CS229A Project Report: Flame War Detection using Naïve Bayes classification techniques

Louis Boval (boval@stanford.edu), Jimmy Tobin (jtobin1@stanford.edu)

Abstract

Classifying text using multinomial naïve bayes is now a common technique, notably for spam e-mail filtering. Our goal with this project was to attempt to adapt the same technique to classifying a harder class of documents, flame wars, or very heated discussions in public internet forums. To do so we implemented a custom web scraper to build a corpus of data, accompanied with our implementation of a naïve bayes classifier which we then augmented with several tweaks to attempt to gain a better recall measure on large training sets, including a technique described in [1] to leverage the large amount of unlabeled data we could gather.

1 Introduction

Internet forums, albeit very engaging and useful, are sadly a magnet for all kinds of abuse, provocations, hate speech or spam. Various techniques are commonly used to detect a subset of these superfluous postings, including preventing automated scripts from posting using CAPTCHAS, or applying spam detection to filter out unwarranted promotional messages. One type of posts which still require human intervention, however, are incendiary comments and the resulting string of heated messages, which we dub as "Flame". They detract greatly from the atmosphere on public forums, and given the ease with which they can be introduced without administrators noticing means many willingly initiate them for the fun of it.

We wanted to see if spam filtering techniques could be applied on discussion threads which display those issues, and generate a model to detect future flames. Given that the contents of messages which initiate those can be fairly well worded in spite of their meaning, we chose to focus our detection attempts on the thread of messages which follow a flame initiator which would display more telltale signs, rather the just the initiator itself. In the following sections, we will describe our process to generate a useful corpus of data from the notoriously flame-ridden youtube.com comment sections, our implementation of a naïve bayes classifier and the paths we followed to adapt it to the problem at hand.

2 Corpus Creation

The first step of the development process for the project once we ascertained a global outline of the techniques we wanted to deploy was to find a corpus of data on which to train the eventual engine. Most such sets of documents used for text classification engines did not have heated speech as a class, we therefore had to build the corpus ourselves, which given the abundance of examples on the web was clearly feasible as long as websites structures lent themselves to our research and properly reflected in code the structure of discussions. We chose to focus our efforts on YouTube, which provides a stripped-down version of its video pages to view comments in bulk (eg, http://www.youtube.com/all_comments?v=myq8upzJDJc).

We wrote a python program to accomplish this task, using the freely available Beautiful Soup (http://www.crummy.com/software/BeautifulSoup/) module to parse HTML code into meaningful units, and writing a simple command line interface to provide access to the functionalities of the scraper. Additionally, the program was structured to allow us to quickly switch the focus website to guard against possible issues during the two month period of the project (we luckily didn't have to, but a redesign of youtube.com very late in the project, although its changes to our pages of interest were minor, proved the precaution to be justified). We eventually were able to gather thread URL from the youtube frontpage, and use those to generate a corpus of unlabeled documents. Given this, we could launch an interactive training session in the command line where the user could classify a subset of his choice, and output

the corresponding training sets. After applying the trained model to the rest of the data, the program could also generate an HTML representation of each discussion thread (for each video) with styling rules to differentiate between different classes of messages, to allow us to view in the browser the results of the classification. With a little effort the program could have been converted to a CGI application to perform the required tasks entirely controlled from a browser, excluding perhaps the initial data gathering step which could run a little slow due to understandable throttling on you tube's part.

After one hour of running the scraping portion of the program, we obtained roughly 100.000 individual messages from about 50 popular video threads, which amounted to about 50.000 documents from our classifying perspective, once messages were organized in threads of responses. Out of this unlabeled corpus, we manually labeled 5000 documents on a flame/not-flame basis, and built a set of training files of increasing length (chosen at random in the total set of labeled documents to ensure fairness) to test the classifying engine's sensitivity to training set size. The training files took the shape of a series of documents, headed by their classification and followed by a sequence of word/word frequency pairs obtained by splitting the messages texts in words, filtering those to group numbers and exclude one-letter or two-letter words as well a non-alphanumerical data. The files were encoded in Unicode to avoid issues with the original website encoding (UTF-8), which was tested by running most of our early runs on a thread mixing Japanese and western characters.

Once the frontend portion for manual classification and corpus creation was built, we were able to focus on implementing a naïve bayes classifier, which we'll discuss in the next section.

3 Naïve Bayes concepts and implementation

Multinomial Naïve Bayes classifier description

As presented in [1], a multinomial naïve bayes classifier attempts to use the frequency of words in a document relative to all the classes which we intend to represent, to determine the class of an unlabeled document. An essential assumption, and reason for the name of the technique, is to consider words to be independent of each other, thus effectively allowing us to build our probabilistic model in parallel across all words in the training documents, and use baye's theorem to compute the probability of a document belonging to a certain class through a fairly fast multiplication.

Without delving into too much detail, which can be reviewed on both bibliography references, let's establish the basic classification model. A labeled document d is a set of word frequencies f_i for each word w_i in the vocabulary, and a class label c: $d=\{..., w_i, ..., c\}$. Combining the document properties for all documents in the training set, we emerge with a collection of word frequencies per class label, which we can use to compute class probability for each unlabeled document during the classification phase. More specifically, for each unlabeled document we compute the maximum likelihood of classes to choose the best one given the model, as follows:

$$argmax_c P(C=c|d) = \frac{P(C=c) * \prod_{i=1}^{n} P(w_i|C=c)^{f_i}}{P(d)} \quad (1)$$

To simplify computations in the actual implementation, we ignore the P(d) denominator which is a common factor across the values we wish to maximize.

The value of $P(w_i|C=c)$ can be approximated using the training data, as follows:

$$P(w_i|C=c) = \frac{N_{ic}}{\sum_{j=1}^{|V|} N_{jc}} \quad (2)$$

where $N_{ic}$is the count of occurrences of word w_i in training documents of class c, and $|V|$ is the size of the vocabulary. The occurrence count is the sum of all $f_i$for all documents in the training set.

$P(C=c)$ is simply computed using the count of documents of class c in the training set relative to the total count.

Implementation

The algorithm was entirely implemented in python, as a script runnable both as standalone taking training files as input, or directly callable from our scraper instance. This afforded us great flexibility when running tests over variations on our algorithm, and shielded us from potential issues with the frontend program which had to contend with network input and large amounts of data to save between runs. Using an intermediary data format helped us evolve the main data structures without risking invalidating existing data.

The training portion of the script was fed labeled documents as an input and produced a model object containing the frequency vocabulary and various other measures

(class probability) to facilitate its use in a classification run. Equation (1) was computed in log format to offset potential over/underflow issues, and converted back to exponential form if the precise values were needed for comparison.

To evaluate the soundness of the algorithm, a subset (20 to 30%) of the training portion was reserved during development to be used as a test set and not integrated in the trained model. We will describe in more detail the use of this set in the fifth section of this report. One important note at this point is to remark that the flame class is greatly outweighed by non-flames in the documents we trained with, and consequently we had to evaluate our results in the context of skewed classes. As such test accuracy was not as meaningful as precision and recall, and we will use both F1-scores and accuracy to evaluate the performance of our models.

4 Improvements upon original algorithm

In its original form, on our initial data sets, the algorithm produced accuracy levels of roughly 60%, and an F1 score of 50%, which required significant improvement. Although increasing the corpus size eventually brought these values up, we worked on improving the results by implementing as well improvements to the model, at various steps in the creation process.

Our main track of investigation in that view was the semi-supervised techniques described in [1, 2], expectation maximization and frequency estimate from unlabeled data. Given the relative complexities of the techniques,we decided to focus on the latter one (seen in[1]), reserving Expectation maximization to an eventual comparison if the extension to frequency estimate yielded a significant improvement.

The main idea behind extending frequency estimate was to use the frequency of instances of a word found in the unlabeled data, and couple it with the class-specific conditional probabilities obtained from the training set. This led to changing the form of equation (2) into:

$$P(w_i|C=c) = \frac{\frac{P(c,w_i)_l}{P(w_i)_l} P(w_i)_u}{\sum_{j=1}^{|V|} \frac{P(c,w_j)_l}{P(w_j)_l} P(w_j)_u} \qquad (3)$$

where subscript u indicates values obtained from the unlabeled data set, and l values obtained from the labeled data set. Equation (3) is then fed into equation (1) to produce the classification prediction.

Without discussing the theoretical soundness of the technique, we note that our implementation yielded underwhelming results, as the precomputing step on the unlabeled data set we produced, given its sheer size, was particularly lengthy, and the improvement to the F1 score was in the single digits (we could not compute the variance of this improvement given the running time of the algorithm). For lack of time we decided not to attempt to fix the algorithm, as we obtained good results without it, but a probable improvement would be to generate the vocabulary at the scraping step, and update it to separate labeled from unlabeled at every step of the manual classification and later training process. The refactoring work implied seemed too great in comparison with the apparent yield of the technique, but should be explored if the system is intended to be used in a real context.

Other alterations to the algorithm

To augment the algorithm, we decided to try other lighter alterations based on intuitions about our target class of documents.

Firstly, we imported an implementation of the Porter word stemming algorithm from the python natural language toolkit set of modules (http://www.nltk.org/), to filter our vocabulary at the training stage (earlier would have led us to a loss of data from the total scraped set which we deemed unreasonable), and condense the size of the vocabulary by removing synonyms, lightly misspelled words and naturally merging words sharing the same root under the assumption that meaning or use would be similar in the context of flame wars. The next section will describe the results we obtained from this addition, as it proved to be an interesting improvement on small data sets.

In addition to stemming we also added call to a Linux utility, Hunspell (http://hunspell.sourceforge.net/), to perform spell checking on the intake, in the hopes that it would help us offset the general disregard for proper spelling (syntax notwithstanding) displayed in youtube comments. It also yielded a small improvement to the model, although when combined with stemming did not prove effective (most likely because, if used before stemming, it risked separating words which would otherwise have been condensed by the stemming step, and when used after would be mostly lost CPU time given that stemming results are orthographically correct), and increased the running time by a large factor (we used a direct call to the shell utility, which is particularly slow

when executed in fast succession), despite the addition of a cache to minimize calls to the spellchecker.

Finally, we remarked that flame threads would be characterized by a longer than usual length, and so decided to attempt using it as a differentiating factor beyond the bag of words. Using it directly in the vocabulary was deemed too tricky, as we would have to reconstruct a discrete probability function on all natural numbers based on the length measurements from the training set, which in itself added a supplementary machine learning problem, having to perform a linear regression on a model to fit to the collection of lengths. For lack of time we decided instead to predict the form of this model and test a few values to see if it yielded a significant improvement. We asserted that the length probability could be modeled by a continuous distribution function shaped after the arc-tan function, which presents the particularity of quickly rising from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ around 0. By shifting the origin set of the function to cover document lengths and changing its arrival set to [0, 1] we obtain a CDF that quickly moves to high probabilities at a certain threshold L_0. The format is roughly $f(L) = \frac{1}{2} * (\frac{arctan(L-L_0)}{\frac{\pi}{2}} + 1)$.

$f(L_d)$ was used to alter the output probabilities used in equation (1), as follows:$P^{'}(C = Flame|d) = P(C = Flame|d) + f(L_d) * P(C = NotFlame|d)$ and

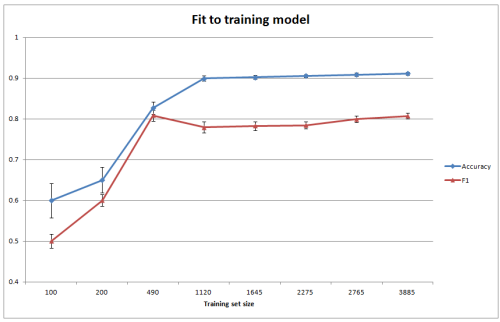
$P^{'}(C = NotFlame|d) = (1 - f(L_d)) * P(C = NotFlame|d)$

We tested a set of values for L_0 to measure its impact on the classifier. We gained better results on small data sets, but this ceased to be true on large ones, and our intuition here is that we need to conduct a regression on both L_0 and an additional α factor applied to $(L - L_0)$ to stretch the transition area.

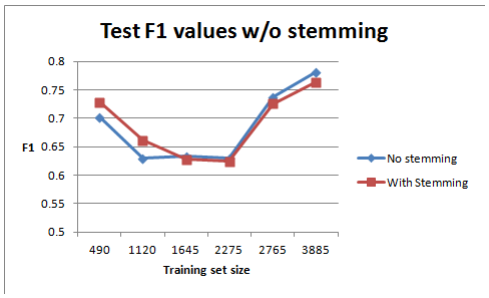
5 Experiments

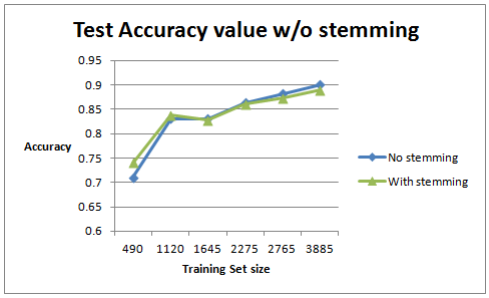
As explained further up, we evaluated the effectiveness of our algorithm by reserving a fraction of the training sets for testing (we did not perform cross-validation). We noted that both accuracy and F1 score improved significantly as the training set size increased, whereas improvement techniques became less effective, possibly due to poor parametrization. The following graphs present a few of the results obtained, focusing on the fit of the trained model to the training data (which only indicated either good performance or massive overfitting), and the performance of the model on the test set with and without the inclusion of stemming, to provide an indication of whether overfitting is occurring in reality.

The following graph presents the values for accuracy and F1 when the trained model is reapplied to the training set. We notice that accuracy and F1 both peak at 0.9 and 0.8 respectively early on, while their variance decreases (the values were obtained by averaging over 15 runs of the algorithm, with a test set being extracted,thus inducing a random factor and indicating the uniformity of the training set). This indicates that the model is stabilizing.



The following two graphs present F1 and accuracy values on the test set with and without stemming. Two remarks of note here, on the one hand stemming loses in effectiveness when the training set size increases, being overtaken by the vanilla implementation, and on the other hand both score continue rising after the previous fit values for the training set stabilize, indicating that overfitting is progressively reduced.





These results indicate that the classifier is fairly effec-

tive on the training set we obtained by manually labeling documents retrieved from youtube comment threads, and lead us to believe the algorithm can be used in some capacity as a flame war detector with the proper source data set. Also, as training examples become more numerous, improvement techniques lose their effectiveness, due probably to a misfit. Taking into account document length, in particular, might be a worthwhile improvement to extract supplementary features from existing data, that would only require implementing a regression technique on the training set, without having to alter the original content gathering.

Conclusion

Through the use of a naïve bayes classifier, and after training roughly 5000 documents obtained by scraping comment threads from youtube.com, we managed to reach a fairly good classification performance for flame war detection. While issues remain with overall performance, scalability and safety against overfitting such a diverse set of text, we believe a naïve bayes based technique can be investigated further to automate the detection of these discussions which claim a significant amount of time for website administrators. Crowd-sourced flame detection already exists by implementing simple voting measures (karma on reddit), and these could be used to augment the model presented here to improve accuracy or possibly better adapt to context that display more complexity than the admittedly limited community on youtube.

References

[1] Su, Sayyad-Shirabad & Matwin, Large Scale Text Classification using Semi-supervised Multinomial Naive Bayes, 2011

[2] Nigam, McCallum, Thrun & Mitchell, Learning to Classify Text from Labeled and Unlabeled Documents, 2000