Don't Feed the Trolls: Insult Detection in Online Communities

Matthew Volk, Arun Kulshreshtha, and Stephanie Tsai
Stanford University, CS229 Project

1 Introduction

Online comments have changed the way that Internet users engage with online content. They allow users to express their beliefs, provide criticism, and engage in discussions, rather than just passively read or watch online content.

However, the anonymous nature of online comments can sometimes cause users to be more readily hostile towards others indeed, most Internet users have likely encountered hateful or insulting comments. These kinds of comments create an unwelcoming environment that is harmful for the communities in which they are posted. As such, it is in the interest of website owners to identify and remove insulting comments.

Typically, this is accomplished by having moderators manually review comments, or relying on users to report insulting content. However, this is inefficient and doesn't scale well due to the need for human input. In this project, we explore automating the process of identifying insulting comments.

In particular, our goal is to build a machine learning system that takes as input a comment, and classifies this comment as insulting or not insulting. This system will be trained on a set of training corpora that include known insult and non-insult comments, and then tested on another set of comments whose labels (insult or non-insult) are also known in order to measure its effectiveness.

We start with a simple baseline model using Naive Bayes, and build on top of it by incorporating several improvements, including Laplace smoothing, Stupid Backoff, and threshold variation. In addition to Naive Bayes, we explore using several other classification schemes on top of Naive Bayes to achieve better results. This is done by first building a feature vector for the comment that includes things like similarity scores between the comment and clean/insulting subsets of the training set, the sentiment analysis score for that comment, the frequencies of various parts of speech, and so on. We then experiment with using several classification algorithms – namely, logistic regression, SVMs, and random forests – to classify the comment. By combining several feature extraction and classification techniques, we were able to obtain reasonably good classification accuracy.

2 Related Work

There have been prior studies on using machine learning to classify online comments. The most common approaches seem to involve using classifiers such as logistic regression or SVM on a feature vectors derived from computing several metrics about the words in each piece of text to be classifier. In [1], Yin et al. used an SVM based classifier along with features extracted using TF-IDF word frequencies, sentiment analysis, detecting curse words, and looking at parts of speech (particularly pronouns) to automatically identify online harassment. A similar approach was employed by Chavan et al in [2], who used a similar set of features (namely TF-IDF and n-grams) combined with SVM and logstic regression. Likewise, Dinakar et al. [3] use TF-IDF, n-grams, and basic sentiment analysis using a pre-built dictionary of negative words and phrases combined with an SVM and decision trees.

Overall, it appears that the state of the art is to start with a bag of words representation of the online comments, and to use TF-IDF and n-grams based approaches for selecting features, combined with classifiers such as SVM (most common) or logistic regression.

Some other approaches focused more on sparse vectors of words or patterns. For example, in [4], Razavi et al. combined several flavors of Naive Bayes with a prebuilt dictionary of text patterns indicative of online flaming. By using a multi-level classification approach, they were able to achieve promising results.

In an older piece of work by Spertus [5], the author relies on manually-specified language-based features, such as the presence of second-person pronouns followed by a noun, and runs the resulting feature vectors through a decision-tree based classification system. This approach does not seem as common in newer literature, but we explore the use of decision trees (particularly random forests) in our project while using the sorts of features more typical of newer studies.

3 Dataset and Features

We obtained our data set through a Kaggle competition [6], and generated features using NLP techniques we thought appropriate from relevant literature.

3.1 Data Set

Our data set consists of 6,182 comments scraped from online forums. These are pre-labeled as 1 (insulting comments) or 0 (neutral comments). It came pre-split into a training set of 2,898 neutral comments and 1,050 insult comments and a test set of 1954 neutral comments and 694 insult comments. As a pre-processing step, we parsed these raw text snippets into vectors of individual words, splitting on whitespace and punctuation. An example of each type of comment is below:

Insult: "Oh, you are such an idiot.....you just confirmed that you can't read ... dumb@rse!"

Neutral: "You get the gold star!The best post I've seen on here in months!!Hilarious...Great job,Canadian!PS I think we need you to come down here.I'll sponsor you!!"

3.2 Features

We used several features in an SVM, a logistic regression model, and a random forest. These features are each the result of their own algorithm and/or preprocessing, so a more detailed description will be given in the methods section. The table below lists the features with a short description of each.

Feature Name	Description
Naive Bayes	NB log-probabilities
PoS Tagging	Fraction of nouns, verbs, adjectives
TF-IDF	Term freq-inverse document freq score
Sentiment	Sentiment analysis score
Misspellings	Fraction of misspelled words

4 Methods

The following is a discussion of the methods we employed in our project. It falls into three main sections. First, we implemented and improved upon the standard Naive Bayes algorithm. Second, we defined a more complex feature set, including part-of-speech-based and TF-IDF-based features. Third, we ran the resulting feature sets through three different models: an SVM, logistic regression, and a random decision forest.

4.1 Naive Bayes

We used the unoptimized algorithm as a baseline. Specifically, the algorithm generates two probability distributions of words in the neutral and the insult training corpora by dividing the frequency of individual words by the total number of words in each corpus. It then iterates over the test set and sums the log probabilities of each word in the comment to generate an insult score and a neutral score. It then classifies the word based on which score is higher.

Formally, let $x^{(i)}$ be the i^{th} training comment, $x_j^{(i)}$ be the j^{th} word in that comment, and $y^{(i)}$ be the class (neutral or insult) that $x^{(i)}$ belongs to. ϕ_{neutral} and ϕ_{insult} are the two class priors, and $\phi_{k|\text{neutral}}$ and $\phi_{k|\text{insult}}$ are the probabilities of individual words k given that $y^{(i)}$ is 'neutral' or 'insult,' respectively. Following from this, we use the following equations in our model:

$$\phi_{k|\text{insult}} = \frac{\sum_i \sum_j 1\{x_j^{(i)} = k \wedge y^{(i)} = \text{insult}\}}{\sum_i \sum_j 1\{y^{(i)} = \text{insult}\}}$$

$$\log P_{\text{insult}}(x^{(i)}) \propto \log \phi_{\text{insult}} + \sum_j \log \phi_{x_j^{(i)}|\text{insult}}$$

and similarly for the neutral comments.

4.1.1 NB Improvements

We added several optimizations to the Naive Bayes baseline: (1) ignoring words not found in either training corpus, (2) using Laplace Smoothing over both probability distributions to create a single combined vocabulary, (3) removing stop words, (4) implementing the Stupid Backoff optimization, and (5) using threshold variation.

For Laplace Smoothing, we ran the model with several different Laplace smoothers and settled on the value of 0.01 after plotting the outcomes. We also removed stop words, the most common words in the English language, like "the," "a", and "is," because these appear commonly in both language models and as such aren't particularly discriminatory.

We also implemented Stupid Backoff [7], an optimization for Naive Bayes that tends to generalize well with large training corpora. This model looks at N-grams instead of just unigrams. Our implementation generates probability distributions for bigrams and trigrams, as well, using frequency counts with Laplace Smoothing. Then, without loss of generality, to calculate $P_{\text{insult}}(x^{(i)})$, we iterate over all indices j in the sentence $x^{(i)}$, generating unigrams $U_j^{(i)} = [x_j^{(i)}]$ bigrams $B_j^{(i)} = [x_j^{(i)}, x_{j+1}^{(i)}]$, and trigrams $T_j^{(i)} = [x_j^{(i)}, x_{j+1}^{(i)}, x_{j+2}^{(i)}]$. We skip the bigrams and trigrams that run past the end of the training examples. Then, we calculate an insult score and a neutral score for each sentence by adding the class priors to the sum of $\phi'_{x_j^{(i)}|\text{insult}}$ as before, just for a new definition of ϕ' that takes into account the trigram and bigrams:

$$\phi'_{x_j^{(i)}|\text{insult}} = \begin{cases} P(T_j^{(i)}|\text{insult}) \text{ if } T_j^{(i)} \text{ in Insult LM} \\ \alpha * P(B_j^{(i)}|\text{insult}) \text{ else if } B_j^{(i)} \text{ in Insult LM} \\ \alpha^2 * P(U_j^{(i)}|\text{insult}) \text{ otherwise} \end{cases}$$

Effectively, this tries to use trigram probabilities first, given that they are most specific, then backs off to a discounted (by α, 0.4 in our model) bigram probability if the trigram is unseen, then backs off one last time to a doubly-discounted unigram probability.

Our last optimization was to run threshold variation. In particular, instead of classifying a test example in whichever class had a higher calculated score, we introduced a weight factor β to vary the threshold, meaning we now compare $\log P_{\text{insult}}(x^{(i)})$ and $\beta * \log P_{\text{neutral}}(x^{(i)})$. We chose the value of β that maximized our F1 score. Note that I adopt a different notation than in the cited paper, as this notation is simpler for our particular application.

4.2 Non-NB Features

We then moved on to generating a more rich feature set. Taking the two score generated by our optimized Naive Bayes implementation as the first two features, we also generated the following 4 feature classes.

4.2.1 Part-of-Speech Based Features

One of the first language-based features we tried experimenting with part-of-speech tagging. Using the part-of-speech tagger provided by the NTLK (Natural Language Toolkit – a Python NLP library), we tagged each word in every comment with its part of speech. We then bucketed the parts of speech into nouns, verbs, and adjectives, and used those counts to calculate the fraction of each part of speech in our comments. We added the fraction of nouns, verbs, and adjectives to each comment's feature vector.

4.2.2 TF-IDF

Another feature we wanted to use was to measure the similarity between a given test comment and the insult and clean training corpora. Ordinarily, this could be done by simply using some similarity metric (say, cosine similarity), and computing this between the test comment and each comment in the training set. However, this is not likely to be effective because the bulk of the similarity will be due to words that are common in the English language, which are likely to be common in both the clean and insult corpora.

To work around this, for each of the training corpora we normalized the word frequendies using TF-IDF (term frequency-inverse document frequency), a normalization strategy that adjusts each word's frequency count by the invervse of that word's frequency across both corpora, ensuring that the influence words that are common in both corpora will be reduced [8]. The formula used for computing the TF-IDF score of a token t is as follows, where $f_{t,c}$ denotes the frequency of the given token in a given corpus (either clean comments or insults), N denotes the number of tokens in both corpora, and n_t refers to the number of occurences of the term across both the clean and insult corpora.

$$tf * idf = f_{t,c} + log(1 + \frac{N}{n_t})$$

Using these normalized word frequencies computed for both corpora, we defined two new features that consisted of the sum of the TF-IDF normalized frequencies of each of the words that appear in the comment being classified; we compute this for both the clean and insult corpora. The idea is that whichever of these scores is higher should be indicative of which class the comment belongs to. As will be seen in the results section, this feature proved to be very helpful.

4.2.3 Sentiment

Since insults and inappropriate comments may be more likely to use negative words than neutral comments, we decided to look at the sentiment of the comments. We used a builtin Python sentiment classifier [9] that gave each comment a score between -1 and 1 depending on how negative or positive the words it contained were. We then used these numbers in our sentiment feature.

4.2.4 Misspellings

We believe that insults and inappropriate comments will have more misspelled words than neutral comments will, so we compute the fraction of misspelled words a comment has and use this as one of our features. We use the Enchant spellchecking library [10] to check if our words are misspelled. The resulting feature was the fraction of misspelled words in that comment (fraction instead of raw count to normalize for comment length).

4.3 Models

We used three models: SVM, Logistic Regression, and the Random Forest.

4.3.1 SVM

The first classifier we tried using our new feature vectors was an SVM, as our research indicated that this was one of the most commonly used and promising classifiers in this area. We used the implementation of SVM provided by the scikit-learn Python library [11], using the default parameters. The implementation of SVM in the library solves the following primal problem:

$$\begin{aligned} \underset{w,b,\zeta}{\text{minimize}} \quad & \frac{1}{2} w^\intercal w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to} \quad & y_i (w^\intercal \phi(x_i) + b) \geq 1 - \zeta_i \text{ and } \zeta_i \geq 0 \end{aligned}$$

By default, this implementation uses a Gaussian kernel function. However, after exploring the literature on the subject, it appears that linear kernels are more commonly used for text classification due to the large number of features involved, and upon trying both, we found that the linear kernel significantly outperformed the Gaussian kernel for our purposes. As such, for our project we are using a linear kernel with our SVM.

4.3.2 Logistic Regression

We tried using a logistic regression model next because we saw based on the learning curve in the result section that our SVM model had high variance. We tried this model, using the dual logistic optimization problem with L2 normalization, in an attempt to minimize the high variance. Specifically, the objective function it sought to optimize was:

$$min_\alpha \frac{1}{2} \alpha^\intercal Q \alpha + \sum_{i:\alpha_i > 0} \alpha_i \log \alpha_i + \sum_{i:\alpha_i < C} (C - \alpha_i) \log (C - \alpha_i) - \sum_{i=1}^{l} C \log C$$

subject to $0 \leq \alpha_i \leq C$, $i = 1, ..., l$, where $C > 0$ is a penalty parameter learned by the library and Q was a matrix defined by $Q_{ij} = y^{(i)} j^{(j)} x^{(i)\intercal} x^{(j)}$.

We used the Logistic Regression library provided by scikit-learn to do this [11], which in turn uses the Liblinear library, from which we took the dual optimization problem [12]. We chose the dual optimization problem, per the advice in the Logistic Regression library documentation, because we had substantially more training examples than features.

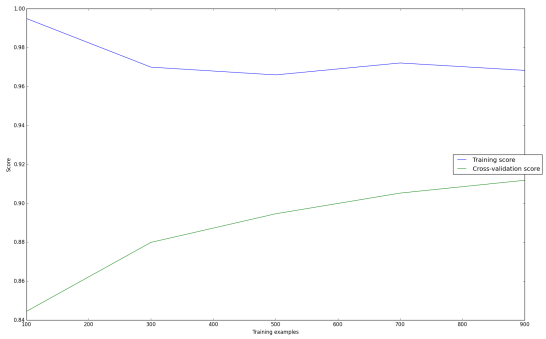
4.3.3 Random Forest

The last model we tried was a random forest classifier. We believe each of our features can be important in classifying comments, but we also believed random forest could algorithmically determine which features are most important better than we could. Random forest helps solve that problem, since at each step in one of its iterations, it picks a random subset of features to try. Thus, it will eventually pick a subset of features that performs best in a tree classifier.

5 Discussion

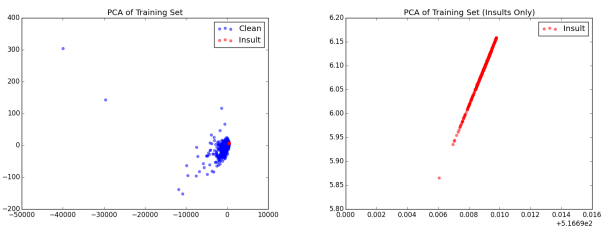
The following is a disclosure of the results of our various models and our analysis of them.

5.1 Bias and Variance



The graph pictured above shows our training and validation scores after running our SVM with a Gaussian kernel. We can see that our data does exhibit some variance, since more training samples seems to help. Because of this, we chose logistic regression and random forest models for our next two models, as these can help reduce variance. We noticed significant improvements in our models after making these decisions, as described in our results.

5.2 Picturing the Data



Before we attempted to feed our data into various classification algorithms, we wanted to visualize our feature space to ensure that our choice of features was reasonable. Since our feature vectors have many dimensions, in order to make a scatter plot of our data we reduced the dimensionality of our features to 2 dimensions using principal component analysis (PCA). At a high level, PCA works by finding the projection of the data that maximimzes the variance between data points in the reduced space. If the PCA plot groups the insult and non-insult comments into distinct clusters, then we believe we can be reasonably confident that our chosen features would enable a classification algorithm to correctly classify the data.

Upon running PCA on our dataset, we obtained the above plots. It appears that there is only one small point at which the insult comments are located. If we zoom in on those data points, as seen in the second plot, it appears that the insult comments actually form a very small line upon being projected into this lower dimensional space. This preliminary result is promising as it suggests that we should be able to classify the dataset somewhat accurately using our choice of features.

5.3 Results of Different Models

Below we show our precision, recall, and F1 scores of our different models, and analyze why they performed as they did. Our key metric was the F1 score. We decided to take an aggressive approach to removing harsh content, meaning we weigh getting all insults about the same as we weigh incorrectly classifying a neutral comment.

5.3.1 Naive Bayes Results

Feature Added (Cumulative)	Precision	Recall	F1 Score
Baseline	0.793	0.116	0.202
Ignore Unseen Words	0.947	0.637	0.762
Laplace Smoothing	0.868	0.818	0.842
Remove Stop Words	0.888	0.798	0.840
Stupid Backoff	0.695	0.778	0.734
Threshold Variation w/o SB	0.812	0.942	0.872
Threshold Variation w/ SB	**0.853**	**0.939**	**0.894**

The baseline performed quite poorly for several reasons, but primarily because it did not handle unseen words well (resulting in many log probabilities of $-\infty$). Once we ignored these words, we saw a substantial improvement. However, instead of ignoring all unseen words, we decided to Laplace Smooth across both language models, (adding words seen only in insults with very low probability to the neutral vocabulary and vice versa). This also helped, allowing us to use these highly discriminatory words that were once skipped.

Next, we tried to remove stop words, thinking it would remove nondiscriminatory words, but it proved to not help here, most likely because some of the comments are quite short, such that removing any context was not helpful. The last major modification was to implemented stupid backoff as described in the methods section. Though it did not help at first, as it only tends to work well with large corpora (to generate enough bigram and trigram samples), we found that once we introduced threshold variation it could be

more discriminatory, in particular improving the recall substantially. More concretely, the combination of these two features helped us reduce the number of false negatives by 70% with a substantially smaller increase in the number of false positives. We believe this to be because higher-order N-grams capture more context, in particular helping discern the purpose of modifier words like "very" in cases like "very good" and "very bad." Whereas "very" might previously have had a slight probability towards either end, "very good" can have a strong neutral score and "very bad" can have a strong insult score.

5.3.2 SVM Results

Feature Added (Cumulative)	Precision	Recall	F1 Score
Naive Bayes (with stupid backoff)	0.760	0.860	0.807
Part of speech tagging	0.754	0.866	0.806
TF-IDF	0.941	0.927	0.934
Sentiment	0.942	0.932	0.937
Misspellings	**0.939**	**0.934**	**0.936**

After incorporating all of our new features, the SVM performed better than Naive Bayes in terms of precision, but performed slightly worse in terms of recall, resulting in an overall higher F1 score. Of our features, it appears that TF-IDF normalized similarity scores were the most helpful, causing the largest increase in precision, recall, and F1 score of all of the features used. This makes sense because these scores emphasize words that appear more commonly in one corpus set or another, and in particular, since one would expect the frequency of curse words to be much greater in the insult corpus, one would expect that this would result in TF-IDF scores would have a reasonably high accuracy.

5.3.3 Logistic Regression Results

Feature Added (Cumulative)	Precision	Recall	F1 Score
Naive Bayes (with stupid backoff)	0.769	0.847	0.806
Part of speech tagging	0.77	0.834	0.801
TF-IDF	0.93	0.922	0.926
Sentiment	0.924	0.925	0.924
Misspellings	**0.925**	**0.928**	**0.927**

Logistic regression performed substantially better than the SVM did, in both precision and recall. We attribute this primarily to the use of $L2$ regularization. We believed our model was overfitting based on the learning curve shown in section 5.1. $L2$ regularization helped ameliorate this problem by adding a penalty term to the objective function that penalized models with large weights, forcing the algorithm to be more conservative in the learned weight vector. This translates into a smoother decision boundary, preventing overfitting.

In terms of individual features, though sentiment analysis and part-of-speech fractions seemed to have introduced noise into the model instead of helping it, using TF-IDF scores seemed to help this model substantially. As before, TF-IDF helped due to the fact that we expect certain terms to appear far more frequently in the insult corpus than in the clean comment corpus.

5.3.4 Random Forest Results

Feature Added (Cumulative)	Precision	Recall	F1 Score
Naive Bayes (with stupid backoff)	0.823	0.723	0.77
Part of speech tagging	0.92	0.791	0.851
TF-IDF	0.961	0.934	0.947
Sentiment	0.964	0.932	0.948
Misspellings	**0.963**	**0.941**	**0.952**

Our random forest does the best, with every additional feature improving the scores. In the bias-variance graph above generated from our SVM scores, we can see that SVM exhibits higher variance that bias. Random forest helps remove this variance without increasing bias by its bagging step, which could explain its outstanding performance.

6 Conclusion/Future Work

The highest-performing model from this study was the random forest run on the full feature set we built up throughout the paper. We had four major numerical results: (1) we improved the baseline Naive Bayes classifier F1 Score from 0.202 to 0.894. We then used the NB scores as features in a larger feature set including NLP features such as TF-IDF scores, sentiment scores, and part-of-speech tag fractions. We got F1 Scores of (2) 0.936 from an SVM, (3) 0.927 from logistic regression, and (4) 0.952 from a random forest.

We expected all three models to outperform our Naive Bayes implementation, as these models had more features to work with. We believe the random forest performed the best because it was able to discern most effectively which features mattered, as it randomly generates decision trees. Note that the random forest was the only model to monotonically increase F1 Score with new features; the other models sometimes regressed as we added features, most likely because they added more noise to these models than help. Further, though logistic regression and SVM performed similarly, we attribute the slight edge the SVM had to the fact that its decision boundary is motivated only by the points close to it (the support vectors) whereas logistic regression's decision boundary is affected by all examples. In this particular application, that means we hypothesize that very clearly insult / neutral comments were skewing the logistic regression decision boundary such that it misclassified things close to the decision boundary, whereas the SVM was able to avoid this problem.

Given more time, we would have liked to move in three directions: (1) there were several other promising NLP features we read about in literature that we didn't have enough time to implement. In particular, we would have liked to use syntax trees to build more structure out of the comments. (2) We would also have liked to try using variance-reducing methods on our SVM, because it exhibited some variance that we we didn't mitigate; if fixed, it's entirely possible that the SVM could outperform the random forest. (3) Lastly, we would have liked to run our models through forward/backward feature selection and hyperparameter tuning algorithms. Because the purpose of this project was exploratory, we did not invest time in optimizing the models, but would like to in the future.

References

[1] D. Yin *et al.*, "Detection of Harassment on Web 2.0." Available: http://www.cse.lehigh.edu/ brian/pubs/2009/CAW2/harassment.pdf

[2] K Dinakar et al. "Modeling the Detection of Textual Cyberbullying." Available: http://www.cl.cam.ac.uk/ rr439/papers/3841-16937-1-PB.pdf

[3] V. Chavan et al. "Machine Learning Approach for Detection of Cyber-Aggressive Comments by Peers on Social Media Network," August 2015.

[4] A Razavi1 et al. "Offensive Language Detection Using Multi-level Classification." Available: http://www.eiti.uottawa.ca/ diana/publications/Flame_Final.pdf

[5] E. Spertus, "Smokey: Automatic Recognition of Hostile Messages," in IAAI-97 Proceedings, 1997. Available: https://www.aaai.org/Papers/IAAI/1997/IAAI97-209.pdf

[6] Imperium, Detecting Insults in Social Commentary. Kaggle, 2012. Available: https://www.kaggle.com/c/detecting-insults-in-social-commentary/data.

[7] T. Brants *et al.*, "Large Language Models in Machine Translation," in Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Prague, June 2007, pp. 858-867. Available: http://www.aclweb.org/anthology/D07-1090.pdf.

[8] "TF-IDF Weighting." Available: http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

[9] P. Kathuria, *Python Sentiment Classifier*. March 2015. Available: https://pypi.python.org/pypi/sentiment_classifier.

[10] R. Kelly, *Python Enchant*. June 2014. Available: https://pypi.python.org/pypi/pyenchant.

[11] Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," JMLR 12, pp. 2825-2830, 2011. Available: http://scikit-learn.org/stable/index.html.

[12] R. Fan, "LIBLINEAR: A Library for Large Linear Classification," *Journal of Machine Learning Research*, 9, August 2008. Available: http://www.csie.ntu.edu.tw/˜cjlin/papers/liblinear.pdf.