***Motivation/Applications***

In the early years of the Internet, its only purpose was to facilitate the U.S government in attaining a fault-tolerant network of communication through computers. Once the Internet became more accessible to the public with the birth of the World Wide Web in 1989, a new problem developed []. Finding relevant information in this massive web was difficult, especially with regard to obtaining specific information. As the quantity of content on the Internet grew, the complexity of traversing the web grew exponentially. This was the beginning of Web Search Engines, who’s future would be unprecedentedly bountiful.

The basic steps for a web search engine are: Web Crawling, Indexing, and Searching. Web Crawling and Searching are two very complicated and very interesting problems on their own but *Latent Semantic Analysis* (LSA) pertains to Indexing. Specifically, LSA shines in indexing large collections of documents and this is where its versatility comes into play. Not only does LSA utilize the keywords of documents, it also examines the ‘Latent’, or contextual, meaning of words relative to the entire corpus of documents as a whole. In essence, analyzing the meaning of a word in this set of documents is not dependent on word overlap. LSA uses linear algebra techniques to construct a vector space known as *Semantic Space* that provides a mechanism to analyze the similarity of any collection of words.

Searching for specific documents or web pages requires a user to specify a string of words for the search engine to query against its indexed web pages. If the indexing process only takes into consideration keywords, the user’s search could be: “car gas prices”, but the computer would not be able to recognize that a web page containing: “automobile, oil, Saudi-Arabia” might be very relevant to the user’s query. Or for example, the search: “coke” might return some rather explicit content when all the user wanted to was see some information on soda. With this regard, one can see the considerable value of *Latent Semantic Analysis* and its ability to contextually analyze the meaning of words.

The extreme growth of technology and the incredibly large amount of data bing produced every second, with regard to social media, poses an interesting question. Where are people talking about similar things? The morphology of language in social media has made this problem even harder to address because unlike 10th grade English class, there are no rules on Twitter, or Facebook. How can a computer learn new acronyms like “*LOL*” and associate them with “*LMAO*”, or “*laughing?”* This can be accomplished by using the context of the word and LSA does exactly that. An experiment was conducted at the University of Colorado, where two students wrote short essays demonstrating their knowledge of scientific topics, which were evaluated by two “expert human researchers” as well as LSA. “The first principal finding that LSA-based measures—which take no account of word order—were as closely related to human judgments as the latter were to each other.” [1]

Having collected over 200 thousand geo-tagged *tweets*  from the U.S.A. this paper addresses the question: “Where are people *tweeting* about similar topics, specifically in Colorado?”

***Data: Sampling, Filtering, Cleaning***

To accomplish the goal of determining where people are tweeting about similar topics the sample must be chosen properly. Using the public Twitter Streaming API, which samples the public stream of tweets randomly [2] we are off to a good start. LSA does work for all languages (after tokenization of the symbols), however due to the small percentage of tweets the public Twitter API actually samples some filters were set in place to retrieve a certain subset of tweets:

Tweet Filters:

- Geo-Tagged (200k from U.S.A)

- 300 Mile Radius of Denver, CO. (around 2700)

- English only

When the random sample had been generated, the data was transformed into a format that LSA can use. The data for each tweet was contained in a JSON (JavaScript object notation) document, which provides lots of information unnecessary to this analysis. Using Python, each JSON document was parsed for the geo-tag (latitude and longitude), and the text of the tweet. The text of each tweet was very messy. Using the NLTK (natural language tool kit) module for Python I implemented cleaning scripts that search engines utilize to normalize the text. This consisted of lower casing all letters, removing stop words (words that carry no meaning), deleting words starting with ‘@’, ‘#’, removing punctuation, and tokenizing the text by whitespace. Once these steps had been taken I had a clean set of documents (tweets) that could be analyzed using LSA.

The starting point for LSA is constructing a tf-idf (term frequency – inverse document frequency) Matrix :

🡪 for {i = 1, 2, … , m} and {j = 1, 2, … , n}

Where: = ( , ) element of the matrix

= Number of occurrences of term i in document j

= Total # of terms in document j

= Total # of documents in corpus

= Total # of documents containing term-i

Once constructed T is a very sparse, very large matrix. With the random sample of tweets , which means there were 1169 tweets collected within a 300-mile radius of Denver, CO. Now with 2743 unique *tokens* (terms) there is a need to combine terms to create the *Semantic Space* that contains vectors of contextually similar words. In order to get this dimension reduction, first apply the *Singular Value Decomposition* to A.

**SVD of T:**

🡪

Where, are the singular values of A, are the left singular vectors of A, and are the right singular vectors of A. Further are the Eigen-vectors of and are the Eigen-vectors of with being the Eigen-values respectively.

Due to computer accuracy, there are rounding errors and the limitations of machine epsilon prevent The singular values of the tweet data drop off substantially from to = 1.195e-15. Therefore we see there are some linearly dependent columns and we can drop the last 14 singular values and set them equal to 0. Now the condition number of A is the ratio of the largest singular value to the smallest,

🡪 = = 21.245

Now this condition number indicates that A is ill-conditioned and that makes relating documents to one another a very inaccurate system. Namely in order to relate documents to one another, there needs to be a measure of similarity in *Semantic Space* which requires relating document vectors to one another. In order to construct a system to relate documents to one another in a system where the rows are term vectors and the columns are document vectors, we need to calculate:

🡪 , which has condition number 🡪 = = 451.35

Therefore the way the system is now, is very ill-conditioned. Even though the stop words were removed and the text was normalized as much as possible, there is still *noise* in the system creating a large gap between the largest and smallest Eigen-values of

Because we are working with the SVD, we have a trick to reduce this condition number and essentially *de-noise* the system by truncating the SVD of A and then forming a new approximation . To truncate the SVD we delete a certain number of singular values from starting from the smallest, until we get a truncated system that can be recomputed as an approximation A with some acceptable error. From a presentation given by Tim McCandles, who works for Oracle doing LSA and other types of text analytics said k was around 300 for most analytics in industry however in industry they are analyzing 50k tweets a second so the dimensions of A would be much, much larger than this case. Testing with Mr. McCandles suggestion as well as few more the results:

🡪 k = 300 🡪

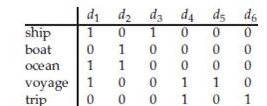
🡪 k = 80 🡪

🡪 k = 40 🡪

🡪 k = 10 🡪

🡪 🡪

This truncated SVD decomposition when multiplied together yields a full rank approximation to A with very little error. However



***Mathematical Methods: Latent Semantic Analysis***

Assumption 1: words that have similar meaning will occur in similar locations of text

Bibliography:

[1]<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.3.6720&rep=rep1&type=pdf>

[2] <https://dev.twitter.com/docs/api/1/get/statuses/sample>