Twitch Chat Analysis

The main goal of the project is to analyze the online social network by using machine learning and we aim at Twitch Chat Room. Twitch is the platform for live streaming. Any users can stream almost what every they want such as playing games, cooking, programming, art, talking show…etc. And one important thing is how you get more viewers. To get better performance and have a good interaction with your viewers, I build a automatic chat analyzer combined with ToPIN to summarize a streamer's performance. I picked four different fields which are content, emotion, relation and topic respectively to analyze different aspects of performance. Before starting, we talk about the common issues of chatting system. Most of time, the twitch chat messages suffer from being short and sparse, and also people may use abbreviated words, slangs, or multiple languages when chatting. Besides, there may some noises like misspelling words, repeated letters in word or composition words…,etc. For simplicity, I only analyze english-based texts, and use preprocessor to tackle some of the problems above to reduce noise signals as much as possible. the preprocessor is being applied with customized tokenizer when converting the raw data that we collected from "chatty" (twitch chat logging software) into tokenized data set. customized tokenizer that can separate the message into the pre-defined token form.

- What does the customized tokenizer do?
  (1) separate the utterances into the pre-defined token form
  (2) remove the repeated letters. e.g. "crrrrrrrrazyyyyyy" => 'crazy'
  (3) remove english's stop words which is meaning word in the sentence such as 'the', 'a',…etc.
  (4) remove punctuations
  (5) be able to recognize the properties of separated sentence. The properties are "normal" , "url", "emoticon", "hashtag", "word with one length", or "digits only".
  (6) lemmatize and produce part-of-speech (POS) tag for each token by using NLTK

Those tokens that contain "normal" property is what we want to keep as the cleaned data. Those cleaned data will be served as an training data in topic analysis. In topic analysis, we focus on finding the topic distributions among the corpus (cleaned data set). Because we don't have pre-labeled data, I chose Biterm Topic Modeling (BTM) which is unsupervised machine learning algorithm for clustering the data into topics. In conventional topic model like Latent Dirichlet Allocation (LDA), it learns topics based on word co-occurrence patterns over the document-level which is highly influenced when your corpus are short and sparse. However, BTM directly models word co-occurrence patterns based on biterms. A "biterm" denotes an unordered word-pair co-occurring in a short context (i.e. any two distinct words in an utterance). That says BTM not only has a better performance than conventional topic model (e.g.), but also is a good way to deal with the sparsity problem. Besides, in order to enhance the accuracy of topics distributions, BTM explicitly models the global word co-occurrence patterns over corpus-level. More specifically, BTM considers the whole corpus as a mixture of topics, where each biterm is drawn from a specific topic independently. The algorithm is as follows (referenced from: "*A Biterm Topic Model for Short Texts*" Xiaohui Yan, Jiafeng Guo, Yanyan Lan, Xueqi Cheng):

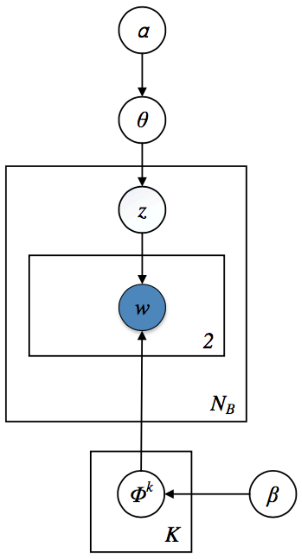
1. For each topic z
   (a) draw a topic-specific word distribution $\boldsymbol{\phi}_z \sim \text{Dir}(\boldsymbol{\beta})$
2. Draw a topic distribution $\boldsymbol{\theta} \sim \text{Dir}(\boldsymbol{\alpha})$ for the whole collection
3. For each biterm b in the biterm set B
   (a) draw a topic assignment $z \sim \text{Multi}(\boldsymbol{\theta})$
   (b) draw two words: $w_i, w_j \sim \text{Mulit}(\boldsymbol{\phi}_z)$

Following the above procedure, the joint probability of a biterm $b = (w_i, w_j)$ can be written as:

$$\begin{aligned} P(b) &= \sum_z P(z)P(w_i|z)P(w_j|z). \\ &= \sum_z \theta_z \phi_{i|z} \phi_{j|z} \end{aligned} \qquad (1)$$

Thus the likelihood of the whole corpus is:

$$P(B) = \prod_{(i,j)} \sum_z \theta_z \phi_{i|z} \phi_{j|z} \qquad (2)$$



In the above process,

(1) each topic z is associated with a multinomial distribution ϕ_z over a vocabulary of unique words drawn from a Dirichlet prior Dir(β)
(2) the whole corpus is associated with a distributions θ over K topics drawn from a Dirichlet prior Dir(α).
(3) To infer the parameters ϕ and θ, BTM adopts collapsed Gibbs sampling algorithm (Griffiths and Steyvers, 2004).
(4) For the corpus which consists of N biterms B = {b_1, ..., b_N}, where b_i = (w_{i_1}, w_{i_2}), we draw the topic assignment z for each utterance from a corpus-level topic distribution θ.
(5) Meanwhile, to surmount the sparsity problem, it also breaks utterances into biterms. BTM not only can keep the correlation between words, but also can capture multiple topic gradients in a utterance, since the topic assignments of different biterms in a utterance are independent.

Inferring topic for each utterance:

Because BTM uses corpus-level topic distributions, we cannot directly obtain the topic proportions of utterances during the topic learning process. To infer the topics in a utterance, we assume that the topic proportions of a utterance equals to the expectation of the topic proportions of biterms generated from the utterance. the algorithm is on the right, where $P(z) = \theta_z$, and $P(w_i|z) = \phi_{i|z}$ and $n_d(b)$ is the frequency of the biterm b in the utterance d.

$$P(z|d) = \sum_b P(z|b)P(b|d). \tag{3}$$

In Eq.(3), $P(z|b)$ can be calculated via Bayes' formula based on the parameters estimated in BTM:

$$P(z|b) = \frac{P(z)P(w_i|z)P(w_j|z)}{\sum_z P(z)P(w_i|z)P(w_j|z)},$$

$$P(b|d) = \frac{n_d(b)}{\sum_b n_d(b)},$$

Next, in the semantic analysis, to detect the positiveness, negativeness or neutral for each utterance, couple of pre-labeled dictionaries are being used. They are dictionary files consists of the words which are labeled by “positive”, “negative”, “inverter”, “incrementer” or “decrementer”. I use those files to add new tag to each expression which contains one or more than one word, and expressions follow the rule “longest match first”. Then, I iterate every token of the expression to accumulate the total score. The positive value, zero or negative value represent the inclination of positiveness, neutral and negativeness respectively.

In content analysis, I classify the twitch chat corpus into six different types of categories which are subscriber only, emote only, bot and command, questions, normal conversations and keyword-based utterances.

In relation analysis, I calculate the total frequency of each token in the utterance to see how much weight of words in the utterance over the whole corpus. The relation value of utterance d is defined as $V(d) = \Sigma C(t|d) / \Sigma_t(C(t))$. The higher score means that the words in that utterance are more discussed or talked in the chat room.

After finishing all the analysis steps, we generate a output folder which include “topics.txt”, “comments.txt”, “usernames.txt” and “analysis.csv” in the streamer folder. Those files will be served as input data for ToPIN.

Code is on GitHub: https://github.com/feelsbadman/TwitchChatAnalysis