Automatic Detection of Offensive Texts

Elizabeth Andrews
University of Wisconsin-Milwaukee
andrew64@uwm.edu

Sayan Banerjee
University of Wisconsin-Milwaukee
sayan@uwm.edu

ABSTRACT

As the popularity of social networking and blogging increases, the problems that are associated with its widespread use are also on the rise. One of the most common problems faced by internet users today is online bullying or cyber bullying. While it is simple for a human to distinguish between information and insult, the complexities in language, along with the numerous possible ways in which a person can be insulted, makes this process much more difficult for a language processor. In our project, we have studied the structures of various insulting sentences, and created a set of rules to separate information from insults. We have considered insults including offensive language, racial connotations and insults which do not include offensive words, but are still considered offensive.

1. Introduction

Abusive comments and texts on social networks, blogs and other online platforms can be a source of frustration and in some cases, even dangerous. Bullying, in both physical and cyber worlds, has been recognized as a serious national health issue among adolescents. [1] In many cases, administrators of social media manually filter and remove offensive material. However, due to the rapid growth of information online, this process will not be sustainable in the near future. Current software packages that perform filtering are based on a simple keyword search i.e. comments, pages or other forms of content, are blocked if an offensive or vulgar word is found. While this technique can capture some of the offensive texts that are found online, research has shown that only 12% of insults contain vulgar words while over 33% of texts containing vulgar words are not insults [2]. Some messages can contain insulting words but are still not considered insults because they provide factual information. For example, a sentence of the form, ‘John is an idiot’ is an insult. However, a sentence of the form ‘Mary said that John is an idiot’ is not an insult. [3] This is because the latter provides us with information on what Mary said about John. A simple keyword search would have flagged both these sentences as insults.

In order to distinguish between such sentences, in our project we used principles of Natural Language Processing to create a set of rules which will be applied to a given sentence. The rules study the structure of the sentence and the relationship between the words in the sentence to perform this classification.

2. Related Work

A prototype of an insult detection system, called Smokey, has been created by Dr. Ellen Spertus [2]. Smokey combines natural language processing and sociolinguistic observations to identify messages that not only contain insulting words but also use them in an insulting manner. The system first runs each message through a parser. The messages are then converted into Lisp s-expressions. The expressions include an ordinary string, the words in the sentence and the grammar tree for each sentence. These expressions are then processed through some semantic rules written in Emacs Lisp, producing a 47-element feature vector for each message. The feature vector is based on the syntax and semantics of each sentence. This feature vector is then evaluated using simple rules produced Quinlan’s C4.5 decision-tree generator, in order to determine whether a particular message is offensive or not. [2] [3]

Research into this area has also been done by Dr. Chen, Dr. Zhou, Dr. Zhu and Dr. Xu, as presented in their paper ‘Detecting Offensive Language in Social Media to Protect Adolescent Online Safety’ [4]. The study proposed a system that would detect offensive text at a user-level rather than message level. They reasoned that since textual content on social media is unstructured, informal, and often misspelled, techniques to identify offenses at a message-level would not be accurate. In order to perform the required user-level offensiveness detection, the Lexical Syntactic Feature (LSF) architecture was proposed. LSF examines not only the messages but also the person who posts the messages and his/her patterns of posting. In particular, LSF incorporates the user’s writing style, structure and specific cyber bullying content as features to predict the user’s potentiality to send out offensive texts,

Other researched approaches include a bag-of-words approach, viewing the issue as a statistical binary classification problem [5] and using bullying traces for text categorization and role labeling. [1]

3. Methodology

After studying and carefully analyzing the paper, ‘Detecting Flames and Insults in text’ [3], we decided to use a similar annotation scheme in our project. Following the annotation, we created our own set of rules to analyze the structure of a sentence.

3.1 Annotation

Annotation is used in order to store some additional information about each word in the sentence. Each word in the sentence is given an attribute if it belongs to any of the following categories:

1. Person: Any word denoting a human being. For example, he, she, you, sister, etc
2. Religion: Any word denoting a religion or religious item. For example, Christian, Islam, Hindu, Bible, Quran, etc
3. Nationality. Any word denoting a person’s nationality or race. For example, Americans, Indians, etc
4. Personal: Any word denoting a person’s personal attributes. For example, manners, behavior, body, etc
5. ComparableObj: Any non-insulting words a person can be compared to, which would make the overall sentence an insult. For Example, Donkey, Monkey, Pig, etc

A separate lexicon is created, holding these categories and words that fall into this category. For the purpose of this project we have created a file holding about 5 – 10 words in each category. In order to increase the accuracy of the software, additional words should be added to the lexicon.

3.2 Create the node

Each sentence is first parsed using the Stanford Parser to get the dependency relations between the words in the sentence.

Every word then undergoes processing in order to structure them as a node containing all the related relevant information. The structure of each node is as follows:

```
class treeNode{
     String wordInSentence;
     String POStag;
     int positionInSentence;
     int isInsult;
     String parent;
     String parentRelationship;
     int parentPosition;
     String attribute;
}
```

An ArrayList of type treeNode is used to store all the words in the sentence along with the additional information. An explanation of this information is given below:

1. `wordInSentence` – Stores the actual word used in the sentence
2. `POStag` – Stores the POS tag of the word in the sentence
3. `positionInSentence` – Stores the original position of the word in the sentence
4. `isInsult` – Indicates whether the word is an insult or whether that node has been 'insulted' by another node. In order to determine whether a word itself is an insult, we maintain a lexicon of 'bad' words. While processing the word we check if the current word matches any word in the lexicon. If a match is found, we set this property to 1.
5. `parent` – Stores the parent of the current node, in the dependency tree
6. `parentRelationship` – Stores the relation of the node to its parent in the dependency tree.
7. `parentPosition` – Stores the position of its parent.

8. `attribute` – Stores additional information about the word, as mentioned in the annotation scheme above. If no additional property is required for that word, we assign this attribute as “none”

3.3 Rules

Once all the words have been processed and the additional information for each word has been stored in the corresponding node, we traverse through the ArrayList of Nodes in order to apply the rules we have developed. We developed our rules by studying the dependency relations generated when the sentence is passed through the Stanford parser. After all the rules have been applied, we check the root node of our dependency tree. If the isInsult property of the root node is true, we classify that particular sentence as an insult.

By analyzing and studying the structures of various insulting sentences we were able to generalize certain rules as mentioned below:

3.3.1 Rule 1

If the sentence is reporting information, regardless of whether the information being reported is an insult or not, we classify this sentence as ‘Not an insult’.

In such sentences, the root of the dependency tree will be a ‘factive’ verb such as ‘said’, ‘told’, etc. For example, consider the following sentence: ‘Mary said that John is an idiot.’ The dependency relations for this sentence will be as follows:

[nsubj(said-2, Mary-1), root(ROOT-0, said-2), mark(idiot-7, that-3), nsubj(idiot-7, John-4), cop(idiot-7, is-5), det(idiot-7, an-6), ccomp(said-2, idiot-7), punct(said-2, .-8)]

As you can see, the factive verb ‘said’ comes at the root. Therefore, this rule can be applied by checking if the root of the tree is factive or not.

3.3.2 Rule 2

If the root of the tree is insulted, it means that in some subtree (or the entire tree itself), something or someone has been insulted.

For example, consider the sentence ‘John is an idiot’. The dependency relations for this sentence is as follows:

[nsubj(idiot-4, John-1), cop(idiot-4, is-2), det(idiot-4, an-3), root(ROOT-0, idiot-4), punct(idiot-4, .-5)]

The ArrayList of this sentence will contain the following information.

```
Word Label: John
POS Tag: NNP
Position: 1
Parent: idiot
Relationship with parent: nsubj
Position of parent: 4
isInsult: 0
Attribute: none

Word Label: is
POS Tag: VBZ
Position: 2
Parent: idiot
Relationship with parent: cop
Position of parent: 4
isInsult: 0
Attribute: none

Word Label: an
POS Tag: DT
Position: 3
Parent: idiot
Relationship with parent: det
Position of parent: 4
isInsult: 0
Attribute: none
```

```
Word Label: idiot
POS Tag: NN
Position: 4
Parent: ROOT
Relationship with parent: root
Position of parent: 0
isInsult: 1
Attribute: none
```

We see that the root of the tree is ‘idiot’ and the ‘isInsult’ property of the root node is set to 1. However, this does not mean that the overall sentence is an insult. Further processing using additional rules, as explained below, needs to be done before we can come to that conclusion.

3.3.3 Rule 3

Check the noun subject (relation: nsubj) of the insulted root. If the noun subject is a person, nationality, personal attribute, religion or religious item, we keep the ‘isInsult’ property of the root as true.

For example, consider the sentence ‘John is an idiot’. The dependency relation for this sentence is as follows:

[nsubj(idiot-4, John-1), cop(idiot-4, is-2), det(idiot-4, an-3), root(ROOT-0, idiot-4), punct(idiot-4, .-5)]

From the dependency relation, we see that the noun subject of the insulted root ‘idiot’ is a person ‘John’. Therefore the ‘isInsult’ property of the root node is kept as 1. However, if the noun subject was not a person, religion, nationality, personal attribute or religious item, we would have changed the ‘isInsult’ property of the root to 0.

3.3.4 Rule 4

Sentences without a noun subject can also be an insult.

In order to counter the action done by Rule 3 which changes the insult property to false if the required type of noun subject is not present, we use this rule to separately handle insulting sentences that do not contain a noun subject. For example, consider the sentence ‘That fat b**ch!’ The dependency relations for this sentence is as follows:

[det(b**ch -3, That-1), amod(b**ch -3, fat-2), root(ROOT-0, b**ch -3), punct(b**ch -3, !-4)]

While this sentence does not contain a noun subject, it is clearly an insulting sentence.

3.3.5 Rule 5

If the noun subject of the insulted root is an insult, keep the root node as an insult.

Sometimes the actual noun subject of the root node will not be a person, nationality, personal attribute, religion or religious item as required by Rule 3. For example, consider the sentence ‘That fat b**ch called me!’ The dependency relations for this sentence will be:

[det(b**ch -3, That-1), amod(b**ch -3, fat-2), nsubj(called-4, b**ch -3), root(ROOT-0, called-4), dobj(called-4, me-5), punct(called-4, !-6)]

Here, we see that the insulted root ‘called’, has the noun subject ‘b**ch’. While this does not satisfy the requirements of Rule 3, this sentence is an insult. In order to handle such sentences, we have included the rule that Rule 3 can be overridden, if the noun subject of the root is an insult itself.

3.3.6 Rule 6

The presence of a negative modifier, to the insulted root can negate the insult.

For example, consider the sentence ‘Mary is not an idiot.’ The dependency relations for this sentence is as follows:

[nsubj(idiot-5, Mary-1), cop(idiot-5, is-2), neg(idiot-5, not-3), det(idiot-5, an-4), root(ROOT-0, idiot-5), punct(idiot-5, .-6)]

Here, the root node ‘idiot’ will have its ‘ isInsult’ property set to true. The sentence also satisfies Rule 3. However, the presence of the ‘not’ in the statement should set the ‘isInsult’ property of the root to 0. This is done by Rule 6

3.3.7 Rule 7

The presence of a ‘but’ along with the negative modifier (as children of the insulted root node) can negate the effect of the negative modifier as mentioned in Rule 6.

For example, consider the following sentence ‘She is not an idiot but her religion is stupid.’ The dependency relations of the sentence will be as follows:

[nsubj(idiot-5, She-1), cop(idiot-5, is-2), neg(idiot-5, not-3), det(idiot-5, an-4), root(ROOT-0, idiot-5), cc(idiot-5, but-6), poss(religion-8, her-7), nsubj(stupid-10, religion-8), cop(stupid-10, is-9), conj:but(idiot-5, stupid-10), punct(idiot-5, .-11)]

Here, we see that the insulted root ‘idiot’ has the negative modifier ‘not’. According to Rule 6, this should mean that the overall sentence is no longer an insult. However, we see that the presence of another insulting sentence after the ‘but’ changes this result. Rule 7 has been added to account for such sentences. When the sentence present after the ‘but’ is an insult, the overall sentence will be marked as an insult.

3.3.8 Rule 8

The presence of an ‘only’ along with the negative modifier (as children of the insulted root node) negates the effect of the negative modifier as mentioned in Rule 6.

For example, consider the following sentence ‘Jane is not only an idiot but also a fool.’ The dependency relations of this sentence will be as follows:

[nsubj(idiot-6, Jane-1), cop(idiot-6, is-2), neg(only-4, not-3), preconj(idiot-6, only-4), det(idiot-6, an-5), root(ROOT-0, idiot-6), cc(idiot-6, but-7), advmod(fool-10, also-8), det(fool-10, a-9), conj:and(idiot-6, fool-10), punct(idiot-6, .-11)]

Similar to Rule 7, the effect the negative modifier ‘not’ is cancelled due to the presence of ‘only’

3.3.9 Rule 9

If the insulted node is not the root of the tree, and the relation of this node to its parent is dobj,pobj or iobj, check the attribute of the insulted node. If the attribute is ‘comparableObj’ and the parent node is not ‘has’,’have’ or ‘had’ OR if the attribute is not ‘comparableObj’, set the ‘isInsult’ property of the parent as 1.

For example, consider the sentence ‘She has a donkey.’ In order to classify sentences of the form ‘She is a donkey’ as insults, we have added words that can be used in a context similar to the context in which ‘donkey’ is used here, to the lexicon of insults. Therefore when the given sentence ‘She has a donkey.’ is parsed and processed initially, the ‘isInsult’ property of ‘donkey’ will be set to true. The dependency relation of this sentence will be as follows:

[nsubj(has-2, She-1), root(ROOT-0, has-2), det(donkey-4, a-3), dobj(has-2, donkey-4), punct(has-2, .-5)]

According to our rule, even though ‘donkey’ is a direct object of the root, since the root word is ‘has’, the root and therefore the whole sentence, is not marked as an insult.

3.3.10 Rule 10

If the insulted node is not the root of the tree, and the relation of this node to its parent is 'prep', set the parent as insulted.

This rule was added in order to classify sentences such as ‘He thinks like a donkey’ as an insult. The dependency relation for this sentence is as follows:

[nsubj(thinks-2, He-1), root(ROOT-0, thinks-2), prep(thinks-2, like-3), det(donkey-5, a-4), pobj(like-3, donkey-5), punct(thinks-2, .-6)]

Here, the ‘isInsult’ property of ‘donkey’ is set to 1 in the initial processing. Since ‘donkey’ is a ‘pobj’ of ‘like’, according to rule 9, the ‘isInsult’ property of ‘like’ will be set to 1. The dependency relation between ‘like’ and ‘thinks’ is ‘prep’ and therefore, according to Rule 10, the insult property of ‘thinks’ is set as 1. Since ‘thinks’ is the root of the sentence, the overall sentence is marked as an insult.

3.3.11 Rule 11

If the insulted node is not the root of the tree, and the relation of this node to its parent is ccomp, acomp, nn, nsubj, set the parent of the node to insulted.

For example, consider the sentence ‘That fat b**ch called me!’ The dependency structure of this sentence is:

[det(b**ch -3, That-1), amod(b**ch -3, fat-2), nsubj(called-4, b**ch -3), root(ROOT-0, called-4), dobj(called-4, me-5), punct(called-4, !-6)]

Here, the root of the sentence is ‘called’. The insulted node ‘b**ch’ is a child of this node with the relation nsubj. According to Rule 11, the parent of the insulted node will now be marked as insulted. Therefore, ‘called’ is marked as insulted and the overall sentence is now classified as an insult.

3.3.12 Rule 12

If the insulted node is not the root of the tree, and the relation of this node to its parent is 'amod', check if the node is modifying a person, religion or nationality, personal or attribute. If it is, set the insult property of parent to true.

For example, consider the sentence ‘John has bad manners.’ The dependency structure of this sentence is:

[nsubj(has-2, John-1), root(ROOT-0, has-2), amod(manners-4, bad-3), dobj(has-2, manners-4), punct(has-2, .-5)]

Here the ‘isInsult’ property of ‘bad’ is set as 1 in the initial processing. ‘bad’ is not the root of the sentence and is modifying a personal attribute. Therefore Rule 12 is applied in this case to mark the parent of ‘bad’ as an insult.

3.3.13 Rule 13

If the insulted node is not the root of the tree, and the relation of this node to its parent is 'dobj', set the parent as insulted.

Consider the sentence John has bad manners.’ again. As mentioned above, after the application of Rule 12, ‘manners’ will be marked as insulted. However, ‘manners’ is still not the root of the overall tree. Rule 13 was added in order to handle sentences of this form. Here, the relation between the insulted node ‘manners’ and its parent is ‘dobj’. Rule 13 marks the parent as an insult, thereby making the whole sentence an insult.

3.3.14 Rule 14

If the insulted node is not the root of the tree and the relation of this node to its parent is ‘conj’, check if the conjunction is a ‘but’. If the conjunction is a ‘but’ and the insulting word is not acting on a person, nationality, personal attributes, religion or religious item, we set the ‘isInsult’ property of the insulted node to 0.

This is in order to handle sentences of the form ‘She is not an idiot but her religion is stupid.’ As mentioned earlier, the dependency relations of the sentence will be as follows:

[nsubj(idiot-5, She-1), cop(idiot-5, is-2), neg(idiot-5, not-3), det(idiot-5, an-4), root(ROOT-0, idiot-5), cc(idiot-5, but-6), poss(religion-8, her-7), nsubj(stupid-10, religion-8), cop(stupid-10, is-9), conj:but(idiot-5, stupid-10), punct(idiot-5, .-11)]

Rule 7 stated that when the sentence present after the ‘but’ is an insult, the overall sentence will be marked as an insult. Rule 14 performs this check i.e. whether the sentence after the ‘but’ is an insult or not.

4. Implementation

For the tagging and parsing of our sentences, we used Stanford Parser, which is available http://nlp.stanford.edu/software/nndep.shtml. We obtained a lexicon of ‘bad’ words from https://www.kaggle.com/c/detecting-insults-in-social-commentary/forums/t/2744/what-did-you-use?page=2. We modified this list manually. The other lexicon used in order to assign attributes to the words was created by us. Currently, the list includes about 5 – 10 words per category and should be enhanced to include more in the future.

5. Results

Our rules were tested and written primarily from the point of view of simpler sentences. We tested the software on a test set containing 42 sentences, where 33 sentences were insults while the remaining contained insulting words but were not insults or did not contain any insulting words at all. Out of the 33 insulting sentences, our software correctly identified 27 sentences or 81%. The software did not give any false positives.

Bibliography

[1] J.-M. Xu, K.-S. Jun, X. Zhu and A. Bellmore, "Learning from Bullying Traces in Social Media," *Association for Computing Machinery,* 2012.

[2] E. Spertus, "Smokey: Automatic Recognition of Hostile Messages," in *American Association for Artificial Intelligence*, 1997.

[3] A. Mahmud, K. Z. Ahmed and M. Khan, "Detecting flames and insults in text," in *6th International Conference on Natural Language Processing* , 2008.

[4] Y. Chen, P. C. Dept. of Comput. Sci. & Eng., Y. Zhou, S. Zhu and H. Xu, "International Confernece on Social Computing (SocialCom)," in *IEEE*, 2012.

[5] B. Decoster, F. Ibrahima and J. Yang, "Detecting Peer-to-Peer Insults".