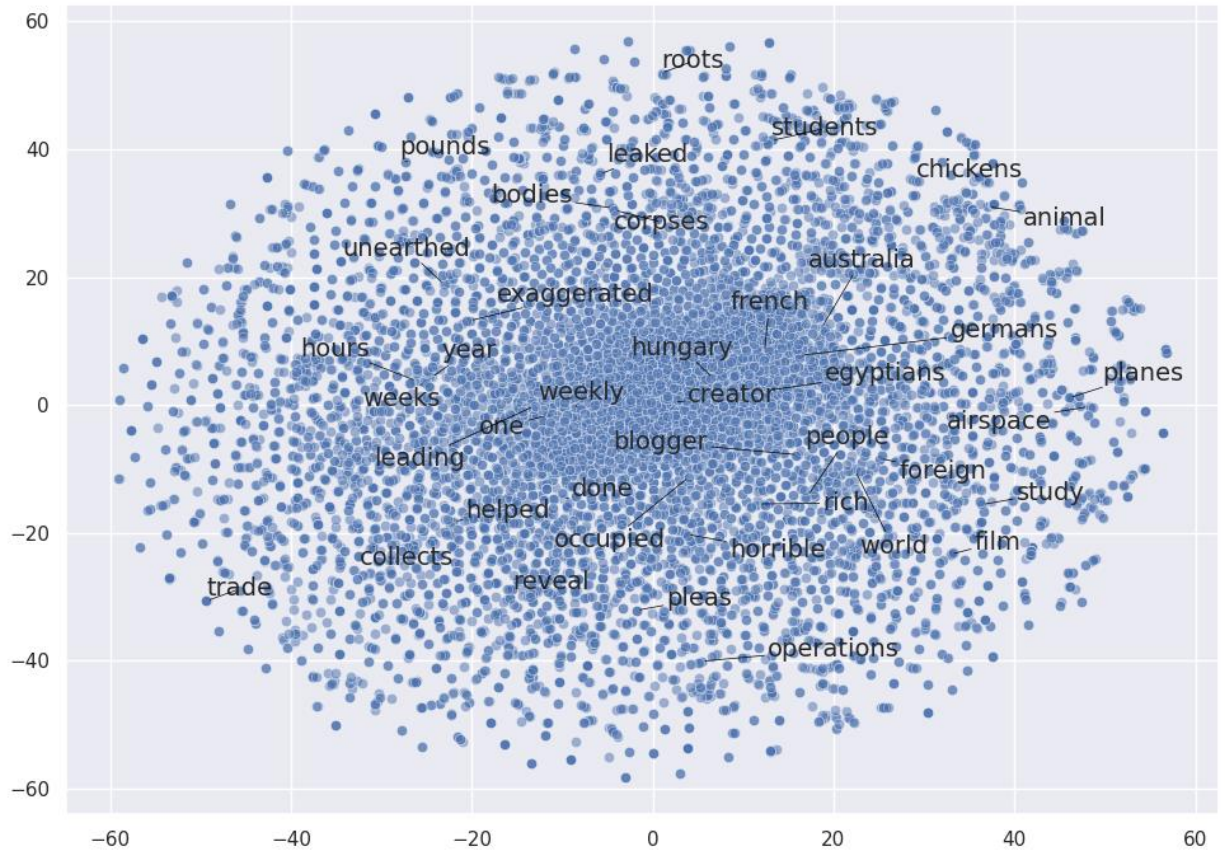

Fig.4

3. Titles similarity from top/down voted records per year

Method:

I’m very happy to see the result from section 2 that words in high voted titles are clustered together, this leads me to wonder if top voted titles are clustered together. Here I represented the whole title from the selected top 30/bottom 30 titles from each year, and visualized these titles in 2-D plots.

Code access:

To retrieve top/bottom titles similarity for each year, function ‘google_embedding_titles’ in WordingTrending.py file plots for the need. Simply set ‘True’ in the corresponding operation request under ‘if __name__ == '__main__'’ (line 258) in the same python file to save the plots under the directory ‘Images/ TopBottomVotesTitleSimilarity.

Results:

From the below result example, I observed that top voted titles are more likely to be at the upper part of the plot and the least voted titles tend to be at a lower position. [Fig5] However, this pattern is not very clear, and my suspicion is on the loss of information for the reduction of word feature dimensions.

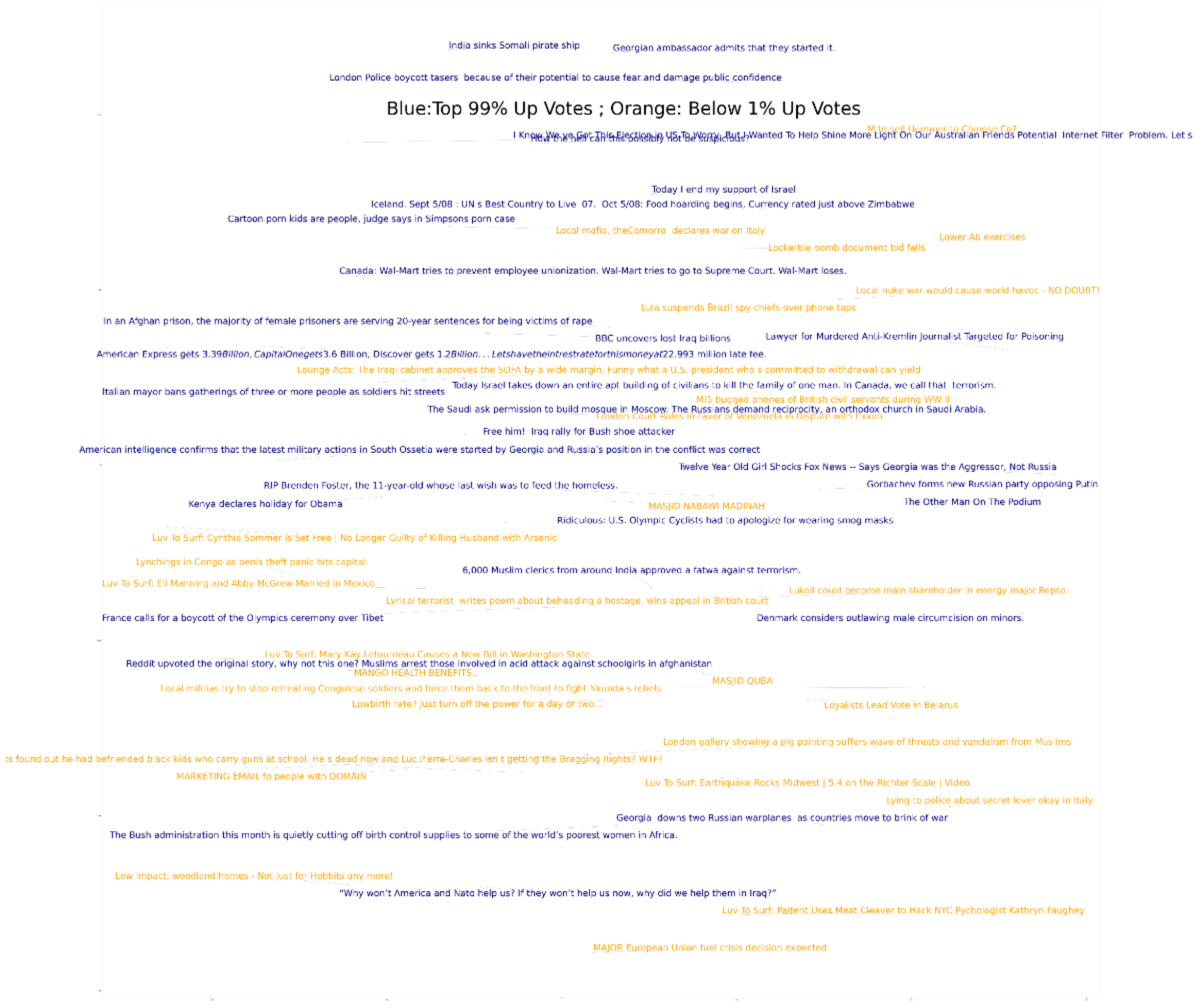

Fig.5

4. Prediction on voting based on titles

Method:

This section builds classification models to predict the votes based on titles. The voting is classified into ‘high’ and ‘low’ using the cutoff - the mean value of overall upvotes score in the dataset. Neural network model is applied using Keras package in Python with 2 benchmark models: logistic regression and Naïve Bayes. Two feature sets (word embedding) are generated for comparison, one used Google pre-trained embedding and the other used the simple frequency matrix (counts of words in positive/negative class) obtained from training data. To avoid negative values in Naïve Bayes model, only frequency matrix feature set is used for the Naïve Bayes model. In summary, this section compared the following 5 models:

1. Logistic regression (frequency matrix as word embedding)
2. Logistic regression (Google word embedding)
3. Naïve Bayes (frequency matrix as word embedding)
4. Keras NN model (frequency matrix as word embedding)
5. Keras NN model (Google word embedding)

Code access:

To run the above model, function 'title_vote_prediction' in WordingTrending.py file does the tick. Simply set 'True' in the corresponding operation request under 'if __name__ == '__main__'' (line 261) in the same python file to save the result plots under the directory 'Images/ Prediction'.