Files

1. scrapeTweets.py - Scrapes tweets from news providers on Twitter and stores them in a text file. Currently only stores the following attributes: tweet id, tweet text, news provider and number of retweets, but these attributes can be modified according to requirement. Note: Twitter credentials need to be input for the script to run.
2. clusterTweets.py - Reads tweets from text (CSV) file storing scraped tweets from news providers on Twitter. Clusters them into different topics according to keywords using mini batch k-means. The clusters are grown over time as new tweets are added to the text file.

Scraping Tweets

Tweets are scraped by a service that pulls data from the list of news providers every 10 minutes. This service runs continuously in the background. The tweets and their attributes are then stored to a CSV file. The tweets could also have been dumped to a database, or pickled as tweet objects, however I chose to write only the required attributes to a CSV file in the interest of space.

Clustering Tweets

The tweets and their attributes that are stored to a text file, are read by another service that runs every 10 minutes. It uses tf-idf to vectorize the tweets, after normalizing them and removing stop words. It then uses mini batch k-means to cluster the tweets according to a default value of 7 for number of clusters. I also investigated the use of DBSCAN, LDA and regular k-means for clustering, but found mini batch k-means to work best in terms of scalability and interpretability. LDA for example, can be used for efficient topic modelling, but does not work as well when identifying which tweets are associated with which topic. DBSCAN on the other hand performs quite poorly on large datasets. This is a problem that is specific to the sklearn library's implementation of DBSCAN however.

The drawback with using k-means, or mini batch k-means, is that one needs to know the number of clusters beforehand. It can be determined through heuristics such as gap statistics or the silhouette method, which, although not guaranteed to give the optimal number, work better than randomly choosing a number. If I had more time, this is what I would implement, but for now, through experimentation, I found 7 as the initial value to work best.

As new tweets are downloaded, it is possible that the number of clusters change. We use a threshold value (= max distance between a cluster centroid and a tweet assigned to that cluster) to determine what is the distance between the new tweets and existing cluster

centroids, beyond which the new tweets would be classified as belonging to different clusters. We “grow” the number of clusters using this threshold, and re-cluster tweets every hour. Over multiple runs, this stabilizes to give us an appropriate clustering.

Detecting Twitter Trends

There are many ways in which one can detect Twitter trends, most commonly, using time series analysis. Here, I implement a naive heuristic for detecting Twitter trends, based on whether a new tweet can be absorbed into an existing topic cluster, and whether it has gone ‘viral’, that is, whether it has received a higher than average number of retweets.
If both are true, then it is likely a trending tweet, and it is added to the list of two trending tweets per day.