//Author: Sam Watson

//3.15.15

\*\*\*\*\*\*README\*\*\*\*\*\*\*\*\*\*

1. OVERVIEW
   1. What does this program do?

This program implements a novel First Story Detection algorithm for streaming Twitter data. It is inspired by the approach described by Petrovic, Osborne, and Lavrenko in their paper Streaming First Story Detection with Application to Twitter (2010).

* 1. How do I run it?

If you are on windows, navigate to the appropriate directory in the command prompt and execute these lines:

javac -cp .;twitter4j-core-4.0.2.jar FirstStoryDetector.java

java -cp .;twitter4j-core-4.0.2.jar FirstStoryDetector

If you are on unix-like, execute these (this is untested):

javac -cp '.: twitter4j-core-4.0.2.jar ' FirstStoryDetector.java

java -cp '.: twitter4j-core-4.0.2.jar' FirstStoryDetector

FirstStoryDetector contains a main method that will run first story detection on a sample dataset of 500,000 tweets. It should only take a couple of minutes to fully execute.

The program starts by reading a raw data file called cleanTweets\_25\_500000.txt. It splits this file into five smaller files containing 100,000 tweets each. Within each of those files, each word is given an integer ID code. Those tweets are hashed according to a scheme explained below. The console will show updates as tweets are processed. After each file has been processed, the console will output the top “threads” in the file (see section 2.2 for an explanation of threads) by showing the full text of the parent tweet and the IDs of a few of the other tweets in that thread. If you’d like to see the textual content of those tweets, you can search for their IDs in the cleanTweets file.

* 1. What is FSD?

FSD involves finding the first document in a dataset which introduces some new topic of conversation that other, later documents also discuss. In this case, that means identifying the first tweet to break some type of news that spreads across Twitter. This problem is primarily concerned with identifying the first tweet to discuss some popular topic, but an implicit subproblem is identifying popular topics on Twitter.

* 1. How does it work?

This implementation is similar to the one proposed by Petrovic, Osborne, and Lavrenko (2010) in that it uses Locality Sensitive Hashing (LSH) to efficiently estimate the cosine distance between tweets, and in doing so identify sets of tweets that are similar (i.e. discuss the same topic). Similar tweets are grouped into “threads,” and the first tweet to appear in a thread is considered the thread that started the topic. For example, if there is a political scandal then the first tweet to report on the scandal is the considered the “parent” of the thread, and all subsequent tweets discussing the scandal are grouped under it. Threads that grow the fastest can therefore be considered the most popular topics on twitter (at that particular time).

1. THEORETICAL/TECHNICAL EXPLANATION
   1. Locality Sensitive Hashing

LSH is a technique for efficiently finding similar documents in large datasets. This particular implementation uses a cosine distance metric. Rather than calculating the cosine distance between all tweets in the dataset, we want to use a hashing scheme that will assign the same hash code to tweets that are likely similar and then limit the search to those probable nearest neighbors. For cosine distance, this is done by intersecting the highly dimensional space formed by all possible words in the dataset with random hyperplanes. As it turns out, the probability of two points falling into the same subspace formed by these hyperplanes is proportional to their cosine distance. The number of hyperplanes can be considered as the number of bits per key in our hashing scheme. Thus, we can assign each tweet a hash code in the following way:

1. For each point in dataset Ti:
   1. For each hyperplane Hj:
      1. If Ti \* Hj < 0, set the jth bit in the Ti’s hash code to 0, 1 otherwise.

Tweets with the same hash code fall into the same subspace, and therefore belong in the same “bucket” of similar tweets. As the number of hyperplanes increases, there will be fewer tweets in each bucket, and therefore fewer distance calculations to do. However, this also reduces the probability that two most similar tweets land in the same bucket. Therefore, it is necessary to create multiple hash tables, each representing a space intersected by different random hyperplanes so that we increase the chance that two most similar tweets will land in the same bucket in one of the tables. Obviously, this leads to a tradeoff between the number of distance calculations we need to make vs. the number of tables we need to keep track of.

* 1. Identifying Hot Topics/Threading Tweets

As tweets arrive, their nearest neighbor across all the different tables is calculated. If a new tweet N is sufficiently different from its nearest neighbor, then it is unique, and may be the first to break a new story. Thus, we say it is the parent of a new thread. Any tweet that arrives later whose nearest neighbor is N will get placed in N’s thread. Likewise, any incoming tweet whose nearest neighbor is already in N’s thread will also get placed in that thread. Hence, threads will contain tweets discussing similar content, and will grow proportionally to the number of users discussing those topics. We can identify the fastest growing threads by looking at their growth rate within a window of 100,000 tweets, and the first tweet to start those topics is the parent of the thread.

* 1. What is Novel About this Implementation?

Petrovic, Osborne, and Lavrenko do not describe their strategy for generating random hyperplanes. The sources they cite describe it as generating a vector of random coefficients of the appropriate dimensionality. However, this is not appropriate for this application. There is no concept of negativity in textual data; a tweet either contains a word (in which case that coordinate has a value of 1) or it does not (giving that coordinate a value of 0). If coefficients are simply generated randomly, there is only a very small chance that the hyperplane will actually intersect the (relatively) small positive subspace that the points can occupy. Therefore, I developed the following method for generating hyperplanes:

1. Pick two different words w1 and w2 at random proportional to the number of times they appear in the data. These words can be considered coordinates in hyperpace (this is easy since words are assigned integer IDs).
2. Choose one value v1 at random ranging from -1 to 0. Choose another value v2 at random ranging from 0 to 1.
3. Create a vector that has value v1 at coordinate w1 and value v2 and coordinate w2. Set all other values in the vector to 0. This vector can be considered the coefficients of a hyperplane that passes through the origin and the subspace that our points actually occupy.

This approach has several advantages. First, choosing coordinates proportional to the number of times they appear in the data helps to divide the subspace evenly. It helps to delineate our buckets based on words that are common, since more points will tend to fall on different sides of the plane. Second, the hyperplane only has two nonzero coefficients, so we can represent it using a small amount of data and can calculate dot products very quickly since we can ignore all but two components. This speeds up hashing tremendously.

A downside to this approach is that Petrovic, Osborne, and Lavrenko’s formulas for the optimal number of hyperplanes and hash tables no longer apply, so these parameters had to be estimated via experimentation. It is probable that the parameters in the current implementation are not optimal, but they do produce good results.

1. RESULTS
   1. Overall, this implementation is successful. The program is able to identify threads of similar tweets that grow quickly, and can accurately report on the first tweets to start those topics. Here are a few examples of “first stories” that the program found (tweet contents are given in full as they appeared online):
2. The BBC has suspended Top Gear host Jeremy Clarkson "following a fracas" with a producer
3. RT @BlueJays: Marcus Stroman as suffered a torn left ACL and will be out for the 2015 season.
4. RT @JayGlazer: The Saints and Seahawks are trying to finalize trade Jimmy Graham and a 4th rounder for Max Unger and Seattle first rounder

Note that 2) and 3) are not the actual first tweets to break those news items, because they are “retweets.” However, the results are still accurate, because they are the first tweets in this particular dataset to mention that news. The free Twitter API only makes 1% of the total tweets available to stream.

However, the program does not always identify such quality content. Some of the threads that it identifies look more like this:

1. Sold #EURUSD 1.08063 SL 1.08318 TP 1.07995 | Auto-copy FREE & $1000 bonus via http://t.co/fsM3as4Hda #fx #win #news #ipad #android #ff #rt
2. RETWEET

FOLLOW ALL WHO RT

FOLLOWBACK

GAIN WITH ME

RT NOW

#RETWEET #ipad #gameinsight m)xg

1. @KatGraham Follow me please, i love you

Thread 1) is obviously spam. All of the other tweets in that thread are more or less identical. The same is true for thread 2). Thread 3) is not spam, it just turns out that there are lots of people on twitter who like to beg for followers. However, this still shows that the program is working as intended, since it accurately categorized these similar tweets into the same threads, and these were indeed the threads that grew most quickly. It is possible to weed out some of the spam by eliminating threads that have low “entropy” (a measure of content variance within threads), but this is not a perfect metric.

1. CLOSING THOUGHTS

I think this project was a great success and I had a lot of fun with it. I am proud of my method for generating hyperplanes, and find it to be effective. It’s worth noting that the current implementation is not truly “streaming,” since it uses cached data for the sake of simplicity and consistency. However, it would be fairly trivial to modify the program to work alongside Twitter’s streaming API. Additionally, although I succeeded at detecting some first stories, none of them were huge “breaking news” items. I believe this is just because over the scope of my data collection there were no earth-shattering events causing widespread conversations on twitter. Petrovik, Osbrone, and Lavrenko mention that threads about celebrity deaths were the ones that grew fastest, and I’d really like to see how my program responds to a huge (although certainly tragic) event like that.

Source Cited:

Petrović, S., Osborne, M., & Lavrenko, V. (2010, June). Streaming first story detection with application to Twitter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics* (pp. 181-189). Association for Computational Linguistics.