Recognizing Chatting Style

Rohan Puttagunta, Nick Wu, Renjie You

December 14, 2012

1 Introduction

There has been significant work in text classification and stylometry, which is the study of linguistic style. Much of this work has been focused on more formal bodies of text. For example, one might classify emails as spam or not spam, or attempt to identify the true author of a literary work. Our project, however, is concerned with classifying who the writer is in an online one-on-one chat. Online chats use significantly more informal diction and syntax than previously studied texts. This fact allows us to attempt to use unique features and idiosyncrasies of people's informal online writing styles, in addition to traditional techniques and characteristics, to distinguish between writers.

Being able to identify between writers in online chat has a number of significant uses. Court cases may often include a review of online chat communications, and this would be useful in potentially detecting fraudulent or tampered evidence. Furthermore, especially in a live chat conversation, this may also be used to identify social engineering attempts.

2 Data Collection and Formatting

Our data sets use the authors' personal Google chat logs, as downloaded via the Mozilla Thunderbird mail client. We have three data sets, with the primary one being the largest data set containing 65,656 chats, 1,009,638 lines of chat, and 4,325,227 words (excluding words/lines by the authors). The chats span roughly 6 years and feature over 200 unique writers, although over 75% of the words are concentrated in just 20 writers, 50% of the words in 8 writers, and 18% of the words in the top writer. The other data sets are used as well for diagnostics and additional verification of results. Although smaller, they have similar distributions and concentrations.

For the considerations of this project, we will only consider chats by writers with more than 1% of the total number of words. For the main data set, this is 20 people, i.e. we have 20 classification labels. Furthermore, we will only train and test on chats containing at least 20 words by the other writer, because if the chat contains not enough words, it becomes nearly

impossible to make accurate predictions.

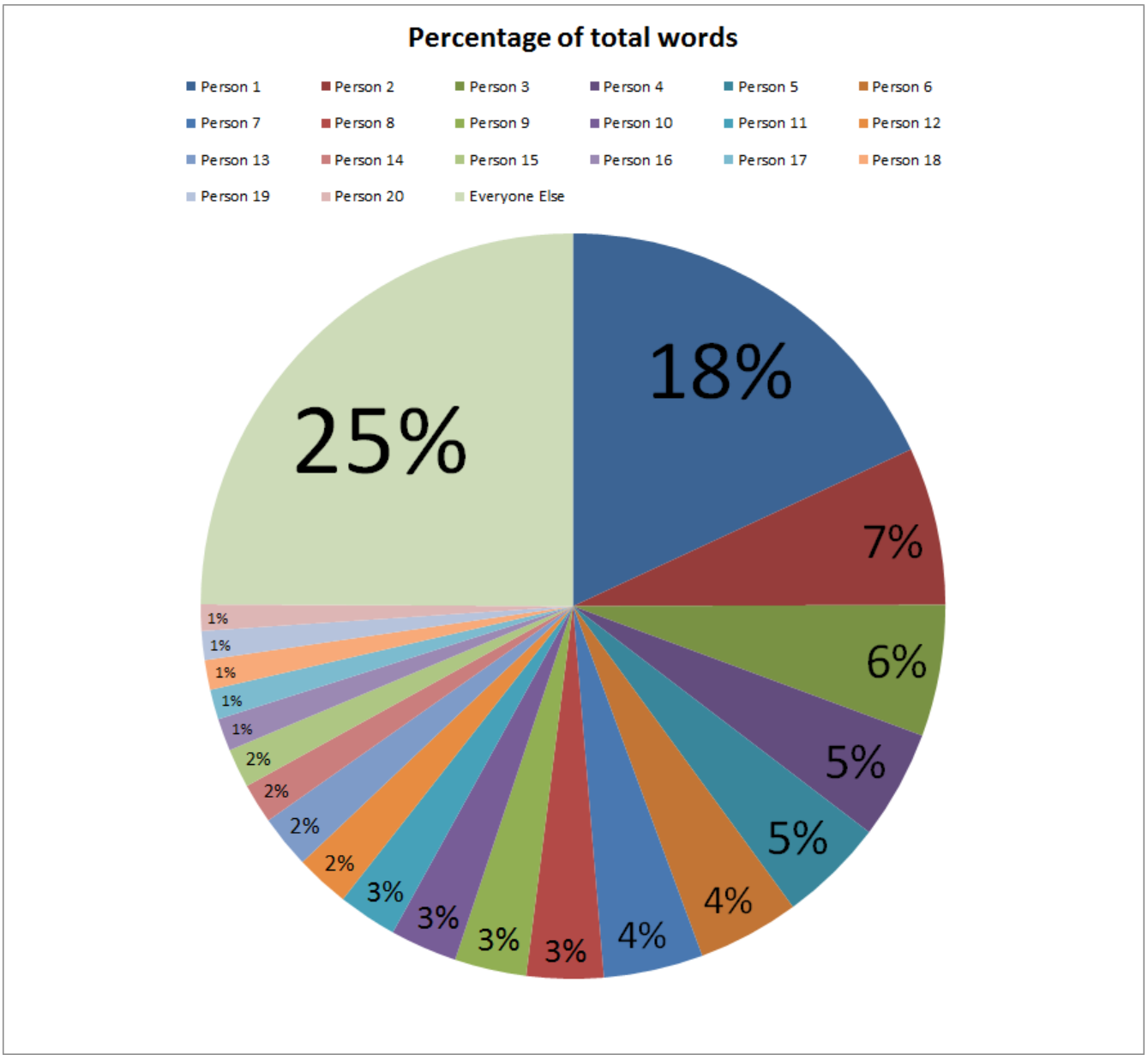

Figure 1: Distribution of total words among the various writers. Notice that the largest portion of the chart, the 25%, actually belongs to "everyone else", i.e. everyone not in the top 20.

We formatted the text by tokenizing across whitespace to create our feature vectors. One additional formatting technique we used was punctuation splitting, which selectively separated punctuation from words and used punctuation as its own feature.

3 Models Used

We implemented four main models:

1. Naive Bayes multinomial event model with Laplace smoothing[1]
2. SVMs with one-versus-one implementation
3. SVMs with one-versus-rest implementation
4. Softmax regression

In each of the first three models, our feature vector represented absolute word frequencies. The SVMs were implemented with linear kernels. In the Softmax regression model, the feature vector was a boolean array which represented the appearance of a word in a chat.

To determine which features were used, we tried the following 3 filters:

1. No filter. If any string appeared after tokenizing across whitespace, it was used as a feature.
2. Top 100 most common English words. We removed all features that were in a dictionary of the 100 most frequently used English words. This is because we believed that common English words would not be a signal for distinguishing between classifications.
3. All English words. We removed all features that were in a relatively comprehensive dictionary of 58,112 English words. This was done to isolate chat specific "words" and idiosyncrasies such as acronyms like "lol" or emoticons.

The filtering was case-sensitive in that we did not filter out any word with a capital letter because we believe capitalization is a fairly distinguishing feature. For each of the three feature vectors, we used both punctuation splitting, as mentioned in the previous section, and no punctuation splitting. This brought the total number of unique feature vectors up to 6. Each of the feature vectors had over 100,000 features; though the filtering made the feature vectors a bit smaller, the primary difference was that most of the more prevalent features were removed.

Furthermore, we also attempted the following models and ideas, but ultimately they were not used for various reasons:

1. Naive Bayes multi-variate Bernoulli event model. This had a low success rate (less than 30%) and was hampered by the fact that line and chat lengths (measured in terms of the number of words) vary wildly from person to person, which significantly influences the training of the parameters and skews the results.

[1]Same as CS 229 lecture notes: Generative Learning algorithms, page 13

2. Adding non-word frequency features to SVMs, such as number of words per line. We found that this did not contribute much to the results, i.e. there was no noticeable change from the existing SVMs. This is largely because the existing number of features is overwhelmingly large (on the order of 100,000) compared to a small number of handpicked features.

3. Using word frequencies as the feature vector for Softmax instead of a binary vector. We could not train the model because the computation either overflowed or underflowed (if we normalized the frequencies) during the exponentiation process.

4 Results

The table below features the four aforementioned models and their success rates under the three types of feature filtering. Each table entry contains two elements. The first element is the accuracy rate without punctuation splitting and the second element is the accuracy rate with punctuation splitting. In each case, we split off 70% of the data set for training and used the remaining 30% for testing to obtain these values. The highest value for each model is italicized and the global maximum of the entire table is bolded.

	Exclude all English words	**Exclude 100 most common English words**	**Exclude nothing**
Naive Bayes	*81.9%*, 79.9%	72.8%, 74.7%	62.6%, 67.6%
SVMs (one-versus-one)	75.4%, 76.3%	78.8%, *79.9%*	77.8%, 79.6%
SVMs (one-versus-rest)	80.4%, 81.2%	83.8%, 84.3%	83.9%, *84.6%*
Softmax	81.5%, 82.3%	83.4%, 84.5%	85.0%, **85.3%**

5 Discussion and Further Work

Our results were overall very satisfactory. The accuracy was as high as 85% under certain settings, which is a significant amount considering there were 20 classification labels and the largest only made up about 18% of the total samples. Generally, our models made classification mistakes on relatively short chats, which is to be expected given that shorter chats contain less information. Furthermore, we found that if we were to give our models a second or third classification choice, our total accuracy may be higher than 95%.

The results suggest that the feature vector after we exclude all English words captures the bulk of the models' predictive powers. Although in some cases adding back in the regular English words improved the testing accuracy, the overall increase was marginal compared to the original base (the left column).

However, there is potentially some latent unexplored structure. For example, in our Naive Bayes model, whenever our model erred, it tended to systematically over-guess classifications

with higher priors. In particular, it tended to guess the writer with the highest prior about 60% to 70% of the time, but the writer with the highest prior only encompassed 18% of the total number of words in chats.

Furthermore, our SVMs might improve with more innovative features or even possibly fewer features (due to the fact that they are currently overfitting). Instead of excluding features (e.g. English words) entirely, we could also investigate weighting schemes to keep them in, but lower their influence on the models' guesses.

Finally, the Softmax classification was hindered at various instances by precision limitations in MATLAB, which was our primary language for implementing it. In particular, the gradient ascent was very sensitive; even small changes to the learning rate caused potential overflow. Additionally, we might've been able to improve Softmax by using a feature vector that includes actual word frequencies (instead of a binary vector) and other interesting features. To implement these changes, one can use Python and its mathematical computing packages that can support higher precision.

6 Acknowledgments

We would like to thank Professor Andrew Ng and the rest of the CS 229 teaching staff for teaching us the techniques that made this project possible. We would also like to thank Catherine Chen for helping with the initial project idea.