Detecting Insults in Social Commentary

Prashant Ravi
Department of Computer Science
University of Illinois at Urbana Champaign
email:ravi7[at]illinois[dot]edu

Abstract

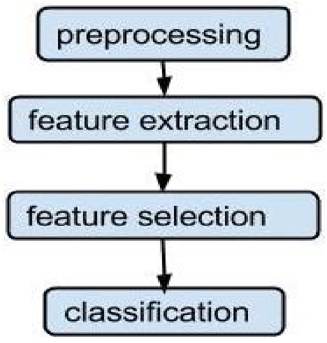
This report gives an overview of the various machine learning algorithms implemented to detect certain comments that may appear insulting to another participant on a social networking platform. Feature selection was performed using n-grams, and the WEKA machine learning toolkit was used to build supervised learning clasifiers, that provided an accuracy of 82% on the test dataset. The dataset was obtained from the popular data science competition portal, Kaggle.

1 Introduction

Machine learning is an important field today with mass availability of internet access, and with it the amount of context-specific data that could be analysed to optimize daily pracitices. Be it Netflix's movie recommendation system or Facebook's face recognition software, the common underlying magic owes it to machine learning algorithms. A big motivation for this, is that humans are great at finding patterns in data sets, and its true that if we had enough humans to find these complex patterns, they may actually perform better than computers, but with the amount of data that has amassed due to the 'Internet of things' its practically impossible to have a human eye to go over such large data sets. Enter, machine learning, which for this very reason has become an indispensable tool for the analysis of data. While, the internet of things has completely revolutionized the communication industry, sometimes, users of these websites break the terms and conditions of the host website that strictly prevent the usage of abusive language. In this project we will apply and analyse some machine learning algorithms that are bundled in WEKA (Waikato Environment for Knowledge Analysis), to a data set to predict if a comment would be considered insulting or not, to a peer on a social networking platform.

2 Methodology

The basic methodology can be described by the following figure, which will be further elaborated on, in the following subsections.



2.1 Preprocessing

The data was presented in the form of a csv file on the Kaggle data science competition portal and this data had to be preprocessed before any further analysis could be carried out. The preprocessing involved the removal of any non-printable hexadecimal characters, special characters and html tags. The raw data, for example contained text such as "A \\xc2\\xa0majority of Canadians can and has been wrong before now and will be again.\\n Unless you're supportive of the idea" would be converted to "a majority of canadians can and has been wrong before now and will be again unless youre supportive of the idea." This was performed in order to get a cleaner training data set. Regex, was used to perform the parse and discard of these special characters, and the rest of the text was converted to lowercase for simplicity. The second part of the preprocessing step involved the stemming of tokens, to reduce the words to its roots, however in the final iteration of the code write up, this step was discarded because some of the insults were being wrongly tokenized and a reduction in overall accuracy occured.

2.2 Feature Extraction

The feature extraction step involved the tokenizing of the words to make a ”bag of words” that could be later analysed using the WEKA toolkit. For this step, the StringToWordVector WEKA command-line tool was very effective in making a word-vector for the data sets. The word vector was constructed such that words occuring in a sentence were presented in this vector along with its frequency. For this feature-extraction step n-grams, specifically 1,2,3-grams were used to find 1,2 or 3 consecutive words in each sentence of the training set. For example, if the following phrase ”a cat walk” is considered, a 2-gram would be ”cat walk”. While genereating these n-grams it was observed that a lot of common words were occuring in the attribute list that won't add any importance to the feature set due to its common use in sentence structuring. Words such as ”the”, ”is”, ”and” are examples of 1-grams that should be ignored. In order to carry this out, a stopwords list that is provided by WEKA by default was used to ignore 1-grams that won't add value to the feature set.

2.2.1 Final feature set

2.2.1.1 Word Vectors : Vector for each comment that consists of tokens of words that occur in the comment along with its Term frequency x Inverse document freqeuncy, commonly known as tf-idf score. This score measures fraction of the occurence of an attribute in the given comment over the entire given document. As discussed by Priya Goyal and Kalra (2013) a keyword's term frequency is the number of times the wor appears in the title. Its document frequency is the number of title the word appears over the total number of titles. The keywords's tf-idf weright is its term frequency divided by its document frequency.

2.2.1.2 Length of string : The length of the comment was included as a numeric attribute

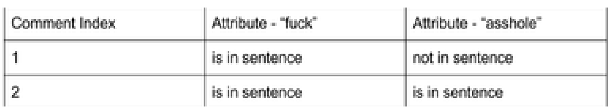
2.2.1.3 Number of special characters : The number of special characters such as ”¿!@¡” that were present in the original comment. But these don't inculde commas, semicolons and period signs as these are syntax that are required by every sentence, and hence not a meaningful feature.

2.2.1.4 Number of upper case letters : It was observed that comments usually contain expletives that were exclaiming or loudly putting forward the abusive terms in upper case letters. For example. ”you SUCK!!!!” is an easy illustration. In order to retrieve the term ”SUCK” all the special characters were replaced with spaces and eventually in order to make tokenizing simpler, the uppercase terms were converted back to lower case.

2.3 Feature Selection

Feature selection is performed to automatically search for the subset of the attributes in the dataset to find the one with with the highest accuracy. We looked at how we were able to condense the feature set of several words in comments and their upto 3-grams but for such a large data set how did we come up with the 1000 ”top” features, ranked per-class basis. A feature slection algorithm was in effect that by the statistical tool, ”Chi-Squared Test” was able to find the best 1000 features from the set of nearly 5000 features that we obtained from the training data set.

2.3.0.5 Chi-Squared Test :The chi-squared test is in effect to reduce the number of attributes that we have in our feature set of word vectors.It does so by testing the independence of two events. To be more specific of its functionality and its relation to our problem, it tests the dependence of the occurnce of the given attribute to the occurence of the class isGood, isBad that we have set up to distinguish between an insult or not.



Comment Index	Attribute - "fuck"	Attribute - "asshole"
1	is in sentence	not in sentence
2	is in sentence	is in sentence

Then to find the dependence of this variable ”fuck” to insult is given by the expection E('fuck', isBad) = $\frac{N('fuck')*N('isBad')}{N}$ where N is the total number of rows, and the numerator is read as the number of rows with attribute ”fuck” times the number of attributes that have the class attribute ”isBad”. And this is true for any n-gram variable too. Then to measure the dependence the formula is given by

$$\chi^2 = \sum_x \frac{(Ex(x,y) - E(x,y))^2}{E(x,y)}$$

where Ex(x,y) reads as the experimental observation of the attribute x, and class attribute y as we look through each of the atttributes, the chi-square is summed up and 1000 variables with the highest dependence to the class attribute are selected by this feature selection algorithm. The end goal for such a procedure is to ultimately avoid overfitting, reduce memory consumption , find meaningful attributes that would later aid decision tree based algorithms to find splitting nodes and in general improve the speed of such algorithms due to the size of the feature space.

2.3.0.6 Batch Filtering After realizing the attribute set, a batch filter was run to tokenize the word vectors for both the training set data and the test data. This assures that both data sets contain the same set of attributes that would be evaluated by the different classifiers.

2.4 Classification

Several different classifiers were appplied on the arff files that were generated by carrying out the feature extraction and feature selection phases, but in the following subsection only the classifiers with the highest accuracy will be discussed for the evaluation of performance on test data set. In addition, k-fold cross validation was performed on the training set before evaluating the test data set, and in this project, for all classifiers, the default k is set to 5. The advanatages of performing k-fold cross validation included that it prevents overfitting of the classifier model and provides generality to the model that could later better classify an independent data set, such as the test data set. The size of test data set here is 2649 instances and train data set is 3948 instances.

2.4.1 Support Vector Machines

Support Vector machines function by putting the training data instances in an n-dimensional vector space to find a clear gap or margin between the support vectors that best represent the different classes of the space.Next, the test data instances are also mapped to this vector space and is classified to be an Insult or not Insult based on which side of the gap, the attributes cause the instance to fall . Refer Table 1 which contains metrics based on test instance classification and other parameters. The C parameter was chosen to be a low value to avoid overfitting.

Metric	Value
Correctly Classified Instances	81.564%
Cross-Validation Accuracy	78.2366%
C parameter	4.0
Mean Absolute Error	0.26319

Table 1: Evaluation metrics using Support Vector Machine Classifier

2.4.2 Naive Bayes Multinomial

Naive Bayes Multinomial was expected to have very good results since its ideal for text classification but it performed poorly. A good reason for this could be that the test data set contained several expletives that were represented alternatively such as "you are a f00l" which by the human eye is read as "you are a fool" but while tokenizing it was not mapped correctly to token with high prior probability of being a negative term.

Metric	Value
Correctly Classified Instances	75.06%
Cross-Validation Accuracy	74.91%
Mean Absolute Error	0.1881

Table 2: Evaluation metrics using Naive Bayes Multinomial Classifier

2.4.3 Random Forest

Random forest is a spin-off of the decision learning algorithm where many decision trees are created over an arbitrary subspace and the decision at each split of the tree is done by a random process instead of a discrete optimized split, and the mode of the classfications of these individual decision trees forms the final output classification, in our case Insults/Not an insult. A parameter that needed to be set was the maximum depth of tree. This is an important factor to determine classifier accuracy as deeper trees reduce bias and more trees reduce variance. The max depth of the trees was set to be the maximum possible, which is the size of attribute set.

Metric	Value
Correctly Classified Instances	79.675%
Cross-Validation Accuracy	78.2366%
Mean Absolute Error	0.1841

Table 3: Evaluation metrics using Random Forest algorithm

2.4.4 AdaBoostM1

Kohavi (1996) discusses the NBTree which is a combination of the decision tree classifier and naive bayes classifier where the decision-tree nodes contain univariate splits as regular decision-trees, but the leaves are comprised of

Naive-Bayesian classifiers. As we can see the classifier when used as the WeakLearn algorithm for the AdaBoostM1 wth iteration count as 100 we see the following results. Table 4 illustrates the metrics evaluated using this approach

Metric	Value
Correctly Classified Instances	79.93 %
Incorrectly Classified Instances	20.07 %
Mean Absolute Error	0.1696

Table 4: Evaluation metrics using AdaBoostM1 with NBTree WeakLearn

In addition, the (LMT) logistic model tree classifier was also used as a WeakLearn algorithm that replaced the NBTree classifier in the previous step. The objective was to learn weak learners that minimize training error on each iteration of the set and thus resultant hypthesis is an aggregation of weak learners that when combined, provide a strong learning classifier. The LMT classifier when used as the weka learner with same number of iterations was observed to perform better than the NBTree classifier on the test data set.

Metric	Value
Correctly Classified Instances	80.846 %
Cross-validation Accuracy	79.14%
Mean Absolute Error	0.1291 %

Table 5: Evaluation metrics using LMT Classifier as WeakLearn for AdaBoostM1 algorithm

3 Related Work

As discussed by Heh (2013), there were a number of things that increased accuracy to his project. These include, adding Google's bad word list to count the number of bad words in the sentence as a feature, the inclusion Stanford's NLTK tokenizer and the Lancaster Stemmer. Such methods greatly increased the cross validation accuracy as shown in Figure 1.

Another interesting feature that was applied by Priya Goyal and Kalra (2013) was the inclusion of the count of words that followed the phrases such as "you are a", "you're" "you" "your". It was observed that this feature was a constant theme in abusive comments, as they were the most direct way to append an insult to one of these second person phrases. In their paper, Priya Goyal and Kalra (2013) discuss

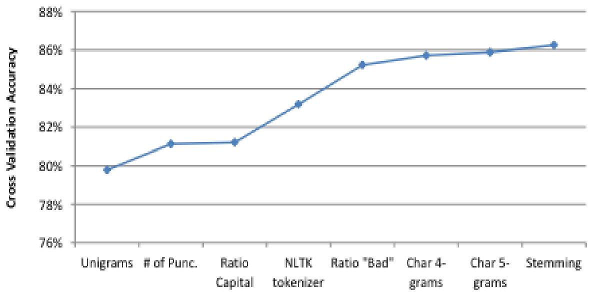


Figure 1: Figure: Cross Validation Accuracy Measure Heh (2013)

how they had originally had a classifier right around 82% accurate and with the inclusion of this feature it rose to 86%, as it was found to be a strong classifier. Recently, another paper was published on Twitter insult detection, by Guang Xiang (2012), performing statistical topic and lexicon modelling on a set of Twitter tweets, using Bootrapping algorithms in NLP.

4 Future work

4.1 Adding a customized stop words and bad words list

The default WEKA stop words list was used for this project however in the future, a more targeted stop words list could be generated. A dictionary of commonly used bad words was released by Google, which they use to flag abuses on their own social media platforms. By integrating a bad words list and a words list that contains common religions, sexual orientations, and generally racy topics could also be included as a feature where its presence in a comment would be a good classifier for insult detection.

4.2 Ability to detect sarcasm

Sometimes people use sarcasm to hint offensive dialog. An example of this, is when a user offends a man of color by saying, "you are too fair skinned to be invited to the party". Currently, this program is unable to detect such sarcastic comments that may offend users but with the help of NLP projects such as Stanford's coreNLP this may be achieved by further understanding the structure of the sentence constructed.

4.3 Collecting more data

To better classifier our data we could collect more data from the social networking platform such as the userIds of the sender and the recipient, the location and date/time of comment posted and the age of users to better classify the instances. For instance, some phrases such as "that artist's voice is sick!" would be interpreted with a positive connotation to a young crowd but an older crowd would find it insulting. In addition, certain users who resort to expletive terms would be more likely to do so, in the future, so it would be a good classifier. Furthermore, data such as number of threads following a post could also be a good classifier, as its natural that one insult follows many more counter insults. Such a strong classifer constructed from these attributes could help social networking platforms to take some action against the users violating their terms and policies or at least dissuade users from using such expletives.

5 Conclusions

It is clear from the metric tables of each of the classifier algorithms that the best classifier in terms of the most number of correctly classified instances is the SVM algorithm , with a close second being the AdaBoost.M1 algorithm using the LMT algorithm as its weakLearn algorithm. However, the classifier performance would still be considered low,overall, and some of the techniques that could be applied to increase the accuracy have been discussed in Related Work and Future Work.In addition,it was observed that some of the most negatively weighted words included "fuck" and "idiot". We can conclude that we have obtained 82% accuracy on the test data set and that the analysis of machine learning is based on not only the algorithm that we apply but also how we handle the data.

6 Acknowledgement

This was project was made possible by the guidance and support of Professor Dan Roth who introduced the students to the WEKA machine learning tool early on in the semester in the form of a problem set. This added some comfort with using the tool and eventually led to the execution of this project.

References

Ling Wang Jason I. Hong Carolyn P. Rose Guang Xiang, Bin Fan. 2012. Detecting offensive tweets via topical feature discovery over a large scale twitter corpus. [Online; accessed 10-Dec-2014].

Kevin Heh. 2013. Detection of insults in social commentary, December. [Online; accessed 8-Dec-2014].

Ron Kohavi. 1996. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 202–207. AAAI Press.

Dr. Amitabha Mukherjee Priya Goyal and Gaganpreet Singh Kalra. 2013. Peer-to-peer insult detection in online communities.