Facebook Status Generator (FSG)

Vu Le, Heidi Chen, Nico Ekasumara

November 20, 2013

1 Introduction

Natural languages are inherently difficult and cryptic to analyze algorithmically. That said, it can be very useful to identify phrases and sentences that people notice and like. This project involves both understanding how languages are used in daily lives and finding catchy phrases. In particular, FSG creates random Facebook statuses that aims at maximizing the number of likes. We collect user's past Facebook status messages and utilize AI algorithm to learn his/her way of writing that has historically generated likes among his/her specific friend group. We adopt the traditional text processing approaches such as 'bag of words', stem words, inter-quartile range filter. We pay careful attention to spelling mistakes, synonyms and acronyms. In order to determine the words that generate many likes when appearing together, we adopt a clustering algorithm with the words as nodes and the average number of likes as edges. Hence, the densest and heaviest cluster will point toward the group of words that can form catchy phrases together. We then input the heaviest cluster to a sentence generator to form random Facebook statuses. Similar ideas have been recently adopted with different methodologies and received much popularity (without us knowing before starting this project); for exmaple, that by the Princeton hackathon group, `http://www.telegraph.co.uk/technology/facebook/10449279/What-would-I-say-app-which-regurgitates-old-Facebook-status-updates-goes-viral.html`. We believe that this is an interesting, fun, and enriching project that will benefits every member.

Keywords: Facebook, Clustering, AI, Text Processing, Python, Social Network

2 Plan

We will use Python for coding and github for version control and code sharing. The git repo for our project is `git@github.com:AlexVuLe/FB_STATUS_GEN.git`. The components of the project are as followed:

2.1 Facebook API

The Facebook script needs to interact with Facebook's Graph API to support the following functions: i) user authentication, ii) retrieval of historical Facebook statuses and their associated numbers of likes in a clean format, iii) posting select generated statuses to user's Facebook page. All of these functions can be done using the facebook-sdk module, `https://github.com/pythonforfacebook/facebook-sdk`. The Graph API allows user access to his/her own Facebook objects, namely, 'post', 'status message', 'note', etc,.... together with their numbers of likes and other related information. The details regarding these objects can be found at `https://developers.facebook.com/docs/reference/api/post/`. We target to only use status messages. However, status messages are usually short and have poor contents, which may not give us enough data. In such case, we may use posts or notes. To protect user's privacy, all data will be deleted once the script completes. For testing, we will provide a clean set of anonimized data.

2.2 Text Processing

We use package NLTK to perform the following tasks on Facebook text data to collect desired words in a clean and informative format. We remove stop words. Conventional practices involve removing the most frequent words in the top percentiles. Thus, we first run through the data and store a list of stop

words. We will also take a general list of stop words online and combine the two lists. We also remove infrequent words at the bottom percentiles since they are probably not important. We then reduce words to their stems; for instance, 'flied' will be reduced to 'fly'. We also check for spelling mistakes. All of these can be handled by NLTK. We consider grouping synonymns but are unlikely to pursue this feature because: some synonyms can receive more likes than others; whether two words are synonyms depend on the context. Another issue with text processing is to understand acronyms which we are yet able to resovle.

2.3 Clustering Algorithm

Once we have the set of word stems we are working with, we would like to find out which words tend to generate more likes, and also which words together in a status tend to gain popularity. We will first implement a clustering algorithm to extract the most popular group of words out of the entire set of word stems.

To set up for clustering, we need to construct a network from the word stems in the complete set. We will connect two word stems with an edge if they have appeared in the same status, and we will weight an edge $e_{w_1 w_2}$ based on the average number of likes we observe from the statuses which contain the two word stems w_1 and w_2.

We will be implementing spectral clustering in this project because it is invariant to the shape and density of the clustered network. The algorithm was proposed by Jianbo Shi and Jitendra Malik (Shi and Malik, 2000) and explained in detail by Ulrike von Luxburg (Luxburg, 2007). Below is a summary:

1. Construct an adjacency matrix based on the weights described above.
2. Compute the unnormalized Laplacian matrix L.
3. For clustering into k cluster, compute the first k eigenvectors $\mathbf{u_1}, \cdots, \mathbf{u_k}$ of the generalized eigenproblem $Lu = \lambda Du$.
4. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $\mathbf{u_1}, \cdots, \mathbf{u_k}$ as columns.
5. For $i = 1, \cdots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U.
6. Cluster the points $(y_i)_i = 1, ..., n$ in $\mathbb{R}^k$ with the k-means algorithm into clusters $C_1, ..., C_k$.

Depending on what the graph actually looks like, we may also consider clustering methods using modularity matrix or k-nearest neighbors similarity matrix as alternatives. Eventually, we would like to cluster from the original graph the densiest subgraph containing the nodes that seem to generate more popular statuses.

2.4 Sentence Generator

The package NLTK provides built-in functions to generate texts and sentences based on some seed documents. It also allows us to customize sentences by inducing a grammatical structure. We aim to just understand the process used by NLTK without having to manually implement the algorithms. That said, if time permits, it will be interesting to build our own Markov-Chain sentence generator.

2.5 Potential Extensions

- If time permits, we will consider introducing additional features such as tags (of people), hashtags, etc.
- We can also implement the analysis for the three project collaborators' personal statuses. We can see how the "more popular" clusters overlap and differ.
- If time permits, we will consider the effect of posting time on the popularity of a post.
- We are concerned that some **pairs** of words may appear so rarely that the average estimate will be inappropriate. We consider removing these edges.

3 Schedule

- Week ending 11/24: Vu completes Facebook API. Nico and Heidi write up and discuss details of text-processing and clustering algorithm respectively.
- Week ending 12/1: Heidi and Nico start implementation. Vu implements sentence generator and provides assistance. Thanks Giving.
- Week ending 12/8: Complete all implementations by the end of Wednesday. Visualize clusters and clean up codes on Thursday. Prepare slides and send to Vu by the end of Friday. Practice presentation Saturday and Sunday evening.
- 12/9 Code due. Presentation and demo
- 12/13 Final paper due

4 References

U. von Luxburg, "A tutorial on spectral clustering", Stat. Comp. Vol. 17, Issue 4, 395-416 (2007), Papercore summary http://papercore.org/vonLuxburg2007.

Jianbo Shi and Jitendra Malik, "Normalized Cuts and Image Segmentation", IEEE Transactions on PAMI, Vol. 22, No. 8, Aug 2000.

Sholom M. Weiss, "Automated Learning of Decision Rules for Text Categorization", ACM Transactions on Information Systems, Volume 12 Issue 3.