Detection of Insults in Social Commentary

Hemant Pandey, Sayan Bandyopadhyay and Snigdha Kamal
Computer Science Department
Stony Brook University

Abstract

Insults in social media and public forums is a serious problem which can destroy user experience and possibly ruin the reputation of the forum. In this project, we attempt to solve this problem by building a system which can detect insults in social media commentary and labeling them appropriately. Based on these labels, the website owners or the site moderators can take appropriate action against those problematic users.

Introduction

The world of communication has changed drastically due to the growing abundance of public discussion forums on the Internet. Online spaces allow us to share our thoughts, express our opinions and also understand others point of view, be it on a comments section of a debatable news article or a discussion forum like Reddit. Although the discussions are often constructive, people get the audacity to post insulting comments and posts under the veil of anonymity.Such posts create discomfort for the entire community, even discouraging some users from using the website, which makes this a serious problem faced by social media websites. A possible solution to this problem is a system which automatically detects whether or not a given comment is insulting and takes appropriate action against the user, by blocking him/her from the forum, reporting the user etc. The website owner could also choose to deal with these comments, either by hiding them, or by flagging them to bring them to the notice of site moderators.

In this project, we have developed such a system, which can classify online posts and comments into 'insulting' or' not insulting' using Natural Language Processing Techniques and classification systems based on Machine Learning. It is an application-specific projects, where we have applied the NLP and classification techniques learnt in class to solve a real world problem.
Our algorithm yielded a training accuracy of and a test accuracy of

Dataset

The dataset for this competition was obtained from Kaggle, which is a website which hosts machine learning competitions. Detection of insults in social commentary was one such competition on Kaggle as well. The dataset is in the form of .csv format, with separate files both for training and testing. The training file has 3947 samples, each sample representing a piece of text or a comment, with a corresponding label specifying its value. Label '1' classifies the comment as insulting, whereas Label '0' classifies the insult as harmless or non-insulting. Example comments from the dataset are:

1. "You are a moron, truth is beyond your reach", Label: 1
2. "I will take that temp,I really hate the heat", Label: 0

Totally, 2870 examples have a label of '0' whereas 1077 examples have a label of '1' and classified as insulting.

In order to detect bad and insulting words in the comments in our training and test datasets, we utilized Google's List of Bad words(bad) which is a comprehensive list covering offensive words and phrases which commonly occur in public forums.

Methodology

Preprocessing

The major challenge and the most time consuming task of the project was preprocessing and cleaning of data to achieve better results. The following steps have been performed for the preprocessing:

1. Converted all upper-case letters in the comments to lower case for ease of comparison with the Google list of bad-words
2. Removed all new lines and and HTML tags occurring in text
3. Removed all the non ASCII characters from the comments and trimmed all the white spaces from the beginning and end of the string.
4. Tokenized all words in a given comment using Python's Natural Language ToolKit(NLTK)(nlt)

5. Used the Snowball Stemmer to stem all words in the comment. This converts all the words into their root form. For eg. 'eating' will change to 'eat'.
6. We removed stopwords at first, but words which refers to the person posting the comment like my, me, myself etc and to whom the comment is referred like you, your etc were of utmost significance to our results since we needed the context of the object. So, we have created a custom list and used that in place of the stopwords list provided by NLTK.
7. Replaced shorthand and other Internet slang words with their correct English representations, such as converting u to you, em to them etc.
8. Stored the preprocessed training and test datasets into separate .csv files for further processing

Feature Selection

We have used Sklearn's CountVectorizer, TfidFTransformer and made a custom transformer called BadWordTransformer. In BadWordTransformer, we take into account the frequency of bad words in each comment and then merge both the results of TfidfTransformer and BadWordTransformer using the Feature Union of sklearn(sci)
The pipeline for the process is shown in Figure 1

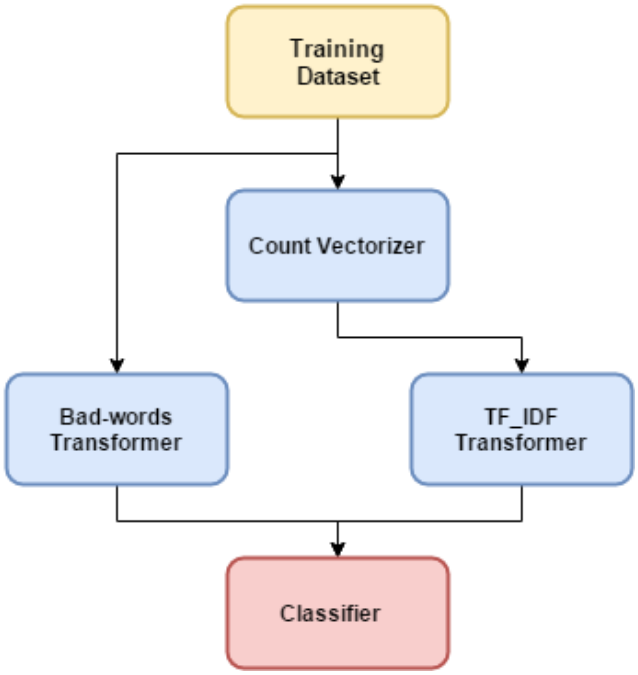

Figure 1: Algorithm Pipeline

Classification

We employed the following classifiers to generate various configurations in an attempt to derive the best accuracy for our system

Naive Bayes Classifier The Naive Bayes model gave an accuracy of 78.8% and an F1-score of 0.78.The Naive Bayes classifier is a simple classifier and performs well when the training set is small, without tending to overfit the data unlike K-nn or logistic regression over small datasets. It is a generative model. If the independence conditions holds, It converges more quickly than discriminative models like logistic regression. It is good for semi-supervised learning. Figure 2

```
Using Naive Baye's
Print here the mean value of Naive Baye's
Print metric report of Naive Baye's
0.780884019645
              precision    recall  f1-score   support

Not an insult       0.77      0.99      0.87      1954
       Insult       0.92      0.18      0.30       693

  avg / total       0.81      0.78      0.72      2647
```

Figure 2: Metric Report for Naive Bayes Classifier

Logistic Regression Unlike Naive Bayes, it works well even when the data may not be correlated. It is also good when the training data is not fixes, as more incoming training data can easily be incorporated into the model. It is used when one wants to obtain a probabilistic framework e.g., to easily adjust classification thresholds, to say when you're unsure, or to get confidence intervals. The metric report we received when we used logistic regression is shown in Figure 3. We received an accuracy of 80% and an F1-score of 0.77 using logistic regression.

```
Print here the mean value of Logistic Regression
Print metric report of Logistic Regression
0.800906686815
              precision    recall  f1-score   support

Not an insult       0.80      0.97      0.88      1954
       Insult       0.77      0.34      0.47       693

  avg / total       0.80      0.80      0.77      2647
```

Figure 3: Metric Report for Logistic Regression

Random Forests Random forests or random decision forests represent an ensemble learning method for data classification, regression and other tasks. They work by constructing many decision trees at the time of training and yield the class which is the mode of classes(classification) or mean prediction(regression) of the individual decision trees. Random decision forests do not overfit to the training data unlike decision trees f overfitting to their training set. We got an accuracy of 76.4% and an F1-score of 0.76. This accuracy was the lowest amongst all the machine learning algorithms tried on this dataset, which implies that random forests probably did not generalize well to the data. Their running time is also much slower compared to other classification methods. 4

```
Print here the mean value of Random Forest Classifier
Print metric report of Random Forest Classifier
0.764261428032
               precision    recall  f1-score   support

Not an insult       0.82      0.87      0.84      1954
       Insult       0.56      0.48      0.52       693

  avg / total       0.75      0.76      0.76      2647
```

Figure 4: Metric Report for Decision Trees

Support Vector Machine Support Vector Machines are extremely useful when the data is not linearly separable in the current dimension. They are highly accurate and tend to overfit the data the least if an appropriate kernel can be found for the data when the data is not linearly separable in the current dimension like Radial Basis, polynomial, linear kernels etc. SVM donâĂŹt give a good probabilistic interpretation of the data however.We received an accuracy of 82.4% and an F1-score of 0.81 with linear SVC. This was the highest accuracy we received of all the classification models. Hence, we chose to further optimize the model by modifying the hyper parameters passed to Linear SVC. Figure 5

```
Print here the mean value of LinearSVC
Print metric report of LinearSVC
0.824329429543
               precision    recall  f1-score   support

Not an insult       0.85      0.93      0.89      1954
       Insult       0.73      0.53      0.61       693

  avg / total       0.82      0.82      0.81      2647
```

Figure 5: Metric Report for SVM

K-means Clustering K-means clustering is an unsupervised algorithm which attempts to classify data points into different clusters or groups. It is a coordinate-descent algorithm. K-clusters(in our case,2) are defined. The choice of k-clusters is crucial to the optimal convergence of the algorithm, hence they should be chosen wisely. K-means clustering algorithm is initialized with k-centroids and each data point assigns itself to the nearest centroid. After this, the new centroid is calculated for each cluster. This process iterates till no data point moves from one cluster to another and the algorithm comes to a convergence. Our model gave an accuracy of 74.6% and an F1-score of 0.69 with k-means clustering. K-means clustering did not work as well probably because the data-points could not align themselves into different clusters completely.Figure 6

Results

We modified the hyper-parameters of the basic model using Linear SVC. We included a L2 regularization constant, and tokenized words into unigrams and bigrams. Our best accuracy obtained was 85.37% and the F1-score obtained was 0.85. Figure 7

Figure 8 shows the learning curve we received for all our classification methods.

```
Print here the mean value of KMeans Clustering
Print metric report of KMeans Clustering
0.746883264073
               precision    recall  f1-score   support

Not an insult       0.76      0.96      0.85      1954
       Insult       0.56      0.15      0.24       693

  avg / total       0.71      0.75      0.69      2647
```

Figure 6: Metric Report for k-means clustering

```
Print here the mean value of Best Configuration
Print metric report of Best Configuration
0.853796751039
               precision    recall  f1-score   support

Not an insult       0.88      0.93      0.90      1954
       Insult       0.76      0.64      0.70       693

  avg / total       0.85      0.85      0.85      2647
```

Figure 7: Metric Report for Best Configuration

Visualizations We generated some visualizations in the form of graphs to gain a better understanding of our training and test datasets.

1. Plot of Frequency of Comments in train dataset where comment is an insult. Figure 9
2. Plot of Frequency of Comments in train dataset where comment is not an insult. Figure 10
3. Plot of Frequency of Comments in test dataset. Figure 11
4. Scatter plot of Number of Badwords versus Comments labeled as Insult 12

Conclusion

The major things which have improved the accuracy of result :

1. Creation of custom stop words list At first, we used the default stop words list, but the words like you, your, me, my etc which plays an important role in our classification were removed and the results were not satisfactory. Thus, a custom stop word lists was created.
2. Using Google List of bad words and creating Transformer We used the list of bad words which have been banned by Google and used it to create a feature which determines the frequency of bad words in a particular comment. This, when combined with Tf-idf Transformer has improved the results tremendously.

Future Work

Although our system achieved a good accuracy and the results are favorable, there are numerous ways this project can be improved, given the leisure of time.
First, more features can be generated from the texts and employed for training and classification. Our system examines words and texts on their own, however it might be useful to consider relationships between posts for example, number

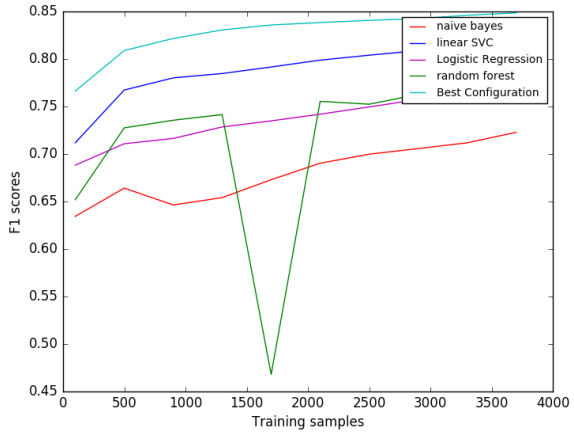

Figure 8: Plot of Learning Curve

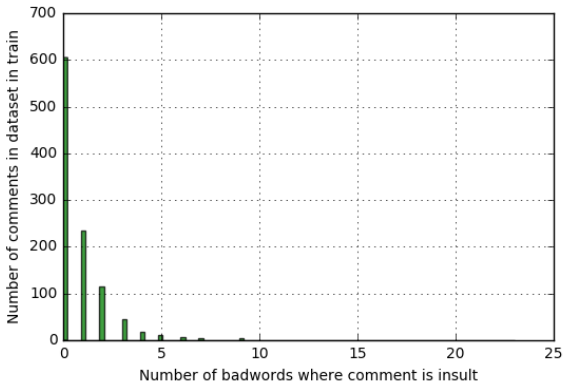

Figure 9: Plot of Frequency of Comments in train dataset where comment is an insult

of insulting comments in the same thread.
Second, in a real world application, we could also keep track of a user's history and flag the user if he/she has a streak of posting insulting comments online.
Third, we could also use other NLP techniques to extract insults from texts, such as semantic parsing to possibly obtain better accuracy.

References

Google's official list of bad words. http://fffff.at/googles-official-list-of-bad-words/.

Nltk official documentation. http://www.nltk.org//.

Python's scikit library official documentation. http://scikit-learn.org/stable/documentation.html/.

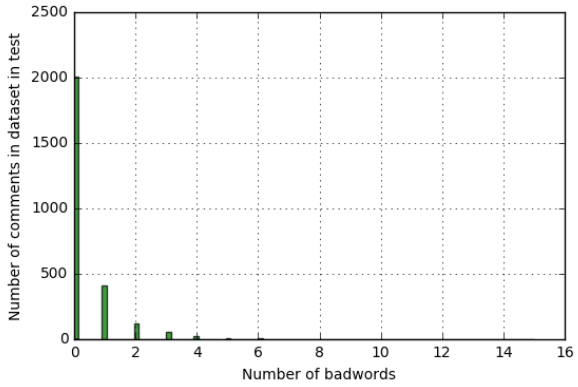

Figure 10: Plot of Frequency of Comments in train dataset where comment is not an insult

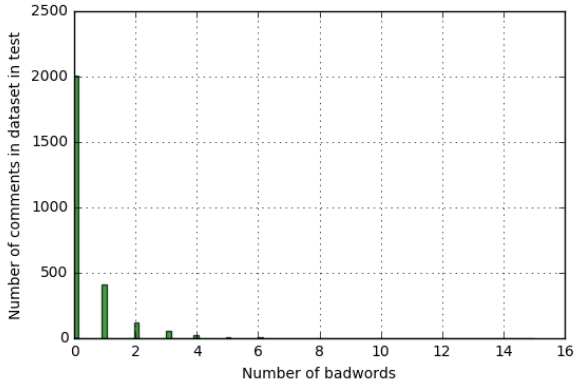

Figure 11: Plot of Frequency of Comments in test dataset

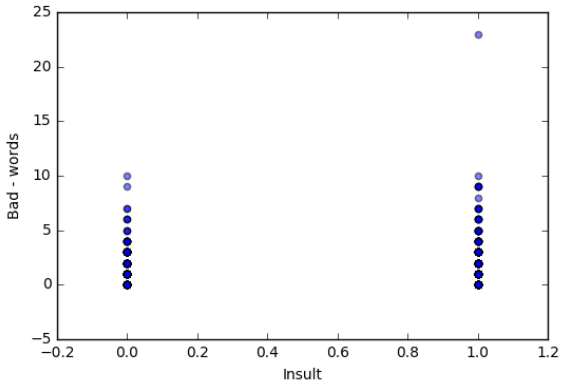

Figure 12: Plot of Frequency of Comments in test dataset