Detecting Peer-to-Peer Insults

Bertrand Decoster [*†] Fayadhoi Ibrahima [*†] Jiyan Yang [*†]

Abstract

We are mainly using tools in supervised learning to detect insulting remarks in social discussion such as forums or tweets. In the feature selection, we extract features such as the frequencies of words and components of the sentence. We also show that the subject of the sentence does not provide useful information here. A new approach, CNL, combining the idea of naive Bayes and Logistic Regression which gives high accuracy of 82 percents, is provided. Our main approach is applying Ada Boost to a set of predictors obtained from naive Bayes, Support Vector Machine, Logistic Regression, CNL with an outside parameter which is the size of vocabulary. The best accuracy is about 83 percents on the test data with size over 3000.

1 Background

Nowadays, more and more people are using blogs or social networks to express their opinions or share stories. People from different cultures and of different ages may have discussions together in an online forum. Sometimes people may say something that is considered inappropriate by other participants. Insults and other defamatory language may cause hurt feelings and digress from the main topic of conversation. However, it is impossible to have human moderators monitoring every online conversation. Hence we need to use machine learning to detect insulting automatically.

Our aim is to focus on detecting comments that are intended to be obviously insulting to another participant in the blog/forum conversation. We are not looking for insults directed to non-participants (such as celebrities, public figures etc.). Insults could contain profanity, racial slurs, or other offensive language. The insulting nature of the comment should be obvious, and not subtle. Comments which contain profanity or racial slurs, but are not necessarily insulting to another person will not be labeled as such.

We consider this as a binary classification problem. The data we use comes from Kaggle website. Since it is involved with natural language, we use Natural Language Toolkit (NLTK) in Python to extract necessary information from our training data. Furthermore, this problem tends to strongly overfit. We also try to develop techniques or strategies that can be used to guard against overfitting.

We are using knowledge acquired during the quarter in machine learning, as well as some ideas that can be found in papers on the subject written by researchers in computer and information sciences, such as in [RIUM09] and [CZZX11].

This paper will be organized in the following order. In Section 2, we are presenting the methodology we have followed to build a strategy to get a prediction. Then we are exposing some results that we obtained using different methods in Section 3. And finally, we give a conclusion on the work that has been done in Section 4.

2 Methodology

2.1 Philosophy

Here is a basic procedure of our approach. We will explore more details for each step in the following subsections.

[*]Equal contributors

[†]ICME, Stanford University, Stanford, CA 94305.

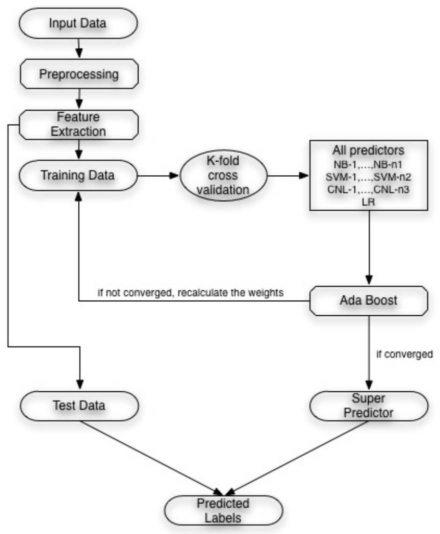


Figure 1: General Strategy for classification

2.2 Cleaning the discussion

We clean the discussion to first get rid of some encoding parts (that may put a bias on results), gather similar words with stemming and discard the less frequent words (to avoid working with too big data). The raw data look like the following data. `\\ xc2\\xa0If you take out the fags and booze and such like, leaving say\\xc2\\xa0 \\c2\\xa3170 per week say and then recognise that they are feeding 8 people then that doesn't seem too extreme (\\xc2\\xa33 or so per day per person).` After cleaning with symbols due to encoding, the new cleaned version of data look like the following. `If you take out the fags and booze and such like, leaving say 170 per week say and then recognise that they are feeding 8 people then that doesn't seem too extreme ( 3 or so per day per person).` Then we do the stemming for the tokens.

2.3 Extracting the features

2.3.1 Finding the subject of the sentence

We tried to use the subject as a feature. While analyzing the data, we realized that a seemingly important part of the results were of the form "You are an XXX" where XXX would be a derogatory word, typically one that appears in the Naive Bayes as the most informative ones. So we had the idea to look deeper into it. If we could detect such sentences, perhaps would they yield a high percentage of insults. We decided to use a Natural Language Processing, namely CoreNLP, from Stanford, to determine what the subject of each sentence is. This statistical tool performs pretty well on standard sentences, and even though a decent portion of our samples were interjections that could not be properly tagged, we thought that we could still go on with this approach. The results were less than impressive. 1. Tagging the subject of a sentence is still not an exact science. Roughly one third of the sentences are very wrongly mistagged. 2. This approach does not detect if the derogatory comment actually targets "you", only the presence of such a term. Finding such a target is even more difficult than finding a subject of a sentence. 3. People are very imaginative when it comes to insult. The sheer variety of ways to insult (interjections, sarcasm, etc.) renders a grammatical approach very hard. We decided not to use this feature alone, and only incorporated it to Naive Bayes as a plus.

2.3.2 Finding other features

Similar to the spam classification problem, "bag-of-words" is always a good idea to start with. We select the 3000 words with highest frequency among all the words in our training set. Denote the vocabulary as V. For each sentence, we compute the frequency of each word in V that appeared in that sentence and store these information in a matrix with size m by $|V|$ where m is the size of our training set.

Also, we compute some basic features that could be extracted simply. These features include the length of the sentence and the ratio of upper case letters in the sentence. Also, to differentiate between an insult directed to a public figure and one towards another participant to the conversation, we can use the fact that the insult should have a target. That means that if the sentence contains the token "you", along with a word considered as insulting, then the sentence will have a higher probability to be insulting. Besides these, we are also interested in the component and structure of the sentence. For example, after tagging each word in the sentence, we compute the frequency of nouns, verbs, adjectives and etc. Also, we compute the number of the most 150 positive informative words appeared in each sentence. After getting this, by a simple feature selection algorithm, we use the following 7 features additional to the frequencies of the words in the bag.

feature name	illustration or condition
`frequencies`	frequencies of each word in the bag
`N_prop`	proportion of nouns
`V_prop`	proportion of verbs
`P_prop`	proportion of propositions
`upper_prop`	proportion of upper case letters
`length`	length of the sentence
`NumPosWords`	number of the most 150 positive words appeared
`SuperFeature`	binary, is 1 if "you" is in the sentence and `NumPosWords` > 0

In the following context, we will call the features other than frequencies as additional features.

2.4 Building "Weak" Classifiers

2.4.1 Trying a simple Naive Bayes approach

The first thing we tried was to use a Naive Bayes approach on the preprocessed raw data. As in the Problem Set 2, we will only use the words in each sentence as features and suppose they are distributed in multinomial. The frequencies of each word in the bag appeared in each sentence might be useful in the computation of maximum likelihood estimator for parameters $\phi_{j,y=0}$ and $\phi_{j,y=1}$.

2.4.2 Adding features and using Logistic Regression and SVM

As discussed above, the features we have are not only the frequencies, but also some other useful quantities. However, the issue is that naive Bayes has the weakness of attributing independence of the features. For these reasons, we turn to some other method. We use the Logistic Regression and Support Vector Machine here. For Logistic Regression, since it is sensitive to be over-fitting, we will only use the 7 additional features to train it. For SVM, since it is more robust, we add every information to it.

2.4.3 Combining naive Bayes and Logistic Regression

Since we have two sets of features, and in practice naive Bayes and Logistic Regression could only use one set of them, we are interested in exploring a new approach to use all the features.

Suppose x and w are two sets of random variables and x and w are independent, meaning that any x_i and w_j are independent. We have the following.

$$\begin{aligned}\mathbf{Pr}(y|x,w) &= \frac{\mathbf{Pr}(y,x,w)}{\mathbf{Pr}(x,w)} = \frac{\mathbf{Pr}(x,w|y)\,\mathbf{Pr}(y)}{\mathbf{Pr}(x)\,\mathbf{Pr}(w)} = \frac{\mathbf{Pr}(x|y)\,\mathbf{Pr}(w|y)\,\mathbf{Pr}(y)}{\mathbf{Pr}\,x\,\mathbf{Pr}(w)} \\ &= \frac{\mathbf{Pr}(x|y)}{\mathbf{Pr}(x)}\frac{\mathbf{Pr}(w|y)\,\mathbf{Pr}(y)}{\mathbf{Pr}(w)} = \frac{\mathbf{Pr}(x|y)}{\mathbf{Pr}(c)}\mathbf{Pr}(y|w) \propto \mathbf{Pr}(x|y)\,\mathbf{Pr}(y|w) = \prod_i \mathbf{Pr}(x_i|y)\,\mathbf{Pr}(y|w)\end{aligned}$$

Hence, this results in a combination of naive Bayes and logistic regression. To see why, suppose that x_i is the feature corresponding to the i-th word of the sentence and w is the additional features. We may get $\mathbf{Pr}(x_i|y)$ by using the traditional MLE for naive Bayes. Meanwhile, $\mathbf{Pr}(y|w)$ is the outcome of Logistic Regression. When comparing this to the traditional naive Bayes formula, it only replaces the term $\mathbf{Pr}(y)$ by $\mathbf{Pr}(y|w)$. Generally, if the accuracy of Logistic Regression is good enough, we should expect a high accuracy of this method since the MLE of $\mathbf{Pr}(y)$ is just the proportion of positive labels. We are not worried about putting too many features to it. Although the x and w we have are not mutually independent, in practice this approach works well (see the results in the next section). We call this new method **CNL**.

2.4.4 Specifying the outside parameter

In naive Bayes, we assume that each word in the sentence is distributed from a multinomial with size 3000. But sometimes some words in the vocabulary are useless. Also, sometimes naive Bayes is vulnerable to the size of V. If we specify the size of the vocabulary as μ, the value of μ will also affect the performance of naive Bayes (see the plot below).

We call such parameter μ an outside parameter since it cannot be optimized for a certain way. What we can do instead is to take some values of μ and treat naive Bayes as different models when it is associated with different μ. Then the only thing to do is to find the optimal predictors amongst these naive Bayes with different μ. The analysis can be made for SVM and CNL as well.

2.5 Obtaining "Strong" Classifiers by Ada Boost

Now we may get a set of "weak" predictors S. For example, naive Bayes with different outside parameters. To get a "strong" classifier, we turn to Ada Boost which can be used in conjunction with many other classifiers to improve their performance. In each iteration Ada Boost could find the optimized predictors in the sense that it can classify the cases correctly where they were misclassified in the previous iterations. To compute this weighted error, we use the k-fold validation approach. In each iteration, the algorithm outputs a classifier and its weight. At the end, we might get a weighted classifier. Its prediction can be made by

$$h_{sp}(x) = \sum_i \alpha_i h_i(x), \tag{1}$$

where α_i, h_i are the weight and predictor from the output of each iteration. It is not hard to show this is a linear combination of all the predictors in S. The final prediction could be made by setting some threshold.

2.6 Overall Strategy

Our main strategy is that firstly we should pre-process the data and extract the features and labels for each sentence we have. Then we will use the following algorithm for training. Basically it contains all the steps we discussed above.

Algorithm 1 Building Super Predictor via Ada Boost

```
Input: Feature matrix $X$ and label vector $y$, outside parameter set $U$.
Output: $predictor, \alpha$.
1: Let $m$ be the size of training data, $d_i = 1/m$ for $i = 1, \dots, m$, $\epsilon_t = 0.1$, $S = \{NB_\mu, SVM_\mu, CNL_\mu, LR\}$
   for $\mu \in U, l = 1$.
2: while $\epsilon_t > 0.45$ do
3:     for $i = 1, \dots, |S|$ do
4:         Compute the $k$-fold cross validation weighted error by $\epsilon_i = \sum_{j=1}^m d_j \mathbf{1}_{h_i(X_j)=y_i}$.
5:     end for
6:     $t = \text{argmin}\epsilon_i, i = 1, \dots, m$.
7:     $predictor_l = t$. $\alpha_l = \frac{1}{2}\log(\frac{1-\epsilon_t}{\epsilon_t})$. $d_j = d_j e^{-\alpha_i y_j h_t(x_j)}/Z$, for $j = 1, \dots, m$ and $Z$ is the normalization
   factor.
8: end while
```

After we get the parameter list and weights α, we can make predictions for any single data according to (1) with a threshold.

3 Results

The results for using the four methods by using a vocabulary of size 3000 are summarized in the following table obtained from a training set with size 4000 and a test set with size 3000.

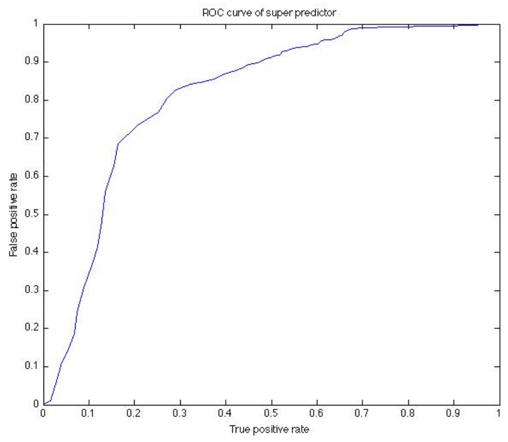
Figure 2: Errors for each predictor

Method	Features	Training error	Test error
Naive Bayes	`frequencies`	0.828184	0.798773
Logistic Regression	additional features	0.854066	0.833396
CNL	everything	0.85333	0.822440
Support Machine Vector	everything	0.814036	0.785795

As we can see, our new approach CNL outperforms naive Bayes. Part of the reason should be the high performance of Logistic Regression. For SVM, its accuracy is smaller than naive Bayes.

Also, we apply Algorithm 2.6 to build the Super Predictor. Basically, the predictors adaptive boosting selected are mainly logistic regression and CNL and the accuracy for Super Predictor is around 83 percents by setting the threshold to zero. It is comparable to the existed predictors. By juggling with the threshold γ, we get the following ROC curve for the Super Predictor.

Figure 3: ROC curve of Super Predictor



By looking into the details, we conclude that when $\gamma = 0$, it gives the most reasonable TPR and FPR. This meets the expectation!

4 Conclusion and Future work

We have tried to investigate how to detect insults, as define in the introduction, that appear in social discussions. To deal with it, we used some knowledge acquired during the quarter in Machine Learning. Indeed, we started with cleaning the data, then finding relevant features and finally trying different classifiers and designing new predictor against over-fitting. Eventually, we used our Super Predictor by combining our predictors via AdaBoost. However, it is still difficult to detect some false negative results such as ”this book is fXXXing good” or new words (wordplays) such as ”yuck fou”. For these cases, we will need to find more elaborated and complex techniques to tackle the issue.