Segmentation and Classification of Online Chats

Justin Weisz
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
jweisz@cs.cmu.edu

Abstract

One method for analyzing textual chat transcripts is a process called *coding*, whereby individual lines of chat, or blocks of chat, are assigned a topical label. This task is typically performed by humans, and takes a great deal of time. Automated analyses can reduce the amount of time it takes to code chat transcripts, and this paper presents the performance of the Naïve Bayes and Viterbi algorithms for classifying chat data.

1 Introduction

Coding is the process by which data – typically textual data – are assigned labels, in order to understand what the data are about. Typically, coding is performed by humans, and is a tedious and time-consuming task. Further, in order to provide confidence in the coding scheme and assignment of labels, multiple human coders are often used to code the same data, and an inter-rater reliability statistic (such as Cohen's Kappa) is computed to determine the degree of agreement between coders. However, coding is a time-consuming process, one that can be greatly sped up by the use of machine learning algorithms. In this project, I consider the problem of automatically coding two types of data: *chat transcripts* and *broadcast instant messages*.

1.1 Chat transcripts

Chat transcripts are sequences of utterances made by one or more persons in a chat session. Each utterance can be classified as belonging to a certain topic of conversation[1]. Chat data comes from a previous research study by the author in which participants watched a movie together and chatted about it at the same time. An example chat excerpt is presented in Table 1. First, the two participants talk about events occurring in the movie. Then, they talk about whether or not they had

[1] While it is possible that a single utterance could be relevant to multiple topics of conversation, we assume that utterances belong to only one topic.

previously seen the movie. Thus, there are two topics of conversation: things going on *in* the movie, and whether or not the participants *had seen the movie before*.

Table 1: Example chat topics

TOPIC	CHAT
Topic is about events occurring in a movie being watched by P1 and P2.	*P1: They're quite self-righteous, these guys. ;)* *P2: yeah* *P2: especially considering what comes later*
Topic switches to talking about having seen the movie previously.	*P1: Oh. I haven't seen this movie before.* *P1: ok, then I'll not spoil you*

1.2 Broadcast instant messages

Instant messages are small snippets of text containing a message for an intended recipient. Traditionally, instant messages are sent in a one-to-one fashion; however, broadcast instant messages use a one-to-many model. A person sending a broadcast instant message must first choose a *community* to which to send the message. These communities are composed of other users using the broadcast instant messaging system, and each community has its own topic or area of interest. For example, the "tennis players" community would consist of people who play tennis (self-selected, as users can subscribe to any communities they wish, and need not actually be tennis players). Thus, this presents another classification problem: *given an instant message, to which community should it be sent?*

2 Data set characteristics

2.1 Chat transcripts

Chats were logged during a previous research study conducted by the author. In total, there were 2,203 lines of chat, from 15 different chat sessions. After manual segmentation, there were 434 topic blocks (this process consisted of bucketing all of the words used in each topic block, with no respect to the order of those words or of who spoke them). The original coding scheme consisted of 8 classes, including talk about the movie being watched (participants were chatting while watching a movie; this type of chat comprises 29% of the data set), chat about the technology used to view the movie (11%), chat about personal topics (32%), chat about the research study in which they were participating (23%) and miscellaneous topics (5%). A reduced coding scheme is also considered, utilizing three classes: chat about the movie (39%), chat about personal topics (18%) and chat about the study (44%).

The final step of processing was converting each line of chat (or topic block) into a vector of words: for each position *i,* the value in the vector is 1 if the *i*th word exists in the line of chat (or topic block). This has the side-effect of eliminating any influence of word order on the classification. In total, there were 2,270 unique words used in this data set.

2.2 Broadcast instant messages

Broadcast instant messages were recorded during another research study conducted by the author. In total, 5,299 broadcast instant messages were sent to a total of 422

different communities. Word vectors were created for this data set as well, and this data set contained 9,924 unique words.

3 Feature set reduction

Many words were used in both data sets. Each word corresponds to a column in the "feature vector" for a line of chat (or broadcast message). Because classification with many features takes a long time, several strategies were employed in order to reduce the number of features in each data set. A summary of the number of features present after each of the feature-reduction strategies is applied is given in Table 2. Since feature reduction may result in data points losing all of their features (e.g. a chat message composed of only stopwords), Table 3 shows how many data points were present after applying stopword removal (stemming did not eliminate any data points).

Further, in the broadcast message data set, many of the communities are similar to one another (e.g. there are multiple communities for "c programming"). Section 3.4 discusses clustering community names in order to treat similar communities as one.

Table 2: Number of features in each data set for each feature reduction strategy

DATA SET	ORIGINAL	STEM	STOPWORD REMOVAL	STEM + STOP	FREQ. CUTOFF
Chat (lines & topics)	2,270	1,898	2,108	1,741	636
Broadcast messages	9,924	7,861	8,662	6,725	1,470

Table 3: Number of data points for each feature reduction strategy (stopword removal and frequency cutoff were the only strategies that eliminated data points)

DATA SET	ORIGINAL	STOPWORD REMOVAL	FREQ. CUTOFF
Chat (lines)	2,203	2,163	2,080
Chat (topics)	434	434	432
Broadcast messages	5,299	5,282	5,206

3.1 Stemming

Stemming [2] is the process by which words are reduced to their root forms. For example, suffixes are removed, such as "-ing" and "-s", such that "digging" and "dig" become the same word. This allows a classifier to focus on differences between concepts, rather than treating "digging" and "dig" as two separate concepts.

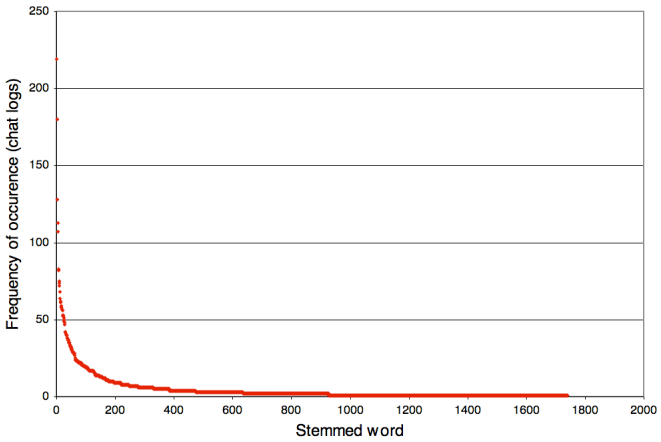
3.2 Stopword removal

Stopwords [1] are words that occur frequently in the English language, such as "a", "and" and "the". Because of their frequent occurrence, they may not add any additional information to aid classification, assuming a uniform distribution over all classes. Stopword removal is performed in order to test this hypothesis.

3.3 Frequency cutoff

Some words are more important than others. For example, misspellings and typos are not as important as properly spelled words, because they may not generalize across chat data. In order to approximate the removal of "unimportant" data, words which do not occur frequently can be removed from the data set. Further, words which occur frequently might also be removed, according to the same logic that justifies stopword removal, however this analysis is not performed. Figure 1 shows the familiar power-law distribution of stemmed word frequencies for the chat data set. The distribution is similar for the words in the broadcast message data set.

Figure 1: Word frequency distribution for chat data



For the purposes of this analysis, we consider words that occur 3 or more times in the chat data, and 6 or more times for the broadcast message data[2]. For both data sets, words that occur 3 or more times comprise roughly 35% and 22% of the total number of words in the data sets, respectively.

3.4 Community name clustering

In order to reduce the number of classes in the broadcast message data set, clustering was performed on the community names. This involved first computing the cosine distance [5] between each community name with each other community name. Next, the `cmdscale` function in MATLAB was used to convert the distance matrix into (X,Y) points for each community name. A plot of these results is shown in Figure 3, and it shows 5 distinct "clusters" of community names (with one outlier). Reducing the number of classes from 422 to 5 seems very drastic, and the implications are discussed in Section 7.

4 Classification algorithms

Two algorithms are used for the classification of the chat data and broadcast message data: Naïve Bayes [3,6], and the Viterbi algorithm for Hidden Markov Models (HMMs). Naïve Bayes is a simple classifier that tries to compute the probability of a class given the input, by considering the base probabilities of each

[2] These were picked based on a combination of visual inspection of the words, and of making the broadcast message data computationally feasible.

class, and the probability of seeing each word in each class. Viterbi takes this one step further, by modeling the probability of seeing one class after another (transitions between classes).

Thus, we have two primary hypotheses: Naïve Bayes will perform equal to Viterbi for the broadcast message data, since there (should be) no correlation between broadcast messages. Viterbi should perform better than Naïve Bayes on the chat data, because there is a high correlation between successive chat utterances. Further, both classifiers should perform better when lines of chat have been grouped into topic blocks, as the topic blocks contain more information than single lines of chat.

5 Method

Naïve Bayes and Viterbi implementations were adopted from the homework for use in this project. Many lines of Perl code were written (over 1,500) in order to process chat and broadcast message logs into a format suitable for MATLAB (feature vectors). Three data sets are used in this analysis: *chat lines* are the individual lines of chat from the chat logs, *chat topics* are the chat data after manual segmentation, and *broadcast messages* are the broadcast message data. For both of the chat data sets, two classification schemes are considered: the original 8 class (section 6.1), and the reduced 3 class (section 6.2). For the broadcast message data, two classifications are also used: the 422 class (section 6.1) and the 5 class (section 6.4).

Each algorithm was trained using two-thirds of the data set, and tested on the remaining third. Unless otherwise stated, the Viterbi algorithm was run with five data points (e.g. using a history of five lines of chat, or five broadcast messages). Five was chosen because the average length of chat topics (from manual segmentation) was five.

6 Results

6.1 Performance on the original data sets

For the *chat lines* data, the training set accuracy of Naïve Bayes was 77.6%, and the test set accuracy was 33.6%. The Viterbi training set accuracy was 76.5%, and test set accuracy was 19.1%. For the *chat topics* data, the training set accuracy of Naïve Bayes was 97.5%, and the test set accuracy was 25.7%. The Viterbi training set accuracy was 88.1%, and test set accuracy was 21.6%.

Neither Naïve Bayes nor Viterbi were run to completion on the original *broadcast message* data set, because the computation took too long to run.

6.2 Effect of reducing the number of classes

Reducing the number of classes for the *chat lines* data from 8 to 3 results in an increase in the training set accuracy of Naïve Bayes, from 77.6% to 83.3%, and an increase in the test set accuracy from 33.6% to 53.9%. The Viterbi training set accuracy increased from 76.5% to 83.5%, and test set accuracy increased from 19.1% to 46.1%.

For the *chat topics* data, the training set accuracy of Naïve Bayes decreased from 97.5% to 95.4%, and the test set accuracy increased from 25.7% to 49.3%. The Viterbi training set accuracy increased from 88.1% to 90.2%, and test set accuracy

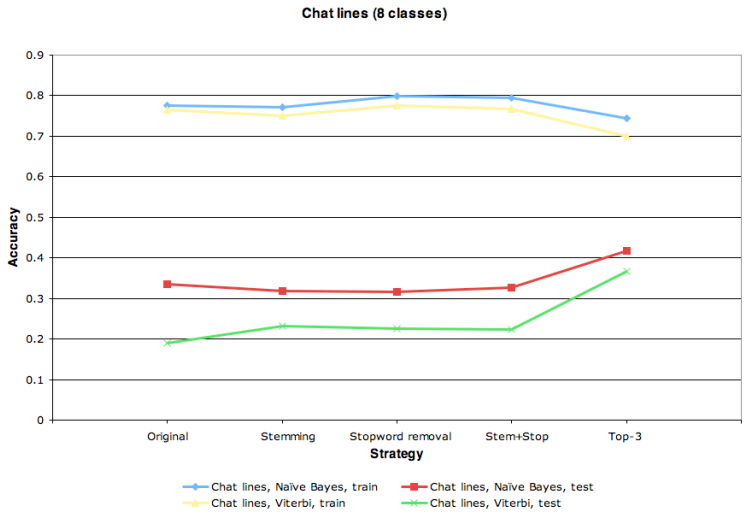
increased from 21.6% to 37.2%. The effect of reducing the number of classes on the *broadcast message* data set is discussed in Section 6.4.

6.3 Effect of feature reduction strategies

Four strategies for reducing the number of features in each data set are considered: *stemming, stopword removal, stemming + stopword removal,* and *frequency cutoff.* The frequency cutoff was performed on the *stemming + stopword* removal data set, and only used words that occurred three or more times.

Figure 2 shows the training and test set accuracies of Naïve Bayes and Viterbi for each of the feature reduction strategies, on the *chat lines* data set. The effects were similar on the *chat topics* data set. Overall, stemming and stopword removal do not have much of an impact on either algorithm's accuracy. Including only words that occur 3 or more times increases test set accuracy, and decreases training set accuracy.

Figure 2: Feature reduction strategies on the *chat lines* data set



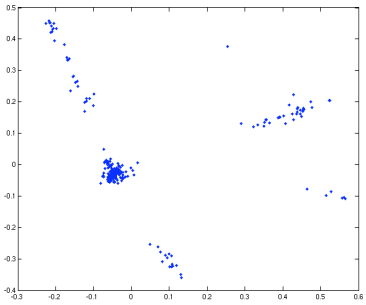
For the *broadcast message* data, performing a frequency cutoff of 6 – only using words that occur 6 or more times – makes classifying the data computationally feasible. In this case, the train & test set accuracies of Naïve Bayes are 85.7% and 26.0%, respectively, for the full 422 classes. For the Viterbi algorithm, the train & test set accuracies are 41.9% and 3.7%, respectively.

6.4 Effect of clustering on the broadcast message data

Figure 3 shows the clustering of community names described in Section 3.4. Five clusters (plus one outlier) are depicted.

Using only 5 classes, Naïve Bayes has train/test set accuracies of 88.0% and 75.9%, respectively. For the Viterbi algorithm, the train & test set accuracies are 86.5% and 73.3%, respectively.

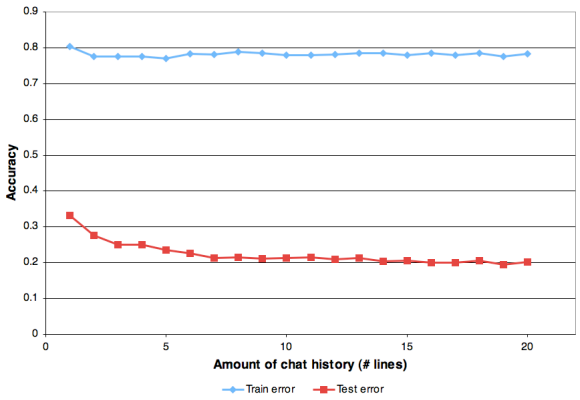
Figure 3: Community name clusters



6.5 Amount of history for Viterbi

Figure 4 shows the effect of history on classification accuracy on the *chat lines* data. The training set error hovers just below 80% no matter how much history is given, but the test set error decreases slightly with more history. The effect is the same on the *chat topics* data set.

Figure 4: Number of chat lines given to Viterbi vs. test set accuracy



7 Discussion

Overall, there was a huge gap between training set accuracy and test set accuracy. This suggests some overfitting is occurring, whereby both Naïve Bayes and Viterbi learn parameters optimal to the training data, but not to the testing data. One possible reason explanation is because of the large number of words used in both data sets. Both algorithms may pick up on rarely used words to classify messages, and as those rarely occurring words do not occur in the test set, accuracy suffers. The frequency cutoff strategy seems to help guard against overfitting, by increasing test set accuracy and decreasing training set accuracy.

Comparing Naïve Bayes and Viterbi on the chat data, in almost every case, Naïve Bayes' test set accuracy was higher than Viterbi's (33.6% vs 19.1% for the original *chat lines* data set). Further, varying how much history Viterbi sees results in a decrease in test set accuracy as more history is given. One explanation for this is that perhaps lines of chat and chat topics are really not as correlated as originally anticipated. Further, the grouping of chat data into topic blocks did not really help either algorithm (Viterbi test set accuracy went from 19.1% to 21.6% on the original

data, Naïve Bayes test set accuracy went from 33.6% to 25.7%). Again, this seems to suggest only a weak correlation in the chat data.

In almost all cases, reducing the number of classes resulted in significant gains in test set accuracy (e.g. 33.6% to 53.8% for Naïve Bayes on the original *chat lines* set). However, this has the drawback of decreased realism: in order to be useful, chat messages really should be classified into 8 (or possibly more classes) instead of just 3, and broadcast messages really should be classified into one or more of 400+ communities.

For the *broadcast message* data, reducing the number of features made the problem computationally solvable in a reasonable amount of time. Using the full set of 422 communities, Naïve Bayes performed fairly well, obtaining a 26% test set accuracy. However, the Viterbi algorithm performed extremely poorly, only classifying about 4% of the examples correctly. This could be because of a strong independence between broadcast messages, and Viterbi is being negatively influenced by previous messages, when there is no correlation among them.

With regard to clustering communities based on their name, while it seemed like a good idea in theory, many of the clusters had communities that really had nothing to do with each other. For example, one cluster included communities named: "IBM Christians", "IBM Discounts" and "IBM hardware repair center", which have nothing in common except "IBM". Future work might use additional information, such as each community's description, to perform the clustering.

Finally, many other algorithms exist for classifying text data; Conditional Random Fields [4] are used to perform the task of segmenting and labeling sequences of text data (i.e. a stream of text data, much as how a chat occurs), and have relaxed assumptions about the independence of the data. EM [7] can also be used to incorporate unlabeled data, enabling automatic collection & classification of chat data, without requiring a human to label many examples.

8 References

[1] List of stop words. http://www.snowball.tartarus.org/algorithms/english/stop.txt

[2] Porter stemming algorithm. http://www.tartarus.org/martin/PorterStemmer/

[3] Friedman, N., Geiger, D., and Goldszmidt, M. Bayesian network classifiers. Machine Learning, 29:131–163, 1997.

[4] Lafferty, J. D., McCallum, A., and Pereira, F. C. 2001. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In Proceedings of the Eighteenth International Conference on Machine Learning, 282-289.

[5] Lee, L. 1999. Measures of distributional similarity. Proceedings of the 37th Annual Meeting of the Association For Computational Linguistics on Computational Linguistics. Association for Computational Linguistics, Morristown, NJ, 25-32.

[6] Mccallum, A., and Nigam, K. A comparison of event models for naïve bayes text classification. In AAAI-98 Workshop on Learning for Text Categorization, 1998.

[7] Nigam, K., Mccallum, A. K., Thrun, S., Mitchell, T. Text classification from labeled and unlabeled documents using EM. Machine Learning, 39:103-134, 2000.