Loom2 Profiles

Scott A. Golder

golder@media.mit.edu

August 24, 2001

Introduction

This paper details the structure and use of profiles of individuals in Loom2 and explains the design decisions made while creating the software. I have taken care to include not only careful structural description of the project, but also to provide an understanding of why each decision was made and the implications each has on the project.

When reading a newsgroup for the first time, it is difficult to tell who the key players are. Steve Whittaker (The Dynamics of Mass Interction) notes that usenet newsgroups have what he calls "participation inequality." That is, a small number of users are responsible for a disproportionately large number of postings. This presents two problems: First, which of the large number of people are the important ones? Learning the social hierarchy of the group takes a significant amount of time. Secondly, all the user knows about high-volume posters is that they post with relative frequency. Nothing is learned about the quality of the poster or posts. Knowing how particpants react to one another is something that may take a very long time to learn simply through reading, but having a basic understanding of each user's reputation immediately may be helpful.

My goal in working on the profile project was to explore which characteristics of usenet communication give the user an understanding of who the people inhabiting the space really are. By design, the profile software seeks to emphasize unusual users - people who stand out for some reason or another, usually a very high or low value for some criterion. It is often the case that the personalities encountered on usenet add as much to the user experience - if not more - than the informational content. It is these personalities that the profile project seeks to emphasize.

Data

Data Source

Broadly speaking, the profile software retrieves data from the Oracle server on concord, compiles statistical information about the users, newsgroups and posts, and outputs a tagged file that can be read in and used in a visualization module in a variety of ways.

The data set in the Oracle database has been placed there by Dan Ramage's perl scripts. There are two tables. The first, LOOM2_MSGS contains body text of postings. The second table, created for use with the profiles project, is called LOOM2_MSGS_PARSED, and contains header information for postings, with the corresponding body text placed in the first table. The test data set contains 200,000 posts, covering 10,000 users in 10 groups.

Data Features

The profile project uses seven features to characterize individuals. These seven characteristics were chosen as an attempt to not only capture interesting features about the users, but also to gain an understanding of where the individual fits in the social communities they frequent. These seven characteristics are calculated for each user for each group in which they post, for each user overall, and for each group. Thus, each user's profile contains a set of overall ratings as well as a collection of partial profiles that contain the characteristics for that user in a given group. However:

- For the partial profiles, each characteristic is a percentage of the average of all the users in that group. For example, if a group averages 40 posts per user, and a given user posts only 30 times, he or she will have an activity rating of 0.75.
- For the overall profile, each characteristic is a weighted average of that characteristic in each of the partial profiles. For example, if a user has a wordiness rating of 1.3 in a group in which he posts 30 times and a 0.7 wordiness rating in a group in which he posts twice, his overall wordiness rating will be ((0.7 * 2) + (1.3 * 30)) / (2 + 30) = 1.2625.
- For the group profile, each characteristic is an average of all the users in that group. For example, if a group has 20 posts that start new threads, and a total of 400 posts, the (average) thread start rating for the group is (20 / 400) = 0.05.

With this construction, a user is compared only with those who post in the groups as he or she does. It is important to view a user in his or her own context; a person who says very angry things in alt.flame is very different from a person who says very angry things in alt.callahans, because two groups are distinct communities and may well have very different standards for acceptable behavior.

The seven characteristics used are as follows: activity level, thread starts, responses per post, orphans, wordiness, anger level and netspeak use.

Activity is a measure of how frequently a user posts to a group in relation to the other members of the group. Users who post relatively frequently are more likely to be deeply invested in the social community of the newsgroup.

Thread Starts is a measure of how many threads this user starts. A thread starter is defined as a post that has no parent post and has one or more followup posts. A person who starts a disproportionately high number of threads may be more connected to the community and may be a more outgoing person.

Responses Per Post is a calculation of the average number of responses to each post. A post that has many responses is, for better or worse, likely to be a more interesting post. A user who has many responses per post may be, on average, a more interesting person.

Orphan rating measures failure to start a thread. An orphan is a post that has neither a parent post nor any followup posts. A user with a disproportionate number of orphan posts is relatively disconnected from the community. That is, he or she tries to initiate a conversation, but is getting no feedback.

Wordiness is a measure of the average length of posts. A poster with a high wordiness rating is more verbose than the people with whom he or she communicates.

Anger is a measure of how angry a poster is, both toward other posters and to ideas in general. This rating is not a rating of how many angry posts a poster had written, but rather a measure of how many angry features are seen in the posts. Since the profiles project is more concerned with posters than posts, counting features seems to capture the circumstances better than categorizing entire posts.

Netspeak is a rating of how many netspeak-like features are found in a user's posts.

Data Choice

For the user profiles and partial profiles, each value represents that user's percentage from the mean for that group. Consequently, a user who has a value of 1 is average. The vast majority of users may well be below average, since in many groups, a small number of users often creates a disproportionately large amount of content. The "average user" will be the one whose criterion values are approaching 1, but the "typical user" may not. For this reason, it may be beneficial at some point to generate profiles based on percentage of the median value.

Percentile is intentionally avoided here, because important information is lost, especially concerning the posters with the highest value for a particular characteristic. Suppose the two most active posters in a newsgroup have posted 300 and 900 times, respectively. Statistically, both would be in the 99th percentile. What would be lost, however, is the fact that the most active poster wrote three times as many posts as the second poster did.

Profile Generation

The Process

The following is a detailed analysis of the source code of the profile generation software.

The classes in the profile.generation package do all the work in generating the profiles. The profile generation process begins by a call to Profile.generate().

The generate() method connects to the Oracle database on concord and builds a list of all the email addresses appearing at least once in the table of parsed messages (LOOM2_MSGS_PARSED). For each name, a profile is generated and saved in a tagged text file called PROFILES.XML. After all the user profiles are generated, a profiles for each group is saved in the same manner in GROUPS.XML. Both of these files are in CURR_DIR\backslashXML and total under five megabytes for 10,000 users.

Source code exists (in method Profile.profilesOracle()) to parse the XML file and place all the profile data in table LOOM2_PROFILES. However, reading from the XML file at runtime is sufficiently fast for now, so no code exists to read the profiles back from Oracle.

Creating Profiles

The Profile constructor is called with an email address. The oracle database is searched for all groups in which the given email address has posted. For each group, a partial profile for that user is created. This process is detailed below. Once the partial profiles are created, the overall profile information for the user is generated. This is the end of the process for generating a single profile.

The partial profile (ProPartProfile) constructor is called with an email address and a reference to a newsgroup profile (see Part c. Analyzing Posts). From the newsgroup, the partial profile retrieves references to each post by the given email address. The partial profile calculates the partial profile's statistics with these posts.

Since it can take hours to analyze all the posts in a group to create its profile, when the Profile constructor creates a newsgroup profile, a reference to the profile is kept in a static vector in the Profile class. Whenever a Profile needs to pass a newsgroup reference to the partial profile constructor, it first checks the vector to see if a newsgroup profile already exists. If not, it generates one.

The group profile (ProGroup) constructor is called with a newsgroup name. The constructor queries Oracle and retrieves all newsgroup posts for that group. The statistics for each criterion are then calculated.

Analyzing Posts

As each post is retrieved from the Oracle database, a ProPost object is created for it and any quoted material is removed, as well as the signature. This object stores the email address associated with the post, a count of the words, lines, anger and netspeak features, number of children, and parent status (whether the post has one or not). The email address is stored as a character array, not as a native Java String object. Since 200,000 posts are held in memory at a time, the memory saved by not using Strings is considerable (see Previous Methods below).

Quoted material is defined as any line starting with “)" or ":". Also, any line of text that comes immediately before commented material is removed if it contains the word "said," "wrote," "writes" or "message." This is an attempt to remove lines like "On 13 August 2001, Joe User said:" which introduce quoted material. See the method ProPost.noFollowup().

Signatures are removed by removing all text after the last instance of sigdashes ("-") that has text following it. This process also handles forwarded messages and replies above quoted material. See the method ProPost.noSig().

Each post has a count of netspeak features and anger features (see the FlameParser section below). Netspeak features are defined as any element that appears in the netspeak lexicon (See the Lexicons section below) as well as net markup (starting and ending with asterisks, underscores, etc.) The anger rating is described in depth later in this paper.

Previous Methods

After a reasonably large data set from which to work was built, it became clear that analyzing the data in real time would not work. The first approach did all post analysis in real time. As the volume of posts, groups and users grew, this became infeasible, as the process took far too much time.

The next step was to pull all the post data from oracle, do minimal computation, and save only the post text and important metadata to disk. Profiles were still generated on the fly, but the process now ran in a fraction of the time. The problem with this approach, which became apparent rather quickly, was that saving the data to disk required a very large amount of disk space. Having over 200 megabytes of tagged text severely diminished the portability and utility of the program. The speed increase of this approach was not great enough to offset the limitations it imposed. Additionally, though the process was orders of magnitude faster than before, the addition of new features (e.g. the lexicons and flame parser) and more data again slowed the generation process to an unacceptable speed.

The third and final approach was to split the project into two components, one for profile generation and one that creates an interface for the profiles and the application using them. The generation process does all the post analysis, saving only the profile data. This approach saves storage space, as only a handful of floating point numbers are saved per user, and no text is saved. The generation process, now taking up to fourteen hours for 200,000 posts and 10,000 users, outputs a relatively small (4MB) tagged text file that the interface classes can parse in about 15 seconds.

Lexicons

Lexicon Structure

Each lexicon is a text file containing tagged entries. Each entry carries (e)(/e)tags and contains a word, parts of speech, a frequency count, and a definition. The tags used are (w)(/w), (p)(/p), (f)(/f)and (d)(/d), respectively. Each dictionary is sorted alphabetically and must be alphabetical to function properly.

The (w)(/w)tags enclose the word this is the actual dictionary entry. This is always lowercase. Since each lexicon file must be sorted, and non-alpha characters make different sorting methods behave differently, each lexicon has had all non-alpha spaces removed. So, the lexicon contains "youre" for "you\'re" and "cant" for "can\'t" and so on. This does not affect dictionary lookups, as the has() method compensates by removing apostrophes from its argument.

The (p)(/p)tags enclose the part of speech, which is an integer value. A 16-bit bitset is used to store non-exclusive parts of speech. The parts of speech are: noun, plural noun, noun phrase, verb, transitive verb, intransitive verb, adjective, adverb, conjunction, preposition, interjection, pronoun, definite article, indefinite article and nominative. These parts of speech come directly from the Moby project (see Choosing a Lexicon below) and each have one bit position starting at the least significant. So, nouns have the value 1 and nominatives have 16384. For example, the Moby entry for "boy" has a part of speech of 1025, since it can be a noun (1) and an interjection (1024).

The (f)(/f)tag is called frequency, since the lexicon structure was designed with a word-frequency lexicon in mind (see Choosing a Lexicon). However, it is simply an integer value that can be used for a special purpose if desired. For example, the bad words lexicon uses this field as a measure of anger level. At some point, I would like to rename this field "extra" or "custom" or something similar, since that is really what it is used for.

The (d)(/d)tag is designed to contain a definition in a text string. There is currently no lexicon that makes use of this feature, but the infrastructure is in place.

Each lexicon is stored in memory as a binary tree of Entry objects. When the data is read in from a text file, the resulting binary tree is balanced for optimal lookup time. The lexicons are created from raw lexicon data in a variety of formats. For this reason, there exist in the Lexicon class methods for each of a few raw data files to convert them to lexicon files.

Functionality exists to add and lookup words, read and write lexicons from and to disk, to join two lexicons, and to do advanced text processing (see below).

Text Processing

Of particular interest is the rootWord() method in the Lexicon class. Given a word and a dictionary, this method will return a best-guess root word for this word. For example, given "happiness," one will get "happy" and so on. This is achieved by repetitive pattern matching of prefixes and suffixes and the changes that accompany them. For example, rootWord() will try "fe" and "f" as a replacement for "-ves," thereby accounting for words like "knife" and "wolf." Whenever a pattern is successfully matched, the changed word is added to the word list and all patterns are attempted again. This continues until all patterns are applied but no changes are made. The list of words is searched for the shortest word that appears in the dictionary. The intuition here is that the shortest word will be the one with the most affixes removed.

There are obvious problems with this method. For example, using the *moby* dictionary, finding the root word of "happiness" returns "hap." This is because "happy" is processed the same way "nutty" might be. One way to minimize the errors is to use a smaller dictionary, one that contains fewer obscure words, when looking for root words. Since people have less exposure to obscure words, they are less likely to have great facility in using them and are therefore less likely to apply morphological rules to them in everyday speech.

Lexical Data

Three specialized lexicons have been created for use with the profile generation software. These lexicons contain "bad" words, "netspeak," and acronyms.

badwords.lex

The badwords lexicon contains a range of words from simple insults to profanity, sexual and excretory terms, and ethnic and sexual slurs. Each word is classified based on the (purely subjective) strength of the term used. For example, "stupid" is less strong than "dumbass." Classification also takes into account the quality of being unmistakable as an insult. For example, "shit" can be used as a strong interjection, but "shithead" is unmistakably an insult. Each word is classified as a 1, 2 or 3. This value is stored in the frequency field, and is based on the following criteria: Simple insults (e.g. "foolish," "hypocrite") receive a 1. Stronger insults, as well as basic "swear words" are 2's. Though individual swear words don't necessarily indicate flaming (this lexicon's main application), these words may appear as rootwords in more creative incarnations, ones the flame parser (see below) should not miss. Finally, words classified as a 3 include unmistakable strong insults using sexual and excretory vulgarities, and racist and ethnic slurs.

The part of speech and definition fields are not used in this lexicon.

The creation of this lexicon was done purely by hand and the classifications are subjective. However, the classification system was inspired by David Kaufer's work on flaming in which he used high- and low-flame dictionaries. See the References section.

netspeak.lex

The netspeak dictionary contains many features of electronic speech, including acronyms and various emoticons. This dictionary is used in calculating the netspeak rating for users. The part of speech field is empty in this lexicon, as well as frequency. The emoticons' definition field often has a description, but the net acronyms have no definition.

acronym.lex

This lexicon contains acronyms that are not netspeak. The part of speech, frequency and definition fields are not used. Currently, this lexicon is used only during classification of "yelling." All caps words that are in the acronym lexicon are not considered yelling, since acronyms are often capitalized.

Choosing a Lexicon

Finding the appropriate dictionary was a complicated task. Necessary conditions for being appropriate involved being:

- *comprehensive, but not too comprehensive*. The ideal dictionary would contain virtually all words used in everyday speech, but omit obscure words.
- *easily parseable*. Converting a plaintext dictionary into the tagged format used by the Lexicon class must be a well-defined process. Tagging and other unambiguous delimiting methods are a must.
- *flexible regarding polysemy*. Since words frequently have multiple meanings, dictionaries often have multiple entries with the same headword. It is undesireable to either have only one part of speech allowed per word or to have multiple entries for headwords.

Three dictionaries were considered for use in the profiles project: BNC, Opted and Moby. Each was considered and, in all but one case, dismissed, for the reasons detailed below.

BNC Database

The BNC database is a word frequency list derived from the British National Corpus (BNC) compiled by Adam Kilgarriff. It is located at *http://www.itri.brighton.ac.uk/ Adam.Kilgarriff/bnc-readme.html#lemmatized*. The BNC is a collection of texts of both written and spoken British English. The lemmatized lists by Kilgarriff contain the 6,300 most frequently used words in the collection. The list is available sorted either alphabetically or by frequency.

The original intent was to use frequency lists from CELEX, the Dutch Centre for Lexical Information, a common linguistic database source. However, since CELEX had recently discontinued offering support and data due to lack of funding, other sources of data became necessary. A related web site pointed to these BNC lists and so they came to be used in this project.

The alphabetically sorted list was used, since alphabetical order is necessary for constructing the lexicon files. This database was effective as a first step, since it was easily parseable, but was clearly unsuitable for this project since it is very small and is not flexible with respect to parts of speech.

As a side note, it is while using this database that the Lexicon tagging format was designed, this the frequency field.

OPTED

The Online Plain-Text English Dictionary (OPTED) is a "public domain English word list dictionary based on the public domain portion of (Project Gutenberg's Webster's dictionary)."It is available at *http://msowww.anu.edu.au/ ralph/OPTED/*. This dictionary is very similar in content to Moby (see next), but lacks some important features. First, OPTED is published as a collection of large HTML files. Even after being combined into a single large file, OPTED was still not easy to parse, since it is HTML and therefore designed for viewing by humans, not processing by computers. Its treatment of multiple definitions is problematic, because it includes separate entries for each. Further, its inconsistent treatment of parts of speech made it complicated to parse and use.

Moby

Grady Ward's Moby project, located at *http://dcs.shef.ac.uk/research/ilash/Moby*, is a collection of lexical information, including part-of-speech (the one used here), pronunciation, shakespeare and a thesaurus. This project is now public domain. The Moby part-of-speech list (hereafter, Moby) is as comprehensive as the OPTED, but is easily parseable. Moby has one exceptional feature in that it contains one entry per word, listing every possible part of speech associated with each word. Moby's treatment of

parts of speech provided the inspiration for designing the Lexicon class' part of speech value as a non-exclusive bitset. Previously, the Lexicon's part of speech was exclusive, since it was designed to use the BNC database list.

Moby is currently the lexicon of choice for the profiles project. Moby is often more comprehensive than necessary, but the number of errors it generates is dwarfed by the increase in accuracy it provides.

Flame Parser

The flame parser analyzes the body text of each post to find expression of anger by the writer. Many design concepts for the loom2 flame parser came from Ellen Spertus' *Smokey* project at Microsoft Research, especially the rule classes used in pattern matching. See the References section for more information on the *Smokey* paper. The Loom2 flame parser contains four sets of rules: "denial," "face-threats," "insults" and "bad words." Each rule class contains patterns that match various phrase types associated with anger or verbal attacks.

The flame parser does not analyze the underlying syntactic structure of the sentences in posts when analyzing them. The extent to which this limits this analysis is unknown, if such a limitation exists at all. Further, since electronic speech[Note: "Electronic Speech" is my own term for informal written communication online, as it seems to be closer to speech than written text.. I have never seen it written elsewhere in this way, and believe it to be original.] is informal, there may be little to gain from a deep-structure analysis - further research is necessary. That is, the informality of electronic speech means that there exists greater difficulty in structuring the text syntactically, making the error greater and utility smaller.

Unlike flame detectors like *Smokey*, the Loom2 flame detector does not seek to classify entire posts as flame or otherwise. Rather, it seeks to identify elements of flaming within a post. The flame elements are then tallied for each user, not for each post. The Loom2 profiles project seeks to explore usenet on an personal level, and therefore this approach seems more appropriate.

Rules

Each post is broken down into sentences, each of which is passed through a rule set in order to tally instances of flame. The four rule sets are explained below. A given sentence may trigger any number of rules at once. For example, "You're full of s**t" will be caught in the denial rule set as well as the bad words rule set.

Denial

Denial rules match phrases in which the poster disagrees with what the person to whom he or she is responding is saying. Phrases like "You missed the point" and "You're full of crap" are examples of denial statements.

Insults

The insult rule class contains two types of insult: insult by simile and noun apposition. Both of these are adapted from rules in *Smokey*. The insult by simile rules match "like you" or "as you" when closely preceded by any word in the badwords dictionary. Noun appositions match "you" followed closely by any word in the bad words dictionary. Note that other forms of you are matched here, like "your," "yourself," etc. This has the side effect of matching more than just noun appositions in the second instance.

Face Threats

Face threats are statements that are made in order to harm the public status or reputation of another. Calling this set of rules face threats is not to imply that other sets of rules do not contain face threats - they do. However, other sets of rules could be classified more specifically. The rules in this set are general attacks. Such matching phrases are, "screw you" and variants of "I hate you." Other forms of "you" are matched here, much in the way they are in the Insults rule set.

Bad Words

The bad words rule set simply matches words that are in the bad words lexicon and increases the anger score by the strength of the word matched (the "1" "2" or "3" rating. See the Lexicons section). These bad words have their ratings doubled if the word is in all caps. For example, "idiot" carries a score of 1, but "IDIOT" carries a score of two. The assumption here is that, in this case, all caps is in fact a proxy for yelling and is used more forcefully.

Profile Use

The profile.loomprofile package contains classes for working with profiles. These are the only classes that should be necessary for integrating profiles into Loom2.

LoomProfile

The LoomProfile class is abstract, and is the parent class for LoomUserProfile and LoomGroupProfile. It contains the seven basic criteria as well as getter and setter methods for each. This package is designed this way to stress the basic similarity between user and group profiles.

LoomGroupProfile

The LoomGroupProfile object represents a newsgroup's profile. All group profiles are created when the LoomUserProfile initialization routine is done. The LoomGroupFile has no constructor; the user of these classes may either get an array of all group profiles with LoomGroupProfile.getArray() or may get a single group profile with LoomGroupProfile.getGroupFromVector(*groupname*).

LoomUserProfile

The LoomUserProfile object represents the profile of an individual user. There is no constructor for the LoomUserProfile; the profiles are read into memory via the LoomUserProfile.init() method. This method must be called before profiles can be used. An array of profiles can be retrieved with the LoomUserProfile.getArray() method. An individual's profile may be retrieved with LoomUserProfile.getProfile(*email_address*). Similarly, an array of all the profiles of users in a given group may be retrieved with LoomUserProfile.getProfilesForGroup(*groupname*) method.

Since each profile is the weighted average of each partial profile, each user's LoomUserProfile object contains partial profiles, also represented with LoomUserProfile objects, which contain the user's criteria in specific newsgroups. Each partial profile contains a reference to the LoomGroupProfile for the group which it represents.

Since only the overall profile contains partial profiles, the parts array is null always, except in the overall profile and the group reference is null only for the overall profile. Just as the LoomUserProfile is a subclass of LoomProfile to emphasize their similarity, the partial profiles and overall profiles are both instances of LoomUserProfile to emphasize their similarity as well.

References

The following papers, mentioned in the above sections, have been used as reference and inspiration during the development of profiles in Loom2.

The Dynamics of Mass Interaction

This paper by Steve Whittaker et. al. describes their findings in a study on the types of interaction that take place in usenet. Some of the features they found, for example, that a few key players often dominate a newsgroup, have been corroborated here. This paper was very helpful in providing a starting point for thinking about socialization on usenet. In it, for example, the authors explored repeat posting as a measure of "*familiarity* of posters in a newsgroup" (emphasis theirs). Here, repeat posting is used as a measure of activity, but now *responses* to that activity can be a measure of *acceptance*, as well.

Smokey: Automatic Recognition of Hostile Messages

Smokey is a project by Ellen Spertus at Microsoft Research. *Smokey* uses pattern matching rules to find flamelike behavior and categoriezes email messages as flame or non-flame. Spertus notes that though *Smokey* works well and could be used to prioritize mail, it is not perfect. Natural Language technology has a hard time, it seems, analyzing text that is written informally. However, this paper was instrumental in deciding how to build rules for matching flame features.

Flaming: A White Paper

David Kaufer, who wrote this paper, is not a computer scientist. Therefore, this paper has very little in the way of technical content. Rather, Kaufer seems to approach the problem of flame analysis from a psychological direction. Some of his ideas seemed quite insightful. For example, having high-flame and low-flame dictionaries has proven helpful in Loom2. However, his classification system seemed arbitrary. For example, one category contains slurs against draft dodgers, Jews and Italians, and another category contains slurs against people of African and Asian descent. Such a separation makes little sense and seems to be of even less utility.