Detection of Insults in Social Commentary

CS 229: Machine Learning

Kevin Heh

December 13, 2013

1. Introduction

The abundance of public discussion spaces on the Internet has in many ways changed how we communicate with others. Whether it is a comments section for a controversial news article or a forum for discussing a particular video game, these online spaces allow us to easily share our own opinions and findings, as well as hear about the thoughts of others. Though these discussions can often be productive, the relative anonymity that comes with hiding behind a username has allowed people to post insulting or inappropriate comments. These posts can often create a hostile or uncomfortable environment for other users, one that may even discourage them from visiting the site. This problem is a serious one that website owners commonly face.

One potential way to mitigate this problem is to build a system that can detect whether or not any given comment is insulting. With such a system, website owners would have a lot of flexibility in dealing with this problem. For instance, the owner could choose to automatically block or hide these insulting comments, or flag them so that they can more easily be found by site moderators. The objective of this project is to do just this: build a machine learning system that can accurately classify online posts and comments as insulting or not.

2. Methods

Data: I obtained data for training and testing from *Kaggle*, which is a popular website that hosts machine learning competitions. The data set contains 3947 examples, each of which consists of the text of a particular post and its desired label. A label of 1 represents an insulting post, while a label of 0 represents a non-insulting post. For instance, two examples from the data set are:

- Text: “You’re a moron, truth is beyond your reach”, Label: 1
- Text: “I’ll take that temp…I really hate the heat”, Label: 0

In total, 1077 of the examples are labelled as "insulting", while the remaining 2870 examples are labelled as "not insulting".

Evaluation: The primary evaluation metrics that I used on my system were training accuracy and cross validation with 10 folds. I used the training accuracy to determine how well the model is optimizing with respect to the training set and to what degree it is overfitting to the training set. The cross validation accuracy was more significant because it determined how well the model generalizes to unseen examples. This value is a more accurate representation of how it would perform in a realistic situation.

Development: The system was developed in Python 2.7.4 on the Stanford University corn cluster machines. It was also done with the help of Python's Natural Language Toolkit (NLTK), as well as the PyEnchant open-source spellchecking library.

3. Classification Model

Model Selection

I started out by trying to determine what kind of machine learning model to implement for this task. I found that Naïve Bayes, SVMs, and Logistic Regression are very common models for text classification, so I decided to compare how they would perform on this task. As a result, I implemented each of these with only unigrams counts as features, combined with some basic preprocessing (such as lowercasing all letters and removing punctuation). Their performance on training accuracy and cross validation accuracy is shown in Figure 1.

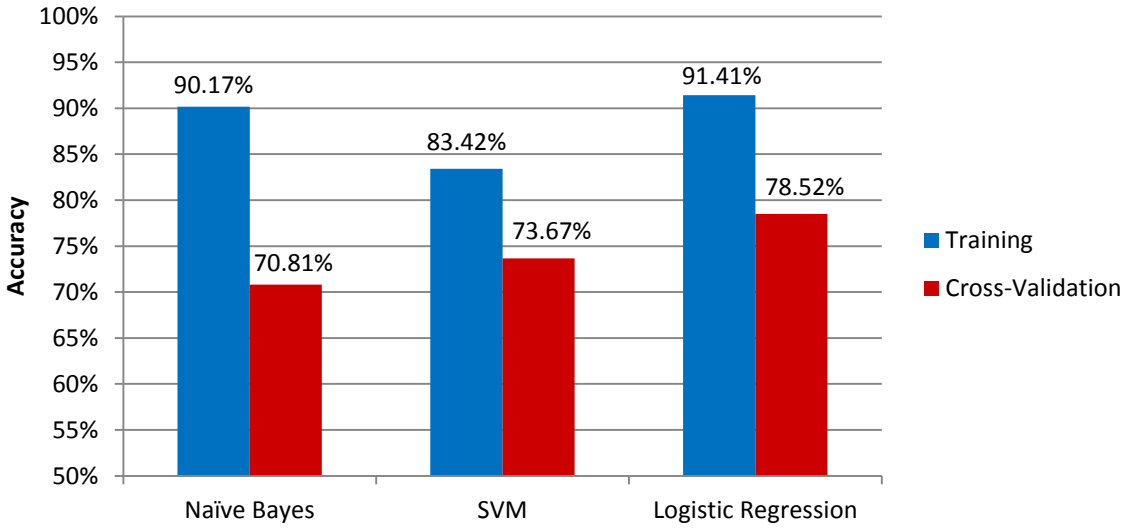

Figure 1: Comparison between models using basic features

It is clear that Logistic Regression performs significantly better than the other two models in terms of cross-validation accuracy (about 5% higher), so I decided to implement the classification system with this model as a base. I also decided to train my parameter vector using stochastic gradient ascent since it is very easy to implement and since my data set is relatively large.

Improving the Model and Feature Engineering

In order to improve the accuracy of the model, I decided to add many more features to my classifier rather than just unigram counts. The additional features that actually helped to increase the cross validation accuracy included:

- Number of punctuation characters within a text
- Ratio of words and characters that are capitalized
- Ratio of words that are "bad" (based on Google's list of "bad" words)
- Character 4-gram and 5-gram counts

Several preprocessing techniques, besides simply lowercasing characters, also helped to improve accuracy, including:

- Tokenizing with NLTK's tokenizer
- Stemming with NLTK's Lancaster Stemmer

The improvement produced by each feature and technique (in the order in which they were added) is shown in Figure 2.

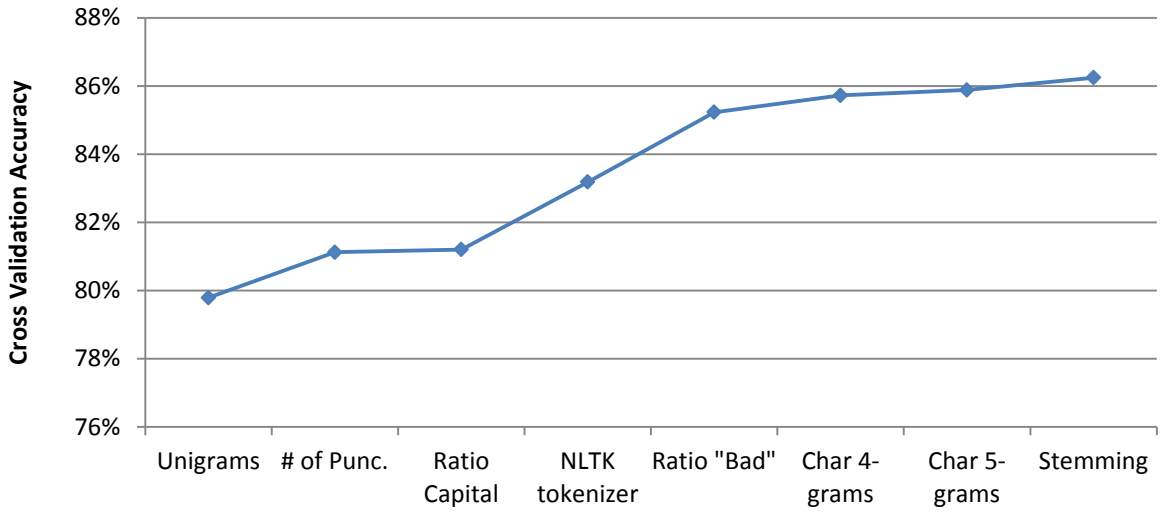

Figure 2: Model improvement from implemented features

On the other hand, many features that I attempted to use did not improve my system, such as:

- Bigram counts, trigram counts, and Character n-gram counts besides n=4 and n=5
- Part-of-speech tagging (provided by NLTK)
- Autocorrection of unigrams (provided by PyEnchant)

Reducing Overfitting

With the addition of all of these features to the Logistic Regression model, the system achieved 100% training accuracy on the full data set, while achieving only about 86% on cross validation accuracy. Hence, it is clear that the model is overfitting significantly in training. The two techniques that I used to deal with this problem are as follows:

- **Bayesian prior:** Instead of using maximum likelihood estimate, I used the Bayesian maximum a posteriori estimate to determine the weights. This technique allows my parameter vector to have a smaller norm, and thus be less susceptible to overfitting.
- **Filter feature selection:** For each feature, I assign it a score which is equal to the absolute value of the weight determined by stochastic gradient ascent. These features will tend to be the most influential during classification. By keeping only a fraction of all extracted features, the dimension of my feature vector will be reduced, which will also helped to reduce overfitting.

After implementing these techniques, the training accuracy dropped to about 94.8%, while the cross validation accuracy increased, indicating that the system was generalizing better than before.

4. Results and Discussion

With all of the above features and techniques implemented in the system, as well as some improvement from parameter tuning, it achieves a cross-validation accuracy of 86.6% on the full data set. Figure 3 shows the breakdown of the results on true positive examples (insulting), and true negative examples (not insulting):

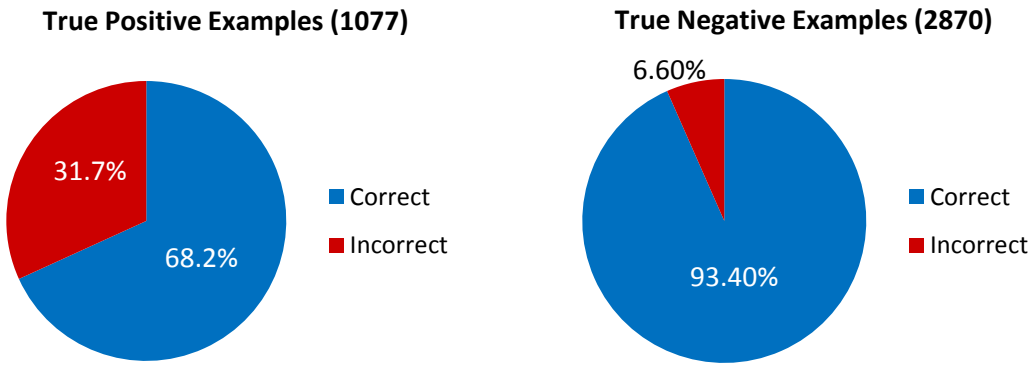

Figure 3: Comparison of accuracy on true positive and true negative examples

It's easy to see that though the system performs very well on negative examples (94.3%), it performs quite poorly in comparison on positive examples (68.2%). Hence, it tends to classify quite conservatively as is. One way of mitigating this issue is to move the decision boundary for classification.

Normally, Logistic Regression classifies using zero as its decision boundary (referring to the dot product between the parameter vector and the feature vector). In order to make the system classify positive more easily, I moved the boundary in the negative direction from 0 to -0.5. After this modification, the accuracy on true positive examples increased to 75.2%, an improvement of about 7%, while the accuracy on true negative examples decreased to 87.9%, a drop of about 5.5%. The overall accuracy, though, dropped slightly to 84.4%. This kind of decision boundary modification can be done to a lesser or greater degree as necessary.

5. Future Work

Though the initial results from this system are promising, there are a number of ways in which I could proceed and improve on this project. One would be to use additional features. The current system only examines the texts in isolation, though it may be useful to consider the relationships between posts, such as the number of insulting posts within the same thread. In a more realistic application, it would also potentially help to include a user's past history, such as the number of insulting posts that a user has written before. It may also be worth trying other NLP techniques, such as chunking to extract phrases from texts. The current system uses stochastic gradient ascent to compute weights because of its simplicity and convergence speed, but I would also like to see how well another optimization algorithm would perform, such as Newton's method.

References

- Dubs, Jamie. "Google's Official List of Bad Words." *Free Art & Technology*. N.p., 28 July 2011. Web.
- Elkan, Charles. *Maximum Likelihood, Logistic Regression, and Stochastic Gradient Training*. Tech. N.p., 17 Jan. 2013. Web.
- Forman, George. *Feature Selection for Text Classification*. Rep. IBM, 3 May 2007. Web.

Acknowledgements

I would like to gratefully thank Professor Ng and all of the CS 229 teaching staff for providing support and advice on this project. I would also like to thank Kuan Peng, a Stanford student, for his contributions toward finding and planning this project.