Introduction

Hypothesis

Conflict on twitter can be measured on a continuous scale of 0 (no conflict) to 1 (total conflict) using the analysis of specialized combination of word embeddings present in this paper. This combination of word embeddings will work as well or better than simply analyzing the word embeddings for each hashtag

Dataset

The dataset that will be used was gathered by [citation]. The dataset consists of 3 main tables: hashtags, tweets, and tweets\_hashtags. The methodology used to gather the data was []. The tweets table contains information about individual tweets, such as the creation date, tweet text, tweet\_id, and the topic that the tweet was relevant to (contained a keyword). The hashtags table assigns a hashtag\_id number to an individual hashtag. Lastly, the tweets\_hashtags table relates tweet\_ids to hashtag\_ids, representing the use of one or more hashtags in a tweet. This gives us a way to relate hashtags to the tweets that they are used in.

Process

This project has three distinct parts: coding, data processing, and result analysis. During the code phase, a code base will be created that facilitates the production of hashtag embeddings. During the data processing phase, multiple result sets are created based on time periods, topics, and other factors. During the results analysis, the effectiveness of hashtag embeddings is compared to other methods of detecting polarization, such as using only word embeddings.

* Code
  + Each of the following steps will be separate command line executables
  + Tokenize + format tweets
    - NLTK tweet tokenize
      * This will need to be adapted to filter more
        + Stop words and symbols
    - Expensive time wise
      * Ideally only do once on all tweets
  + Convert formatted tweets to arrays of integers
  + Generate skip-gram batches for each tweet
  + Input training to neural network
    - Creates word embeddings
  + Create tweet embeddings
    - Sum + normalize word embeddings that belong to each word in a tweet
  + Create hashtag embeddings
    - Sum + normalize tweet embeddings
  + Create hashtag relationships
    - Calculate cosine similarity between every pair of hashtags
  + Create tweet relationships
    - Calculate cosine similarity between every pair of tweets
    - This may be expensive.
* Data Processing
  + Methodologies will include:
  + Hashtags
    - Process with independent topics
      * Over time, location
    - Process issue based topics overall
  + Tweets
    - Process the same way as hashtags
  + Also process words the same
* Results analysis
  + Attempt clustering on graphs
    - Graphs will be weighted
      * Weights will be determined by hashtag embeddings
      * May be other factors, such as sentiment, specificity, volatility
    - Graphs edges
      * May link every point
      * May be determined by simultaneous
    - Clustering algorithms

Theory

Given:

Set of tweets T, which is a table with columns containing tweet\_id, tweet\_text, and a variety of other information about the tweet.

T = {tweet\_id, tweet\_text}

Set of hashtags H, which contains columns of a text hashtag and an integer hashtag\_id for the hashtag.

H = {hashtag, hashtag\_id}

Set of hashtag occurances O, which is a table in which each row contains one tweet\_id and one hashtag\_id, and represents the use of any hashtag in any tweet.

O = {tweet\_id, hashtag\_id}

Data Preparation:

First, we create a new table formatted\_tweets, which has columns tweet\_id and tokens. Tokens consists of an array of strings representing each sequential word in the original tweet text. This step of the process is done in FormattedTweets.py.

F = {tweet\_id, FormatTweet()}

Next, we create a dictionary D of the most common words in the whole set of tokens. This dictionary maps a word in our dictionary to an integer representing it. This step of the process is done in formatted tweets.

In the last step of data preparation, we create a final representation of the tweet text as an array of integers, by substituting each word token in the formatted\_tweets table with the integer it refers to in the dictionary D. This results in a table int\_tweets with columns tweet\_id and int\_arr, which is an array of integers representing the sequence of word tokens in the original tweet.

Embedding creation:

Next we use word to vec to create embeddings for each word by inputting each integerized tweet. This is done using each individual tweet as a sentence in the word2vec model, and skip grams are made accordingly. This is done in Word2Vec.py

First, using stemming and tokenization, each tweet’s text will be formatted into a list of non-whitespace strings. This will allow us to create a mapping of

We can define the set of hashtag

[explain word 2 vec]

Once we obtain word embeddings VW for each word W in the vocabulary, we can attempt to represent the hashtags by accumulating the word embeddings in a specialized way. This summation of word embeddings for each use of a word in a tweet that uses the hashtag. In other words hashtag embedding EH = SUM(W(EJ)) where Ej = SUM(V(EK)). W and V are both weight functions that apply a weight to a given tweet and word embedding respectively. K is a tweet which belongs to the set of tweets in which hashtag H were used, and

Code

The code itself is divided into separate files in order to represent each step in the procedure individually. The code in each of these files interacts with the postgresql database in order to load the input data from the tables, and output the manipulated data.

FormatTweets.py

The first file is FormatTweets.py, which takes as input the database of tweets with columns of text and the tweet id, which is represented as a long integer. The tweet id is used to reference the tweets back to the original, as well as to relate each hashtag used in the tweet. FormatTweets reads in each tweet in the database, and initially tokenizes the text by using NLTK TweetTokenizer, which splits the tweet into individual words, which are referred to as tokens. These tokens are then scanned for stop words, hashtags, mentions, numbers, symbols, and many more character combinations that are especially difficult to quantify.

CreateDictionary.py

This file can be used to create a dictionary based off of the most common words used in the formatted tweets. This allows us to give each word in the dictionary, and to disregard any word not in the dictionary. Ideally, the size of the dictionary would be close to the number of unique words in the language of the data. The Oxford English Dictionary contains approximately 180,000 words, and assuming the stemming used in format words works properly, we can assume that the most words possible will be in that ballpark [1]. However, after running the algorithm on multiple datasets, we may use a dictionary size of 50,000 words and result in a relatively small number of words that are not included in the dictionary.

WordVecToHashVec.py

This file uses the word embeddings to build hashtag embeddings. This is the main novelty of my project, as it incorporates the hypothesized methodology of generating hashtag embeddings. To do so, a list of tweets used with each hashtag is generated. For each word in each tweet,

WordsToTweetVec.py

This file uses the word embeddings created by WordToVec.py to create embeddings to represent a tweet based on the words that the tweet contains. This is done by first finding the embedding for each word that occurs in the tokenized tweet, and then summing the word embeddings by index to make a vector of the same length to represent a tweet embedding. The tweet embedding is then unitized in order to ensure each tweet has equal weight for when they are used for the hashtag embeddings. Additionally, the importance and centrality of a word to the topic of the tweet may be accommodated by multiplying the word by a weight before adding it to the tweet vector. Also, applying the volatility and specificity of words as factors in the weights may offer additional accuracy in the tweet vectors.

TweetVecToHashVec.py

This file uses the tweet embeddings created by WordsToTweetVec.py as an input, as well as the list of hashtags used in every tweet. Then, for each tweet embedding, the embedding is added to a hashtag embedding that is initialized to all 0’s, with the addition occurring in the same way as in WordsToTweetVec.py. This creates a vector which represents the summation of the tweet embeddings. Additionally, prior to adding each tweet embedding to the hashtag embedding, the tweet embeddings may be weighted to account for the relevance of the tweet to the issue.

The main intuition that led to this project is that hashtags are used to represent ideas, but in a special way by allowing users to attach hashtags to their tweets. By using a hashtag, users are providing keywords of a sort to readers, which can somewhat effectively condense the content of the tweet down to a few hashtags. However, hashtags themselves gain their meaning from the tweets they are used in, in addition to the actual name of the hashtag, which is included in the tweet itself. The tweets themselves obviously also derive their meaning from the words that are used in them, so the hashtags can themselves derive meaning from the words that they are use with. This means that any model that can be used to model the relationship between words should be able to be form a relationship between tweets and hashtags as well. In this project, a word2vec representation is used [citation], which allows the creation of word embeddings as multidimensional unit vectors.

In a debated topic, relevant hashtags can represent aspects of both the topic and stances in an issue, while many other hashtags simply act as noise and additional relationships between tweets.

Analysis

Once hashtag embeddings are created, they can be compared to each other. this can be done using cosine similarity, which is 1 - cos(THETA) where THETA is the angle between the embeddings. However, this way of comparing embeddings is very data specific for hashtags with a small number of uses.

A simple way of looking at polarization between two hashtags is to measure the cosine similarity between the two, however, this does not account for the rest of the embeddings as the circumstances for the similarity between the two hashtags. As such, Another idea is to take the two hashtags that are to be compared, and use them as a basis with which to compare the all the other hashtags. If the two hashtags used as an axis are very similar, we would expect the cosine similarities between the axis embeddings and the rest of the hashtags to be near each other. This is supported by the intuition that similarity is somewhat associative. That is, if two hashtags are similar, and the first is similar to a third hashtag, then this implies that the second is also similar to the third. So, every other hashtag is graphed in relation to the two axis embeddings, and and a linear regression is done on them. The slope of the linear regression line represents the polarization between the two

There are a variety of ways to obtain a measure of polarity.

one of these ways is to construct a list of topic relevent hashtags, with an equal number for either party. Find the cosine similarity between each of the hashtags in one camp and each of the hashtags in the other, taking the average similarity of all combinations. However, this methodology is largely dependent upon the hashtags chosen to be relevant, and does not give the full picture. However, it is very simple and may serve as a decent baseline for other methods.

I propose to build a relation between two hashtags by comparing their relations to all the other hashtags in the set. the set of relations to all other hashtags is built for the two hashtags to be compared, and both sets are normalized between -1 and 1. a coordinate is then formed for each hashtag by taking it’s normalized relation from each set, allowing each hashtag to me graphed in two dimensions, which each dimension representing the similarity between one of the two hashtags. A linear regression is then done on all the hashtag points on the graph, giving a slope of the line representing the similarity between the two hashtags used as axiis. The closer the line is to 1, the more similar the hashtags are in relation to the overall relation of hashtags, since any two hashtags that are truly similar in the context of all the hashtags, should have their similarities with all other hashtags be close to equal. for instance, one would predict that taking the cosine similarity between any hashtag and the hashtags ‘#trumptrain’ and ‘#trump2016’ would give similar values. This may produce a better contextualized relation on the scale of -1 to 1.

To obtain the hypothesized one dimensional measure of conflict using the relationships represented as the cosine similarity between embeddings, we must account for a variety of factors. Specifically, the number of ‘noisy’ hashtags that have little relation to the actual topic. Ideally, these hashtags will have close to 0 similarity in relation to topic relevent hashtags.

A K-means clustering of the hashtag embeddings can be done

A temporal examination of the issues will be done, on a weekly basis from the 2016-06-24 to 2016-09-06. The word embeddings will be recreated for each week, along with the hashtag embeddings.

1. <https://en.oxforddictionaries.com/explore/how-many-words-are-there-in-the-english-language>

<http://emnlp2014.org/papers/pdf/EMNLP2014194.pdf>

specificity:

<https://cs.stanford.edu/people/jure/pubs/identity-icwsm17.pdf>

Word2vec

<https://www.tensorflow.org/tutorials/word2vec>

<https://github.com/tensorflow/tensorflow/blob/r1.2/tensorflow/examples/tutorials/word2vec/word2vec_basic.py>