Team WJX: Joel Kemp and Wai Khoo
Instructor: Prof. Jinlin Chen
Web Information Retrieval and Data Mining
Final Project Report

Twitter Hashtag Suggestion System

Testing Data

The system uses an extensive corpus of tweets about a popular Twitter user, Ashton Kutcher. He was chosen solely for his very high number of tweets. In the future, the system could be adapted for any user, but the more active Twitter users supply the system with more data to build a better suggestion model. Using the Twitter API, we were able to retrieve 3200 tweets for use in our system.

The tweets were retrieved in a single file in the webservice-standard, JSON format. For use in our system, the JSON format had to be converted to a plain-text file; this was achieved via regular expression parsing of the file.

The tweets, now organized on a tweet-per-line basis within a plain-text file, had a tremendous amount of noise. Real-world tweets contain noise such as embedded links, Retweets (i.e., rebroadcast of another user's tweet), user handles (for tweets addressed to another user), punctuation, and obscure casing. All of the noisy data was stripped out of each tweet, leaving a near-usable list of words.

The final step in preparing the dataset involved preprocessing the tweets. This entailed removing the stop words and stems from each word in a tweet. An online collection of stop words were collected and stored in a text file. The Porter stemming algorithm was used to remove the stems from non-stop words.

After all of the aforementioned processes, the backlog of tweets were ready for splitting into two sets: training and testing sets. Approximately two-thirds of the tweets were used for training and the remaining third were used for testing purposes.

Algorithm Description

For a query tweet T_q, a hashtag suggestion is achieved by retrieving the dominant term (i.e., most frequently occurring word) of the cluster to which T_q is the most similar. Similarity measurement is achieved by comparing T_q with the K centroids of the Kmeans segmentation of the user's backlog of tweets. In particular, we compute the *tweet distances* between T_q and each centroid.

The tweet distance between two tweets, A and B is the total number of words shared between A and B. For example, if A is "the house is red and large" and B is "ketchup is red," the tweet distance, after stripping out stop words ("the," "is," and "and") and stems (for which there are none in this example) is 1 – since both tweets have the word "red" in common.

The comparison, of T_q to the centroids, which results in the largest tweet distance, implies that T_q is most similar to that centroid. Hence, T_q should belong to the cluster, C, with which it has the largest tweet distance to that of the centroid of C. Once the membership of T_q has been established, the suggested hashtag for T_q is the dominant term of C.

The dominant term of a cluster is the term/word with the largest frequency about all of the words/terms within a cluster's list of tweets.

Clustering algorithm

We decided to use Kmeans as our clustering algorithm because not only is to a popular technique that's easy to implement, but the algorithm reduces our suggestion, search space from N to K, where N is the number of tweets in our corpus and K is the number of clusters. Each cluster contains a dominant term and centroid that are representative of that cluster's population of tweets.

Our Kmeans algorithm randomly samples *K* items in a population, which in our case is tweets, and initializes it as *K* centroids. Then for each item, we compute a *twitter distance* (or similarity) metric to each cluster's centroid and then assign that item to the cluster with the largest distance (i.e., the most similar). The notion of having a larger distance signifies that the two tweets share a lot of words in common, which intuitively should belong in the same cluster.

At the end of a Kmeans iteration, the clusters are updated with a new list of tweets belonging to that cluster and its centroids are re-computed along with its centroid deviations.

The termination criteria that we used are convergence and maximum number of iterations. Our convergence is defined as the maximum of centroid deviations that falls below a threshold, which is supplied by the user.

Every time a cluster updates itself, it also re-computes the term frequency, dominant term, and its centroid. In the following, we will describe how each is computed. The concept is very similar to the classic Kmeans; however, we made some modifications, appropriate for our dataset.

The term frequency of a cluster is simply a dictionary of terms (words from the cluster's list of tweets) and word/term frequency over the set of tweets in the cluster. Once we have the list of frequencies, the dominant term is simply the term with the largest frequency.

In the classic Kmeans, the centroid is computed by averaging all the data points in 2D space. However, we are not dealing with 2D data points. Therefore, we simply just compute the *twitter distances* of each tweet in the cluster to its centroid, and finding the average by summing up the distances and divide by the number of tweets in the cluster. To designate a tweet as a centroid, we took the absolute difference of *twitter distances* from the average and finding the smallest value. Once the new centroid has been designated, the cluster deviation is simply the normalized *twitter distance* between the new centroid and the old centroid.

Evaluation Result

The system is able to suggest a hashtag for a user's tweet input – where the suggestion model is based on Ashton Kutcher's tweets. The system also provides the hashtag suggestions for the testing set of tweets.

The results are not as promising as we expected. The majority of tweets result in the same suggested hashtag. This is primarily due to both the corpus' distribution of words and the features that were chosen for the segmentation algorithm.

Our algorithm attempted to define tweet similarity by the number of matching words. In fact, this may have been too firm of an assumption. We hypothesized that there would be high number of word repetitions about 3200, 140-character messages. Through our experiments, it seems as though the repetitions came from stop words. After preprocessing (i.e., the removal of stop words and stems), we were left with a mostly disparate dataset of words.

The disparate set severely handicaps the segmentation process since many tweets will have tweet distances of zero (i.e., no words in common) compared to the centroids. Hence, these tweets will be left with no matching clusters. We attempted several solutions for how to segment those unmatched tweets.

One simple solution was to just assign them to the first cluster, assuming that the number of unmatched tweets was negligible. Of course, through our analysis, this resulted in a very large first cluster due to a significant number of tweets that were unmatched.

Our revised attempt uniformly distributed the unmatched tweets about the set of clusters. This resulted in a more even distribution of tweets about the clusters. However, even with a more balanced population about the clusters, the lack of exactly matching words between tweets affected tweet suggestion.

For the suggestion process, we compared T_q with the centroids of the clusters. The centroids are actual tweets. Again, due to the disparate nature of the dataset, a (preprocessed) query tweet is unlikely to have matching words with any of the centroids. This results in the default hashtag being the dominant term of the first cluster. In other words, since the distance between T_q and each centroid is zero, there is no closest fitting cluster. Ultimately, either the first cluster is chosen as a match, or we randomly choose. However, both options are horrible hashtag guesses.

An improvement to the system could be to introduce synonyms of words during the tweet distance computation to allow for a matching of "similar" words – not only exact matches. In theory, this would result be a better heuristic for distributing the tweets about the clusters, as we would reduce the number of unmatched tweets. In the suggestion phase, the synonym comparison for tweet distance would also increase the likelihood that T_q would have a matching cluster.

Another experiment that could improve the results would be to use a Markov Modeling of the tweets. We can imagine that tweets that are close together (in regards to time) are likely to have similar content. We can then use the current time of year (or day for more granularity) to guess the content that is important to the user. This could be one feature used in a voting process, where

several features allude to a choice of hashtag. The most likely match would then be served to the user.

Another experiment could be to fluctuate the values of K in Kmeans. Through our tests, there was no evidence that a particular value of K resulted in the best distribution of tweets that would increase the likelihood of fitting a T_q to a cluster. There also doesn't seem to be a way to deduce the best number of clusters since we have no hashtag ground truth.

One way to possibly establish the ground truth is to locate existing hashtags from the raw (unprocessed) corpus and use the hashtags to serve as a way to build a confidence model. In fact, this method could be extended beyond a user's corpus to say that given a collection of words (tweet), a user tagged that collection with the hashtag, *H*. This could then influence our choice of hashtag within the suggestion engine.

There are many areas for improvement in the system. The difficulties in hashtag suggestion are isomorphic to the challenges of NLP-ists trying to find the distance between two sentences. Despite the poor results of our first trial run, we look forward to finding more promising solutions.

Team Division of Workload

The following breakdown corresponds to the module design of the code. The code-base was so that each team-member could own particular high-level portions of the system. For example, for a topic like preprocessing, you can view preprocess.py to see all of the work done by the author.

Please note that the source code also has authorship noted in each file to indicate the distribution of work.

The source code can also be found on our github repository:
https://github.com/mrjoelkemp/hashsuggest

Data Preparation	– Joel Kemp
Preprocessing	– Joel Kemp
Segmentation	– Wai Khoo
Feature Extraction	– Wai Khoo
Suggestion	– Joel Kemp
Front-end	– Wai Khoo
Statistics	– Joel Kemp
Report	– Joel Kemp and Wai Khoo