DBMS - II
CS14BTECH11022 | ES14BTECH11012
April 30, 2017

PROJECT REPORT

METARYA | MUDIT

OVER VIEW :

This project basically takes a major event such as FA-Cup and takes out all the sub-events in it. Using these sub-events we create a summary of the match (Yes! Summary out of tweets).

There are two files included in it. The first file is where all the pre-processing on the tweets happen. This step is very important as not all tweets are useful. There are "noisy" tweets also which may affect our clustering.

I have submitted 2 files along with this :

1) Extract JSON to text stream
2) Twitter topics from json text stream

Along with them I have output files for the program.

METHOD PROPOSED :

The main approach behind our results for the data challenge is based on: (1) Aggressive tweet and term filtering, to remove noisy tweets and vocabulary; (2) Hierarchical clustering of tweets, dynamic dendrogram cutting and ranking of the resulting clusters, to obtain topics.We chose Python due to the ease of development and its available range of powerful libraries (e.g., scipy, numpy, sklearn). In particular for tweet-NLP, e.g., named entity recognition, we have used a Python wrapper (CMUTweetTagger library), and for efficient hierarchical clustering of tweets, we have used the fastcluster library.

Extract JSON to text stream :

An important part of our method is data preprocessing and filtering. For each tweet, we preprocess the text as follows.

We normalize the text to remove urls, user mentions and hashtags, as well as digits and other punctuation.

Next, we tokenize the remaining clean text by white space, and remove stop words. In order to prepare the tweet corpus, in each time window, for each tweet, we first append the user mentions, the hashtags and the resulting clean text tokens.

We check the structure of the resulting tweet, and filter out tweets that have more than 2 usermentions or more than 2 hashtags, or less than 4 text tokens. The idea behind this structure-based filtering is that tweets that have many user mentions or hashtags, but lack enough clean text features, do not carry enough news-like content, or are generally very noisy. This step filters many noisy tweets.

The next step is concerned with vocabulary filtering. For each time window, from the window tweet corpus, we create a (binary) tweet-term matrix, where we remove user mentions (but keep hashtags), and the vocabulary terms are only bi-grams and tri-grams, that occur in at least a number of tweets, where the minimum

is set to 10 tweets, and the maximum threshold is set based on the window corpus length, to max(int(len(window corpus) * 0.0025), 10). The idea behind this filtering step, is that clusters should gather enough tweets to be considered a topic at all (e.g., at least 25 tweets in 10,000 tweets should discuss an event).

In the next filtering step, we reduce this matrix to only the subset of rows containing at least 5 terms (tweets with at least 5 tokens from the vocabulary). This step is meant to remove out-of-vocabulary tweets, as well as tweets that are too short to be meaningfully clustered.

Twitter Topics from JSON text stream :

Computing tweet pairwise distance :

We compute tweet pairwise distances and a hierarchical clustering on the filtered tweet-by-term matrix. For pairwise distances we scale and normalize the tweet-term matrix, and use cosine as a metric. We use the sklearn and scipy python libraries for computing distances and the tweet-term matrix.

Computing hierarchical clustering:

For computing a hierarchical clustering, we use the f astcluster library that can efficiently deal with thousands of tweets/terms. The idea behind tweet clustering is that tweets belonging to the same topic will cluster together, and thus we can consider each cluster as a detected topic.

Cutting the dendrogram:

Finally, we cut the resulting dendrogram at a 0.5 distance threshold. This threshold can control how tight or loose we require our final clusters to be, without having to provide the number of clusters expected a-priori, e.g., as for k-means or other popular clustering algorithms. A higher threshold would result in looser clusters, that potentially collate different topics in the same cluster. A lower

threshold would result in tighter and cleaner clusters, but potentially lead to too much topic fragmentation, i.e., the same topic would be reflected by lots of different clusters. We found that a value of 0.5 works well for our method.

Ranking the resulting clusters:

Once we obtain clusters with the above procedure, we assign a score to each cluster and rank them based on that score. A first attempt was to score and rank clusters by size, therefore allowing clusters with a lot of tweets to rank first as trending topics. This results in topics that tend to be more casual and are unlikely to make the news headlines (e.g., This is what happens when you put two pit bulls in a photo booth), as we show in our evaluation section. Additionally, topics tend to get frequently repeated for several time windows, since we do not consider potential term/topic burstiness in each time window with respect to the previous time windows.

Next, we introduce term weighting, based on the frequency in the time window, as well as boosting named entities. For the frequency based weight, we use the df − idft formula from that discounts the term-frequency in the current time window using the average frequency in the previous t time windows.
We also used a python wrapper around the CMU Java code. This tool is trained on tweets and had better accuracy for named entity recognition in our tests. We apply this tool to each of the terms in our vocabulary, in order to recognize entities

Selecting topic headlines:

We select the first (with respect to publication time) tweet in each cluster of the top-20, as our headline for the detected topic. This clustering/ranking strategy covers several events but many times suffers from topic fragmentation, e.g., we may get several headlines about the same topic. Next we discuss strategies for dealing with topic fragmentation and reducing the set of topics to only top-10.

Re-clustering headlines to avoid topic fragmentation :

Our final step involves clustering of only the headlines selected after the first stage of clustering and ranking. These are cleaned tweets used for clustering in the

first stage (no user mentions, urls, filtered vocabulary). We build a headline-by-term matrix, using unigrams for our vocabulary, without any other restriction on terms. We re-cluster the headlines using hierarchical clustering, and cut the dendrogram at the maximum distance (e.g., 1.0 for cosine). Again setting this threshold decides how many headlines we want to collate into a single topic. We rank the resulting headline-clusters using the headline with the highest score inside a cluster, therefore if the headlines do not cluster at all, the ranking of headlines will stay the same as in the previous step.

Final selection of topics:

From this final clustering and ranking step, we select the headline with the earliest publication time, and present its raw tweet (without urls) as a final topic headline. We pool the keywords of the headlines in the same headline-cluster to extract topic-tags (a list of keywords as a description of the topic). For selecting tweet ids relevant to the extracted topic, we use the ids of the clustered headlines (i.e., the id of the tweet corresponding to the headline), and otherwise a single id, if the headline-cluster contains a single headline. The idea behind this strategy is that if the first stage of clustering did not split a topic, the tweets inside the topic-cluster were very similar to each other. For extracting urls of photos relevant to the topic, we first check if the headlines have any media url tags (as extracted from the JSON object), and if not, we loop through the cluster (from stage 1) of tweets to which the headline belongs, in search of a media url in those tweets. Restricting the number of media urls to 1 or 2 directly affects the speed of the overall topic extraction process, since we don't have to dive too deep into the previous (potentially large) clusters.

Conclusion:

We present a method for topic detection in Twitter streams, based on aggressive tweet/term filtering and two stage hierarchical clustering, first of tweets and second of resulting headlines from the first clustering step. The topics obtained seem encouraging, many of them being published as news in the traditional news media. Our topic-headlines are actual tweets, so the user can trace the news back to its original tweet, and are presented in the context of photos (from tweet media urls) and tags selected from those tweets. One of the potential weaknesses of our

method consists in the aspect of topic fragmentation, where topics get repeated across several clusters. This is most pronounced when news break and the same story is discussed from different points of view