

# **Distinguishing Between Factual Information and Insulting or Abusive Messages bearing Words or Phrases in News Articles**

**A thesis presented  
by**

**Altaf Mahmud**  
Student Id: 02201117

**Kazi Zubair Ahmed**  
Student Id: 02101119

to

Department Of Computer Science and Engineering

In partial fulfillment of the requirements

for the degree of

**Bachelor of Computer Science and Engineering**



BRAC University  
Dhaka, Bangladesh  
August 2006

*To our friends & well-wishers*

## **Acknowledgements**

Dr. Mumit Khan was a wonderful advisor. He gave us not only the intellectual freedom of choice in pursuing this topic as our undergraduate thesis, but also consistently gave us the good advice for the idea. Moreover, his lectures on Natural Language Processing help us a lot to learn plenty of useful things and formulate our basic idea. We want to give our heartiest gratitude to Mr. Sumon Shahriar as one of our advisor; he also pushed us to think from many new angles of this research. We would like to thank NaushadUz Zaman for his remarkable and unwavering help about using some useful tools.

We would like to give a special thanks to Dr. Melanie J. Martin, who has done some work on subjective language, for her influential and effective suggestion about background study.

# **Abstract**

## **DISTINGUISHING BETWEEN INFORMATION AND INSULTING OR ABUSIVE MESSAGES BEARING WORDS OR PHRASES IN NEWS ARTICLES**

Altaf Mahmud

Kazi Zubair Ahmed

Supervisor: Dr. Mumit Khan

Since Internet has become the leading source of information for the users, flames or abusive messages have also become the prominent factors of time wasting for retrieving information. Moreover, a text can contain factual information as well as abusive or insulting contents. This paper describes a new approach for an automated system to distinguish between information and personal attack containing insulting or abusive messages in a given document. In NLP, flames or abusive messages are considered as extreme subjective language, which refers to detect personal opinions or emotions in a news article. Insulting or abusive messages are viewed as extreme subset of the subjective language because of its extreme nature. We defined some rules to extract the semantic information of a given sentence from the general semantic structure of that sentence .

# Contents

Acknowledgements .....	i
Abstract .....	ii
Chapter 1: Background .....	1
1.1 Introduction .....	1
1.2 Definition of Insult .....	2
Chapter 2: Literature Review .....	3
2.1 Subjective Language .....	3
2.1.1 Why flames are extreme subset? .....	5
2.2 Background study in subjectivity .....	6
2.3 Related Work Done in Flame Recognition .....	9
2.3.1 Our contrast with Smokey .....	10
2.4 Statistical parsing .....	11
2.4.1 Basic idea .....	11
2.4.2 Probabilistic Context-Free Grammars (PCFGs) .....	11
2.4.3 Estimating Probabilities using a Treebank .....	12
2.4.4 Using Probabilities to Parse .....	12
2.4.5 Obtaining the best parse .....	13
2.4.6 Problems with PCFGs .....	14
2.4.7 Lexicalized PCFGs .....	15
2.4.8 Incorporating head probabilities .....	16

2.4.9 Calculating rule probabilities .....	17
2.4.10 Adding info about word-word dependencies .....	17
2.5 Dependency Parsing .....	18
2.5.1 Basic Concepts .....	19
2.5.2 Dependency functions .....	21
2.5.2.1 Main functions .....	21
2.5.2.2 Verb complementation .....	22
2.5.2.3 Determinative functions .....	24
2.5.3 Robinson's axiom .....	24
2.5.4 Dependency relation .....	25
2.5.5 Stanford dependency parser by Dan Klein .....	26
Chapter 3: Methodology .....	29
3.1 Preprocessing .....	29
3.2 <i>Processing</i> .....	38
3.2.1 <i>Stack Manipulation</i> .....	40
3.2.2 <i>Marking phase</i> .....	47
3.2.3 Building Tree .....	50
3.2.4 Detection .....	51
Chapter 4: Result .....	71
Chapter 5: Conclusion, Limitations and The Future Work .....	73
5.1 Conclusion .....	74
5.2 Limitations .....	74
5.3 Future Work .....	75
References .....	-1-

## List of figures

fig-1 .....	12
fig -2 .....	16
fig -3: Lexicalized parsing can be seen as producing <i>dependency trees</i> .....	18
fig -4 .....	27
fig -5 .....	28
fig -6 .....	39
fig-7 .....	39
fig -8 .....	41
fig -9 .....	43
fig -10 .....	43
fig -11 .....	44
fig -12 .....	44
fig -13 .....	46
fig -14 .....	46
fig -16 .....	52
fig -17 .....	53
fig -18 .....	54
fig -19 .....	55
fig -20 .....	55
fig -21 .....	56
fig -22 .....	57
fig -23 .....	58
fig -24 .....	58

fig -25 .....	59
fig -26 .....	60
fig -27 .....	61
fig -28 .....	61
fig 29 .....	62
fig 30 .....	63
fig 31 .....	64
fig 32 .....	64
fig 33 .....	64
fig 34 .....	65
fig 35 .....	67
fig 36 .....	68
fig 37 .....	69
fig 38 .....	69



# Chapter 1: Background

## 1.1 Introduction

Most of the time, Internet users get frustrated when they search for any information in a specific site, because some peoples take it as a fun to use personal attacking or insulting messages for on-line communication. The best example can be ‘wikipedia [1]’ where many times these occurrences are happened, which they called ‘wiki vandalism’. If an automated system will help a user for distinguishing flames and information in a web page or in e-mail, user can decide whether or not to read that article. Some messages can contain insulting words or phrases but still they are considered as factual information. For example: a sentence ‘*X* is an idiot’ is an insult, doesn’t contain any factual information and should be discarded. But if a sentence is ‘*Y* said that *X* is an idiot’ is not an insult any more, because it conveys information about what *Y* said about *X*. Normal text searching methods or looking for obscene expressions will annotate both of the sentences as flame. From this perspective, we outlined a sophisticated NLP application, which can identify a message whether it is an insult or information. This program looks for some key words in a given sentence; interpret the basic meaning according to the semantic information of dependency structure; then apply some predefined rules to distinguish whether it is information or a flame.

Recently, some works have been done in Natural Language Processing for detecting personal opinions, emotions and speculations where flames or abusive messages are considered as very high or extreme intensity level. Our

initial idea was to evaluate the head verb of a dependency structure for a sentence whether it is a factual verb or it reflects the writer's personal opinion or emotion. Then if any insulting word or phrase is found within that sentence then we can classify it as a flame.

This system can identify texts as an insult only bearing insulting words such as *idiot*, *nonsense* or phrases *get a life*, *get lost* etc. The semantic information we are getting by only processing the text what it gives us, we are ignoring surroundings and the context. For example: **“Get that socialist out of my pocket”** – this is a personal attack. If this sentence is to be classified as a flame, we extremely need some world knowledge, we need to know that the word ‘socialist’ refers to a human being and someone wants to get him/her off from the pocket. So, the sentence contains a ‘sense’ of demeaning someone’s (socialist) personal status. We are not being able to classify it as a flame because lack of considering world knowledge.

## 1.2 What is Insult?

Human being perceives Insult as demeaning his or her personal status. Researchers from some sub area of NLP, viewed insult as a super subset of subjective language, where the intensity level of personal opinion or emotions is very high or extreme. That’s why it is called the extreme subset.

## Chapter 2: Literature Review

This chapter first gives a short overview about subjective language and our background study about subjectivity. Then we have some discussion about statistical parsing to show how dependency parsing can be originated from statistical parsing, after that we moved to elaborate discussion of the main key concern of this thesis, the dependency parse tree, which gives a syntactical structure of a sentence as well as gives general semantic information.

### 2.1 Subjective Language

Subjective language is language used to express private states in the context of a text or conversation. Private state is a general covering term for opinions, evaluations, emotions, and speculations. Many natural language processing applications could benefit from being able to distinguish subjective language from language used to objectively present factual information. Information retrieval system should be able to distinguish between factual information (which should be extracted) and non-factual information (which should be discarded or labeled as uncertain). Question answering systems should distinguish between factual and speculative answers. Automatic subjectivity analysis would also be useful to perform flame recognition, email classification etc [3].

Expression of private states in language are classified in three main categories:

- Direct mentions of private states
- Speech events expressing private states
- Expressive subjective elements (Banfield, 1982).

An example of a direct private state is “fears” in (1). An example of a speech event expressing a private state is the one referred to by “continued” in (2).

1. *“The US fears a spill-over,” said Xirao-Nima.*
2. *“The report is full of absurdities,” he continued.*

Sentence (2) also contains an example of an expressive subjective element, namely “full of absurdities”. With expressive subjective elements, sarcasm, emotion, evaluation, etc. are expressed through the way something is described or through particular wording. The subjective *strength* of a word or phrase is the strength of the opinion, emotion, or other private state that it expresses.

Following are some examples of subjective sentences taken from reference paper [3].

- *At every different layers, it’s a fascinating tale* [a book review example].
- *“The cost health care is eroding our standard of living and sapping industrial strength,” complains Walter Maher, a Chrysler health-and-benefits specialist* [a news story].

In contrast, following are examples of objective sentences without significant expressions of subjectivity, which are also taken from the previous paper

- *Bell Industries Inc. increased its quarterly to 10 cents from 7 cents a share.*
- *Northwest Airlines settled the remaining lawsuits filed on behalf of 156 people killed in a 1987 crash, but claims against the jetliner's maker are being pursued, a federal judge said. ["Northwest Airlines Settles Rest of Suits", Wall Street Journal, 11/1/89]*

### 2.1.2 Why flames are 'extreme' subset?

Flames or abusive messages always reflect someone's personal opinions and emotions which has very high intensity level. A very little amount personal opinion bearing 'event' or verb has a significant impact on an insulting or an abusive message. For an example: *Mary confirmed that John doesn't know any behavior.* Here, the event is *confirmed* which reflects a very little amount of personal emotion, can be a factive event for a normal message but in this case the sentence is an insult. Suppose the sentence *Mary confirmed that John doesn't know how to eat spaghetti.* This message is considered as factual. The comparison between these two examples shows that although the verb *confirmed* is considered as factual to convey information for a usual message but in case of abusive messages it is no more considered as factual event.

## 2.2 Background Study in Subjectivity

Since subjective language is a very recent swell of interest of the researchers in NLP, we studied some papers on this topic to formulate our basic concept. Following is the list of some papers, for each of that paper we outlined our extracted concept:

1. *Learning Subjective Language* [3]

This paper described a learning mechanism to learn subjective language from corpora by subjectivity clues which were gathered and tested. Their described clues and their analysis in contrast to subjectivity analysis helped us to gather the basic concept of subjective language.

2. *Annotating opinions in the world press* [7]

This paper describes the manual schemes how to annotate the personal opinion, emotions in newspaper articles. Basic concepts about private states, speech events and expressive subjective elements, nested sources, explicit and implicit speech events. It also describes about inter-annotator agreement of annotating articles.

3. *Annotating expressions of opinions and emotions in language* [8]

This paper describes the main annotation scheme of MPQA corpus. We got very good concept about the annotation scheme.

4. *Annotating attributions and private states* [9]

It has extended features containing attitude and target annotation frames (negative or positive attitude). Integrating with other annotating projects of ‘pie in the sky’, which has proposed another layer of annotation scheme of subjectivity would produce the better result. But it is irrelevant to our thesis.

5. *Automatic annotation of speech events and explicit private state in newswire* [10]

This paper describes how to do automatic annotations of speech events (say, told, according to etc) with some training features using CASS (a partial parser by Abney). We found this useful but we won't be able to include this work in our short thesis.

6. *Instructions for annotating opinions in newspaper articles* [11]

We got significant help about nested sources, inside and outsides of scopes, private states etc.

7. *Opinion finder: A system for subjectivity analysis* [12]

It describes a system and its working procedures for detecting opinions. We have been introduced some essential software- Abney stemmer, Sundance Partial Parser.

8. *Automatic detection of opinion bearing words and sentences* [13]

This paper has given idea that opinions can be annotated by their included words and sentences using antonyms and synonyms. This is also helpful for us but the problem is we didn't get much more time to include this.

9. *Just How Mad Are You? Finding strong and weak opinion clauses*

[14]

It has described some clues to annotate strong and weak opinions clauses (polarity) by combining previous annotations scheme-annotate strong or weak opinions manually and the newer scheme is syntactic clues using dependency parse tree. Here, I have got the concept about dependency parse tree and developed the idea for our thesis. The papers didn't give any idea about our concept, but their description of syntactic clues helped us to develop our own idea.

10. *A corpus study of evaluative and speculative language* [15]

This is just a corpus study. It gives an outline of annotating expressions of negative attitude and machine learning. So, it didn't help us at all.

11. *Recognizing and organizing opinions expressed in the world press*

[16]

The paper stated that information about the source of subjective language is also important. A question answering system will need to evaluate the source of the answer by clustering the opinions with their sources. Applying these information to various documents and topics, to build various groups and sources, and observe how attitudes changes over time. We found that if this is included to our thesis we will be able to produce a significant amount of output. But, the problem is as usual, we won't be able to complete our thesis.



## 2.3 Related Work Done in Flame Recognition

*Smokey: Automatic Recognition of Hostile Messages by Ellen Spertus (1997).*

Smokey is an automatic flame recognition system. Based on the syntax and semantics of each sentence they build 47 element feature vectors and combining it for a message, it can recognize flames. The 47 rules or features are the 47 features for each sentence. From these features the value of the rule classes ( 1 or 0 ) are generated and feed it to the decision tree generator (C4.5) which generates a decision tree. The decision tree classifies a message as flame, maybe or ok. It can correctly categorize 64% of flames and 98% of non flames in a test set of 460 messages which trained from 720 messages.

To study on flames the author of this paper collect messages from the controversial pages of -“NewtWatch”, ”The Right Side of the Web”, “FAIR ( Fairness and Accuracy in Reporting )”. They are media watch group and best known for their criticisms.

Another flame recognition was *e-mail* filtering by *Kaufer (2000)*

### 2.3.1 Our contrast with Smokey

- Our application is a sophisticated NLP application, not an AI application, since learning is not involved here.
- Smokey’s semantic rules are some classification rules, which checks some patterns of words sequence and tries to match the pattern

simultaneously for a message through its decision tree. We are using semantic information to interpret the basic meaning of a sentence, not for pattern matching.

- We are using the semantic information as well as syntactic information where smokey Smokey is message level classification but our system is sentence level classification.
- We didn't make any observation from sociolinguistic point of view as **Smokey** does, to identify messages as an insulting manner. Our system can identify texts as an insult only bearing insulting words such as *idiot*, *nonsense* or phrases *get a life*, *get lost* etc. The semantic information we are getting by only processing the text what it gives us, we are ignoring surroundings and the context.

## 2.4 Statistical parsing

We can use CFGs to parse with, but some ambiguous sentences could not be disambiguated, and we would like to know the most likely parse. We could use a corpus to do that.

### 2.4.1 Basic idea

- Start with a Treebank (we can say bank of trees, e.g. Penn Treebank) which is a collection of sentences with syntactic annotation, i.e., already-parsed sentences.
- Examine which parse trees occur frequently

- Extract grammar rules corresponding to those parse trees, estimating the probability of the grammar rule based on its frequency.

That is, we'll have a CFG augmented with probabilities (PCFG).

## 2.4.2 Probabilistic Context-Free Grammars (PCFGs)

Definition of a PCFG:

- Set of non-terminals (N)
- Set of terminals (T)
- Set of rules/productions (P), of the form  $A \rightarrow \beta$
- Designated start symbol (S)
- function, D assigns probabilities to each rule in P

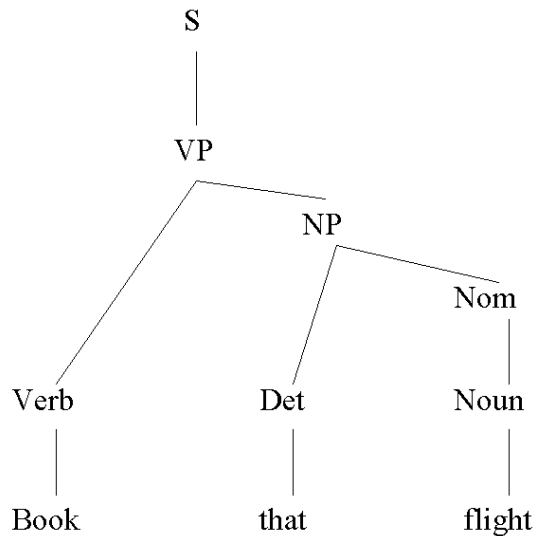
$$D = P(A \rightarrow \beta)$$

## 2.4.3 Estimating Probabilities using a Treebank

- Given a corpus of sentences annotated with syntactic annotation (e.g., the Penn Treebank)
- Consider all parse trees
- (1) Each time we have a rule of the form  $A \rightarrow \beta$  applied in a parse tree, increment a counter for that rule
- (2) Also count the number of times A is on the left hand side of a rule
- Divide (1) by (2)  $D = P(A \rightarrow \beta | A) = \text{Count}(A \rightarrow \beta) / \text{Count}(A)$

## 2.4.4 Using Probabilities to Parse

- $P(T)$  = probability of a particular parse tree  
 = the product of the probabilities of all the rules  $r$  used to expand each node  $n$  in the parse tree



**fig-1**

We have the following rules and probabilities (adopted from figure 12.1, Jurafsky Martin)

- $S \rightarrow VP$  .05
- $VP \rightarrow V NP$  .40
- $NP \rightarrow Det N$  .20
- $V \rightarrow book$  .30
- $Det \rightarrow that$  .05
- $N \rightarrow flight$  .25

$$P(T) = P(S \rightarrow VP) * P(VP \rightarrow V NP) * \dots * P(N \rightarrow flight)$$

$$= .05 * .40 * .20 * .30 * .05 * .25 = .000015$$

So, the probability for that parse is 0.000015. Probabilities are useful for comparing with other probabilities. Whereas we couldn't decide between two parses using a regular CFG, we now can.

### 2.4.5 Obtaining the best parse

- The best parse  $T(S)$ , where  $S$  is our sentence is the tree which has the highest probability.
- We can use the Cocke-Younger-Kasami (CYK) algorithm to calculate best parse
  - CYK is a form of dynamic programming
  - CYK is a chart parser, like the Earley parser

### 2.4.6 Problems with PCFGs

- It's still only a CFG, so dependencies on non-CFG information is not captured.
  - e.g., Pronouns are more likely to be subjects than objects:  
 $P[(NP \rightarrow \text{Pronoun}) \mid NP = \text{subject}] \gg P[(NP \rightarrow \text{Pronoun}) \mid NP = \text{obj}]$
- Ignores lexical dependency information (statistics), which is usually crucial for disambiguation of “PP attachment ambiguity” and “Coordination ambiguity”.

- (T1) America sent [ [250,000 soldiers] [into Iraq] ]
- (T2) America sent [250,000 soldiers] [into Iraq]

“Sent” with “into”-PP always-attached high (T2) probability.

An example of Coordination ambiguity is two parses of the phrase “dogs in houses and cats”

- (T1) [ [NP dogs] in [ NP houses and cats ] ]
- (T2) [ [NP dogs in houses] and [NP cats ] ]

Here T1 is semantically wrong and T2 is correct but both tree results same score. So only PCFG is not enough to disambiguate parse trees, lexical dependency information is also needed.

- To handle lexical information, we’ll turn to lexicalized PCFGs.

## 2.4.7 Lexicalized PCFGs

- Lexicalized Parse Trees
  - Add “headwords” to each phrasal node. Each PCFG rule in a tree is augmented to identify one RHS constituent to be the head daughter
  - The headword for a node is set to the head word of its head daughter
  - Headship not in (most) treebanks
  - Usually *use head rules*, e.g.:

- NP:
  - Take leftmost NP
  - Take rightmost N\*
  - Take rightmost JJ
  - Take right child
- VP:
  - Take leftmost VB\*
  - Take leftmost VP
  - Take left child

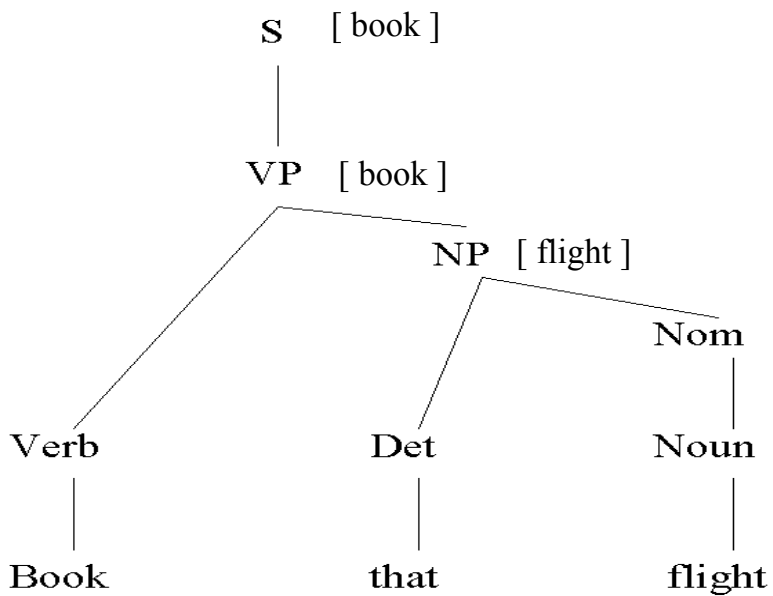


fig-2

### 2.4.8 Incorporating head probabilities

- Previously, we conditioned on the mother node (A):

- $P(A \rightarrow \beta | A)$
- Now, we can condition on the mother node and the headword of A (h(A)):
  - $P(A \rightarrow \beta | A, h(A))$

We're no longer conditioning on simply the mother category A, but on the mother category when h(A) is the head.

- e.g.,  $P(VP \rightarrow VBD NP PP | VP, \textit{dumped})$

### 2.4.9 Calculating rule probabilities

- We calculate this by comparing how many times the rule occurs with h(n) as the headword versus how many times the mother/headword combination appear in total:

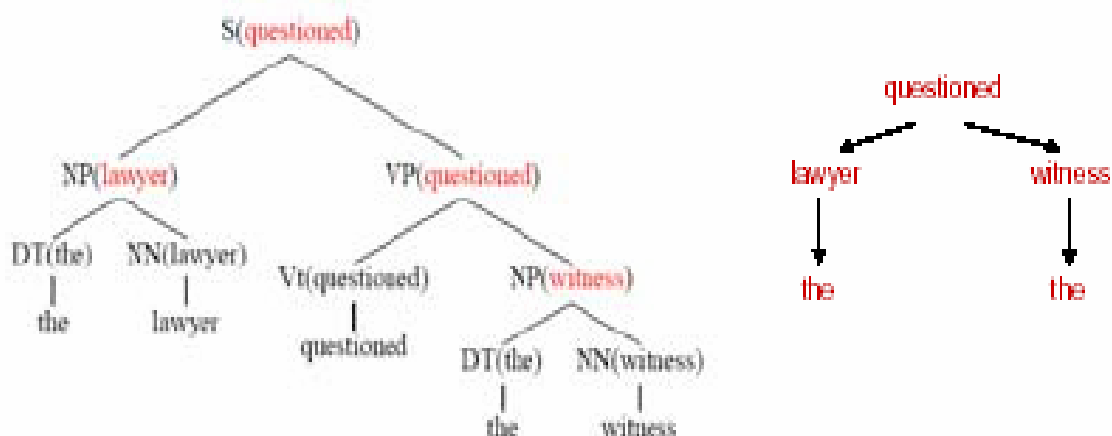
$$P(VP \rightarrow VBD NP PP | VP, \textit{dumped}) \\ = C(VP(\textit{dumped}) \rightarrow VBD NP PP) / \sum_{\beta} C(VP(\textit{dumped}) \rightarrow \beta)$$

### 2.4.10 Adding info about word-word dependencies

- We want to take into account one other factor: the probability of being a head word (in a given context)
  - $P(h(n)=\text{word} | \dots)$



- We condition this probability on two things: 1. the category of the node ( $n$ ), and 2. the headword of the mother ( $h(m(n))$ )
  - $P(h(n)=word \mid n, h(m(n)))$ , shortened as:  $P(h(n) \mid n, h(m(n)))$
  - $P(sacks \mid NP, dumped)$
- What we're really doing is factoring in how words relate to each other
- We will call this a dependency relation later: *sacks* is dependent on *dumped*, in this case



**fig-3:** Lexicalized parsing can be seen as producing *dependency trees*

## 2.5 Dependency Parsing

Modern dependency grammar has been created by French linguistic Lucien Tesnière (1959). Although its roots may be traced back to Panini's grammar of Sanskrit (predecessor of Bangla) several centuries before. In NLP, dependency parse tree is thought as a 'bridge' between syntactic and semantic analysis, since it gives some semantic information as well as syntactic. Some people also argue that it is another version of chunk

parsing, because a very careful observation of a dependency tree will reveal that every subpart of a sentence: *subject*, *object* or *complements* are appeared in different sub trees or under different relation, where each node is dependent on another node. These sub trees or semantically dependent nodes can be thought of as separate chunks.

### 2.5.1 Basic Concepts

In a dependency representation every node in the structure is a surface word (there are no abstract nodes such as NP or VP), but each word may have additional attributes such as its part-of-speech (POS) tag. The parent word is known as the *head*, and its children are its *modifiers*. The observation which derives DG is: In a sentence, all but one word depend on other words. The one word that doesn't depend on any other is called the *root* of the sentence. A typical DG analysis of the sentence *A man sleeps* is demonstrated below:

*A* depends on *man*

*Man* depends on *sleep*

*Sleep* depends on nothing (it is the root of the sentence)

Or, put differently

*A* modifies *man*

*Man* is the subject of *sleep*

*Sleep* is the main verb of the sentence

This is Dependency Grammar. A formulated dependency grammar is given below:

- Capturing relations between words is moving in the direction of dependency grammar (DG)
- In DG, there is no such thing as constituency
- The structure of a sentence is purely the binary relations between words,  $A \rightarrow B$  means that B depends on A

Dependencies are motivated by *grammatical function*, both syntactically and semantically. A word depends on another either if it is a complement or a modifier of the latter. The edge between a parent and a child node specifies the grammatical relationship between the two words (e.g. *subj*, *obj*, and *adj*). In most formulations of DG for example, functional heads or governors (e.g. verbs) subcategorize for their complements. Hence, a transitive verb like ‘like’ requires two complements (dependents), one noun with the grammatical function subject and one with the function object.

In this paper we are using Stanford-Parser version-jdk1.5 for all of the output and figures.

Ex sentence: John likes Italian food.

Tagged output: John/NNP likes/VBZ Italian/NN food/NN

Constituent structure output:

```
(ROOT
  (S
    (NP (NNP John))
    (VP (VBZ likes)
      (NP (NN Italian) (NN food))))))
```

Dependency structure output:

```
nsubj(likes-2, John-1)
nn(food-4, italian-3)
dobj(likes-2, food-4)
```

## 2.5.2 Dependency functions

### 2.5.2.1 Main functions

#### main

main element

The main element of a clause is usually a verb, but in a verbless clause other elements may serve as a head as well.

Ex: a sentence with a verb

*He doesn't **know** whether to send a gift.*

```
nsubj(know-4, He-1)
```

```
aux(know-4, does-2)
```

advmod(know-4, n't-3)  
 aux(send-7, to-6)  
 whether(know-4, send-7)  
 det(gift-9, a-8)  
 dobj(send-7, gift-9)

Ex: a sentence without a verb

*A comprehensive **grammar** of the English **language***

det(grammar-3, A-1)  
 amod(grammar-3, comprehensive-2)  
 det(language-7, the-5)  
 amod(language-7, english-6)  
 of(grammar-3, language-7)

### 2.5.2.2 Verb complementation

#### **nsubj**

nominal subject

The dependency syntax collapses the classes of formal subject and ordinary subject into one. The subject may also be a non-finite clause that-clause, WH-clause, etc.

#### **dobj**

direct object

The notion of object is wider than that in Quirk, comprising essentially all types of second arguments, except subject

complements. The motivation is that the subtypes of second arguments are complementary, i.e. they occupy the same valency slot. There are both simple nominal objects and more complex objects such as a non-finite clause, that-clause, WH-clause or quote structure.

Ex: *John explained that topic*

nsubj(explained-2, John-1)

det(topic-4, that-3)

dobj(explained-2, topic-4)

### **ccomp**

coordinated complement

Subject complement is the second argument of a copular verb.

Ex: *Mary **said** John didn't **go** there*

nsubj(said-2, Mary-1)

nsubj(go-6, John-3)

aux(go-6, did-4)

advmod(go-6, n't-5)

ccomp(said-2, go-6)

advmod(go-6, there-7)

### **iobj**

indirect object

Indirect object corresponds to a third argument. The prepositional

dativ is described accordingly. Again, the syntactic motivation is that the prepositional phrase occupies the same valency slot as the indirect object and is semantically equivalent to it.

Ex: *I gave **him** my address.*

nsubj(gave-2, I-1)

iobj(gave-2, him-3)

dep(address-5, my-4)

dobj(gave-2, address-5)

*What did Pauline give **Tom**?*

*Pauline gave it **to Tom**.*

### 2.5.2.3 Determinative functions

#### **det**

determiner

Central determiners (articles) or a determining pronoun. Successive determiners are linked to each other.

Ex: *This is **an** apple*

nsubj(is-2, This-1)

det(apple-4, an-3)

dobj(is-2, apple-4)

### 2.5.3 Robinson's axiom

Robinson (1970) formulated four axioms to govern the well-formedness of dependency structures, depicted below:

1. One and only one element is independent.
2. All others depend directly on some element.
3. No element depends directly on more than one other.
4. If A depends directly on B and some element C intervenes between them (in the linear order of string), then C depends directly on A or B or some other intervening element.

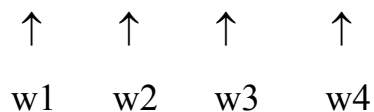
The first three axioms ensure that they shall be trees. Axioms 1 and 2 state that in each sentence, only one element is independent and all others dependent on some other elements. Axiom 3 states that if element A depends on B, it must not depend on another element C. This requirement is referred as *single-headness*. Axiom 4 is called the requirement of *projectivity* and disallows crossing edges in dependency trees.

### 2.5.4 Dependency relation

A mapping  $M$  maps  $W$  to the actual words of a sentence. Now for  $w_1, w_2 \in W$ ,  $\langle w_1, w_2 \rangle \in R$  asserts that  $w_1$  is dependent on  $w_2$ . The properties of  $R$  treeness constraints on dependency graphs as Robinson's axioms.

Ex: Mary loves another Mary





here,  $M(w_1 \dots w_4 \in W)$

1.  $R \subset W \times W$
2.  $\forall w_1 w_2 \dots w_{k-1} w_k \in W: \langle w_1, w_2 \rangle \in R \dots \langle w_{k-1}, w_k \rangle \in R: w_1 \neq w_k$   
(acyclicity)
3.  $\exists! w_1 \in W: \forall w_2 \in W: \langle w_1, w_2 \rangle \notin R$  (rootedness)
4.  $\forall w_1 w_2 w_3 \in W: \langle w_1, w_2 \rangle \in R \wedge \langle w_1, w_3 \rangle \in R \rightarrow w_2 = w_3$  (single-headedness)

### 2.5.5 Stanford dependency parser by Dan Klein

This parser uses the feature of Collins's parser. Michael Collins in his 'Head Driven Statistical Parser' showed mapping of his statistical parser to the dependency relation sets. Dan Klein's Stanford parser deals with tagged words: pairs  $\langle w, t \rangle$ . First the head  $\langle w_h, t_h \rangle$  of a constituent is generated using 'Collins head finder' method, then successive right dependents  $\langle w_d, t_d \rangle$  until a 'stop' token is generated, then successive left dependents until 'stop' token is generated. It supports three formats for output:

1. dependencies
2. typedDependencies
3. typedDependenciesCollapsed

For example: *Factory payrolls fell in September.*

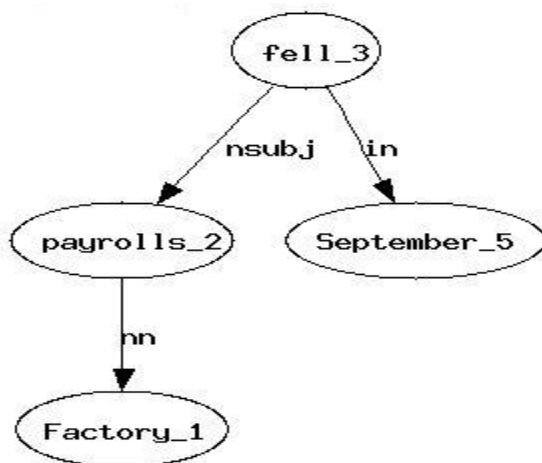
Tagged output: Factory/NN payrolls/NNS fell/VBD in/IN September/NNP

Dependency structure:

nn(payrolls-2, Factory-1)

nsubj(fell-3, payrolls-2)

in(fell-3, September-5)



**fig -4**

First, *fell*-VBD is chosen as the head of the sentence, then, *in*-IN to the right is generated, which then generates *September*-NN to the right, which generates ‘stop’ token on both sides. Then return to *in*-IN, generate ‘stop’ to the right, and so on. The above output is the ‘typedDependenciesCollapsed’ format of Stanford dependency parse tree. This ‘typedDependenciesCollapsed’ doesn’t make separate nodes for the words, which are obvious in any dependency relation in a sentence; instead it makes it a relation between two prominent words. In the above example the

preposition ‘in’ is used as a relation or dependency function between the words ‘fell’ and ‘September’.

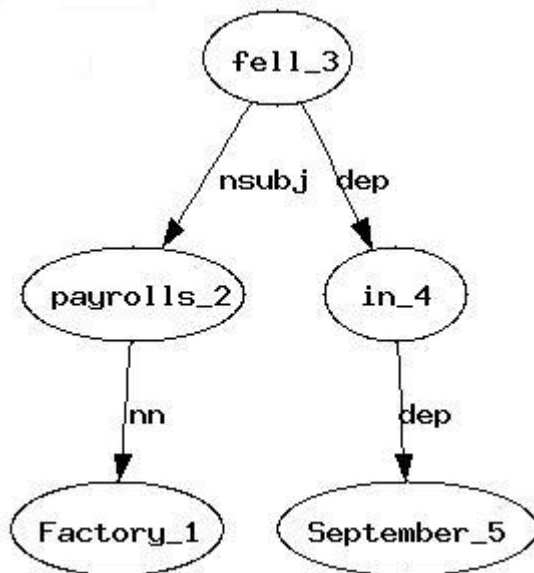
For example, only ‘typedDependencies’ format of the above sentence will be:

nn(payrolls-2, Factory-1)

nsubj(fell-3, payrolls-2)

dep(fell-3, in-4)

dep(in-4, September-5)



**fig -5**

Example shows that it makes a separate node ‘in’ between ‘fell’ and ‘September’, which can be used as a relation to make the tree shorter in depth. This thesis uses the ‘typedDependenciesCollapsed’ format as well because we don’t need to look at every word to extract necessary information.

## Chapter 3: Methodology

This chapter describes the strategies of how do we adjusting a sentence and feeding it to the dependency parser and then interpreting the semantic information to distinguish flames and information. Methodology is classified in two categories:

1. Preprocessing
2. Processing

### 3.1 Preprocessing

At preprocessing part we have to process each sentence in such a way so that the dependency structure will be able to give every necessary information (such as *subject*, *object*, *complements* and other necessary semantic information). Since it is not possible always to construct a sentence in that way, we are leaving some works for the processing part also. But we need to ensure that we have constructed a sentence to give it all of its clues as much as possible.

We pointed out two constraints of the parser to solve our purpose:

- The parser can give the best output and semantic information for a simple sentence. So, we need to split up a sentence into its corresponding clauses and give each clause a simple construction.

- In a simple sentence, an event or verb must follow its subject. If its subject follows an event, we need to swap subject and the event.

The steps of our preprocessing part are depicted below:

1. Separate each sentence one per line. We used `opennlp 1.3.0` for this separation, although sometimes it makes confusion in case of dot or full stop. But we are omitting that mistake since it has the better performance because of its trained feature.
2. Replace the factive event ‘according to’ by ‘implied that’ and swap the subject and the event.

For example: *According to Mary, john didn't go downstairs.*

After replacing ‘According to’ by the event ‘implied that’ the sentence will be: *implied that Mary, john didn't go downstairs.* This is not a correct sentence structure. So, we need to swap the subject and the event.

The final sentence will be like this: *Mary implied that John didn't go downstairs.*

Another example: *According to her, john didn't go downstairs.*

Here, the subject is ‘her’, a pronoun. So, ‘her’ must be replaced by ‘she’ to give the sentence a correct form. Here are the mappings:

*him -> he*

*her -> she*

*me -> I*

*us -> we*

*them -> they*

So, the final sentence becomes: *she implied that, john didn't go downstairs.*

The event 'according to' can be anywhere in a sentence. So, it has to be handled accordingly.

3. Punctuation marks (“”) are used to give a unit scope of speaker's speeches. One or more sentences could be in a scope. For example considering a paragraph:

*“John is waiting for the lift. He didn't go downstairs,” replied Mary while talking with Lisa.*

After first two step this paragraph will be like this:

*“John is waiting for the lift.*

*He didn't go downstairs,” replied Mary while talking with Lisa.*

Here, the speaker is 'Mary' and event is 'replied'. We need to place the agent 'Mary' followed by the event 'replied' at the beginning of the punctuation, where the scope begins and the tail '*replied Mary*

*while talking with Lisa*’ will go to the next line. In the tail here, agent ‘Mary’ and event ‘replied’ needed to be swapped. After applying above operations, the final output is:

*Mary replied “John is waiting for the lift.*

*He didn’t go downstairs.”*

*Mary replied while talking with Lisa.*

This makes a sentence containing clauses simpler than previous. One more condition we have to check that if the sentence is like this:

*“John is waiting for the lift. He didn’t go downstairs,” replied Mary.*

Here, no word is following ‘Mary’. So, we don’t need to place the tail *replied Mary* to the next line. Just cut the tail, swap the agent and event, place it to the beginning.

*Mary replied “John is waiting for the lift. He didn’t go downstairs.”*

Another example: *“John is waiting for the lift.” replied Mary, adding “He didn’t go downstairs.”*

This sentence is processed like this:

*Mary replied “John is waiting for the lift.”*

*adding “He didn’t go downstairs.”*

That’s it. The rest of it will be done at processing part.

4. Tag each sentence using Stanford parser and keep it in memory. Since Stanford parser is a probabilistic parser, give it a full sentence before separate it into clauses.
5. Separate each sentence into clauses. Below is the list of clause separators in a sentence:

, (comma)

and

or

nor

but

because of

because

although

since

otherwise

in order to

while

when

where

whether

whose

who

which

; (semi-colon)

: (colon)



- (dash)

-- (double-dash)

Once we get these separators we split up the sentence. But, for some separators we need to have some special consideration. These are:

, (comma), and, who, which.

Below we described sequentially why these are taken as special consideration.

, (comma): Consider a sentence – *Italy, France, and Germany have played really well in the World Cup '06.*

Here, , (comma) is not used to separate clauses. It just separating proper nouns or can be pronouns. The checking is to see the words on both sides of comma have same tag except verb. Moreover, the word *and* is also separated, so check the word just after the conjunction *and* also. We are not splitting if these checking are true. So, above sentence will not be splitted.

Noun phrases can also be separated by comma.

Suppose a sentence: *President of the club, John, and Finance Minister was present at the meeting.*

Here, *President of the club* is a noun phrase, which is separated by comma with *John*. So, this sentence will not be splitted as well, since we are only splitting sentence into clauses.

Now, taking another example: *John Smith, president of the sports*

*club, said “We will not tolerate it anyway.”* After separate it by comma, it will be like following:

*John Smith*

*<, > president of the sports club*

*<, > said “We will not tolerate it anyway.”*

The comma between angle brackets shows that we are adding every separator in angle brackets in front of a separated clause. Here, the concern is the third clause which is started by speech event verb *said*. It clearly shows that the subject of this speech event is *John Smith*, which is in a separate clause and in the first line. The rule is, at first check the first line to see whether it is only a noun phrase, if it is take it as a nominal subject. Now, if there is any clause started with a verb, put that nominal subject in front of that verb. Here, the verb is *said*.

Then above example will be:

*John Smith*

*<, > president of the sports club*

*<, > John Smith said “We will not tolerate it anyway.”*

Now, consider another example: *We will not tolerate it anyway, because we have to win the match, president of the club John Smith said yesterday.*

After separated by comma:

*We will not tolerate it anyway*  
 <, > *because we have to win the match*  
 <, > *president of the club John Smith said yesterday.*

The third clause here contains a speech event ‘*said*’, and its subject is ‘*president of the club John Smith*’. It shows that first and second clause are the speeches of *John Smith*. Here, the mechanism is, put punctuation marks (“”) at the beginning of the first line and at the end of the previous line where the speech event ‘*said*’ found, to put those clauses in a unit scope. Then make the adjustment for punctuation marks as described in step 3.

Then this example will be:

*president of the club John Smith said “we will not tolerate it anyway*  
 <, > *because we have to win the match”*  
 <, > *president of the club John Smith said yesterday.*

Comma just after the speech event is omitted. Ex: *She said, “There will be no chance.”* This sentence will not be splitted.

and: In case of the separator ‘and’, the checking here is to whether two same tagged words are separated by ‘and’ or not. If those two words have similar tagging then we are not splitting the sentence by ‘and’. As for example: *I eat rice, meat and fish*. In this example *meat* and *fish* both have similar tag (common noun). As per rule no splitting occurred.

who and which: These two separators have same significant, that is these are considered as a same. Two examples are given below:

*The speaker here is John Smith, who is also president of the club.*

*The sports club, which has John Smith as a president.*

At the first example, the separator 'who' refers to the speaker 'John Smith' and at the second clause 'which' refers to 'The sports club'. Both of the noun phrase *John Smith* and *The sports club* are at just before the separators *who* and *which*. The mechanism is, split up a sentence by separators and put the two noun phrase just at beginning of the next separated clause.

For above two examples next processes are given below:

After splitting:

*The speaker here is John Smith*

*<who>is also president of the club.*

*The sports club*

*< which> has John Smith as a president.*

After placing the noun phrases:

*The speaker here is John Smith*

*<who>John Smith is also president of the club.*

*The sports club*

< which> *The sports club has John Smith as a president.*

Of course, the observation made here is for typical cases. Some cases, it is not possible to match these conditions. For those cases, we just leaving it for the processing part just after splitting.

Our preprocessing part is done. Now we are ready to move on to our processing part.

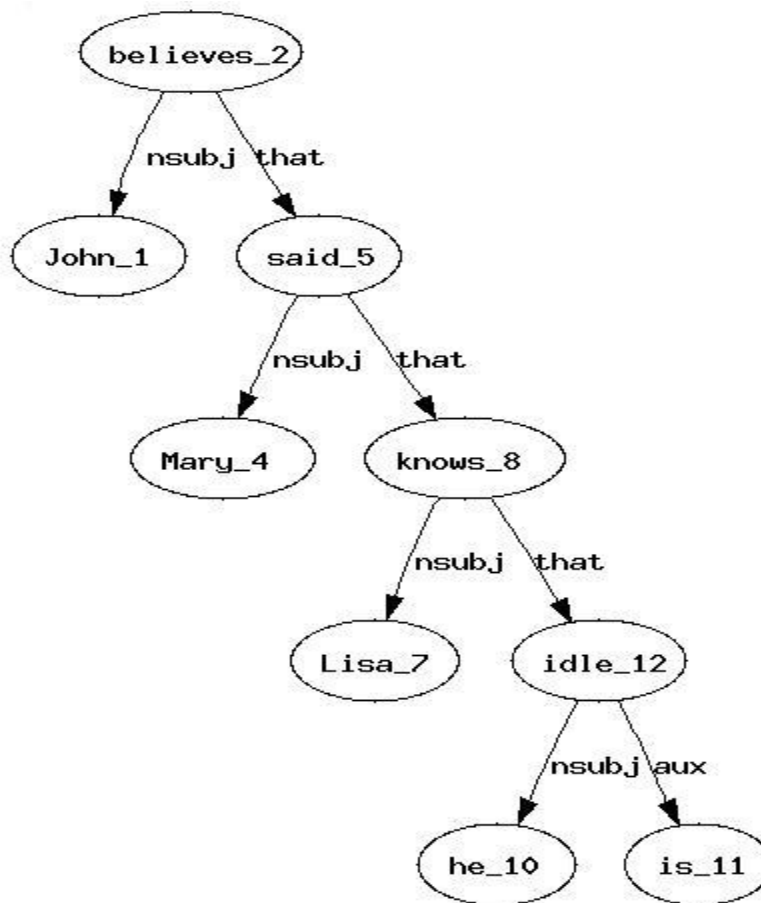
## 3.2 Processing

Now, we have to feed each clause to the dependency parser. That clause we have made in preprocessing part. We have marked each new sentence. If we are getting any nonfactual insulting or abusive message in a clause, we are just printing out the whole sentence. We are keeping a stack, after traversing each dependency tree we are getting some nested sources that is *agent*, *experiencer* with their corresponding *events* or verbs.

Following is an example:

*John believes that Mary said that Lisa knows that he is idle.*

This sentence contains four nested sources – *John*, *Mary*, *Lisa* and *he*  
Corresponding events are: *believes*, *said*, *knows*, and *is*.



**fig -6:** The dependency structure of the above example

The **fig-6** shows that after traversing the tree we can get the nested sources.

experiencer: John
event: believes
agent: Mary
event: said
experiencer: Lisa
event: knows
experiencer: he
event: is



**fig -7:** Corresponding stack contains the nested sources of the tree.

Since, the stack grows downwards, the top of the stack is at the bottom, and we are following this convention for the rest of this paper.

Stack shows the nested sources from top to bottom. We are considering ‘agent’ and ‘experiencer’ by their corresponding events. Once we get an event which is a speech event, we are considering the corresponding subject as an agent, otherwise that will be an ‘experiencer’. This is different from the ‘thematic’ role in semantic analysis. The [fig 4](#) shows that an agent can be in a nested scope of an ‘experiencer’, agent *Mary* is in the scope of the experiencer *John*.

### 3.2.1 Stack Manipulation

This section described how the stack, where every source and its nested sources with their corresponding events are stored, is being manipulated before and after processing of each clause. So, first we gave an overview before go to the actual processing.

Steps are:

1. Before feeding a clause or a sentence to the parser, we are checking the first word of that clause whether it is a verb.

If verb is found:

- Check whether it is a new sentence, or whether this clause was separated by *while* or *because*. If the checking returns true then take the last agent from the stack not the experiencer.

Separators *while* and *because*, we call them scope detachers. For example: *Mary said John is an idiot while talking with Lisa.*


After separating by *< while >*

*Mary said John is an idiot*  
*< while > talking with Lisa.*

The second clause starts with a verb '*talking*'. So, if the question is 'Who was talking with Lisa?' The answer should be 'Mary' not 'John'. This process is elaborated below:

First clause: *Mary said John is an idiot*

agent: Mary
event: said
experiencer: John
event: is



**fig –8:** Corresponding stack of the example

Second clause: *talking with Lisa*

Since this clause starts with the verb *talking* and the separator was *<while>* then we should take the last agent *Mary*. So, this clause will be: *Mary talking with Lisa.*

This rule is applicable for separator *<because>* and if a new sentence begins. Otherwise we have to take the last 'subject'



from the stack. This subject can be *experiencer* or can be an *agent*. For the above example if the separator was *<and>* we had to take the last experiencer which is *John*. The second clause would become: *John talking with Lisa*.

If the first word is not verb

- Then we have to check whether this sentence is not a new sentence (a separated clause) and is separated by *<,>* (comma). If the clause is separated by *<,>* (comma) and doesn't contain any verb, then we are taking the last subject (agent or experiencer) with the corresponding event and placing it at beginning of the clause. An example is given below:

*I ate fresh rice, small fish, and green vegetables.*

After separating by separator *<,>* and *<and>*

*I ate fresh rice*

*<,>small fish*

*<,><and> green vegetables.*

For the first clause: *I ate fresh rice*

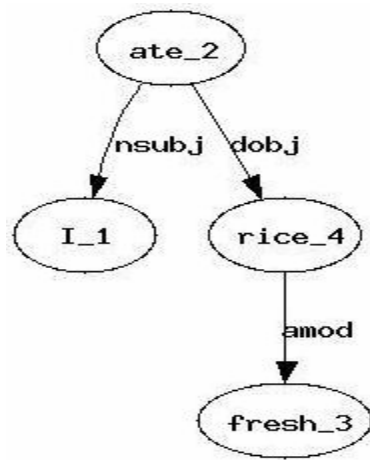


fig-9

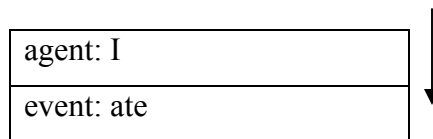


fig-10: Stack from the tree

Now, the second clause: *small fish*

Since, it has no verb and was separated by <,>, take the last subject with corresponding event (Here, subject is *I* and event is *ate*).

So, second clause will be *I ate small fish*. The third clause will be processed at the same way, since the second clause will push the agent *I* and event *ate* to the stack. Then the third clause *green vegetables* will become *I ate green vegetables*.

2. Detach previous scopes from the stack according to the rules. A scope can be opened by an ‘agent’ or by an ‘experiencer’. First check if the current sentence is a new sentence.

If this is a new sentence

- A new sentence can be within a scope. That is, several new sentences can be within a scope defined by a pair of punctuation marks (“”). If this is the case, detach all the scopes except the scope opener. Following example will make it clear:

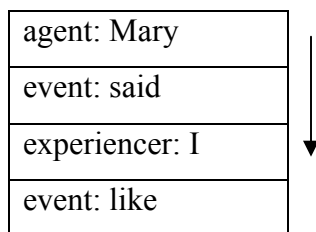
*Mary said, “I like fish and vegetables. I hate meat.”*

After preprocessing:

*Mary said, “I like fish and vegetables.*

*I hate meat.”*

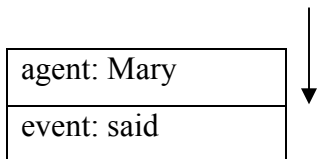
Here, for the first sentence the stack will be:



**fig-11**

Now, the second sentence *I hate meat.”* This new sentence is in the scope of agent *Mary*, who is the scope opener of the current

scope. So, detach all of the scopes except the scope opener *Mary*. After this, the stack will be like the following:



**fig-12**

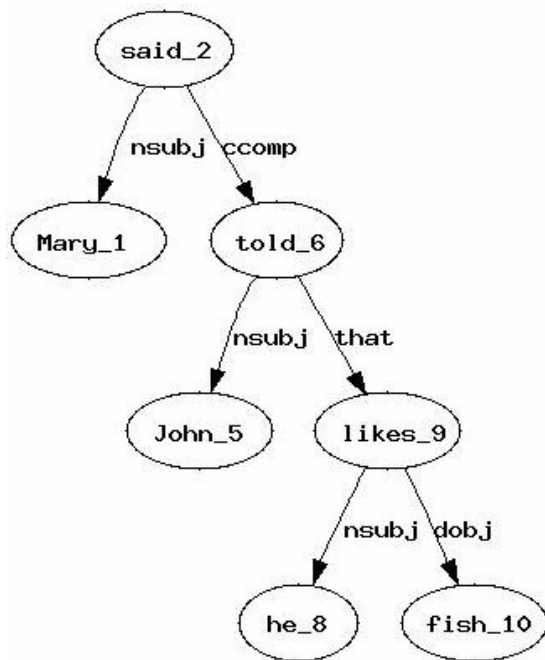
If this is not the new sentence

- Check if any scope is currently open. Then we have to check the separator. If the separator is *<while>* or *<because>*, detach all the scopes except the current scope opener. If there is any another separator, such as *<and>*, then check if there is any agent just after the current scope opener. If this, is then detach all the scopes only after that agent, otherwise if it was an experiencer then detach scopes except the current scope opener. An example: *Mary said, “John told that he likes fish and he hates meat.”*

After preprocessing:

*Mary said, “John told that he likes fish  
<and> he hates meat.”*

First clause: *Mary said, “John told that he likes fish*



**fig-13:** dependency structure for the first clause

agent: Mary
event: said
agent: John
event: told
experiencer: he
event: likes

↓

**fig-14:** Corresponding stack for fig-10

Agent: Mary
Event: said
agent: John
Event: told

↓

**fig-15:** Stack of fig-10 after detaching scopes

The second clause: *he hates meat.*” Now, this clause is not

separated by *<while>* or *<because>*, and this is within a scope. The scope opener is *Mary*, after the scope opener there is an agent *John*. So, the stack will be shortened just after the event *told* of agent *John*. **Fig - 15** exactly shows that next state of the stack. In case of *<while>* or *<because>* the stack will be shortened just after the event of *Mary* which is *said*.

- If no scope is open, the scenario will be the same as described except there will be no scope opener. For the above example, if the separators are *<while>* or *<because>*, the stack will become empty before processing the second clause.

If this is a new sentence, then we just need to check whether this sentence is within a scope. If a scope is open, detach all other except the current scope opener with its events. Otherwise if no scope is open, make the stack empty. For the above example, if the second clause “*he hates meat.*” is completely a new sentence and not within the scope of *Mary*, the stack will be empty as well.

### 3.2.2 Marking phase

For a given sentence, some words or phrases are marked in this phase. These marked words or phrases will be evaluated at the detection phase. This marking is done for each word in the sentence with its tag.

First job is to make all the insulting phrases to one word by putting a ‘-‘ between the words of that phrase. Ex: *Get a life John*. After making the phrase *Get a life* to one word the sentence will become *Get-a-life John*.

The next job is to mark the sentence for each word. Here is the list, which tells what the markings that we are doing. All insulting words and phrases are marked with a ‘\*’ mark.

\**<phrase>*: Any insulting phrase such as *get a life, get lost* etc.

\**<word>*: Any insulting word such as *idiot, nonsense* etc.

\**<comparable>*: If a human being is compared to these words such as *donkey, dog* etc.

*<attribute>*: These are the personal attributes of human being such as *behavior, manner, truthfulness* etc.

*<factive>*: All are the speech events such as *said, told, asked* etc.

*<evaluative>*: These are the verbs, which are used to evaluate a human being’s personal attribute such as *know, show, have, has, expressed* etc.

*<modifier>*: These are also verbs, which are used to modify another verbs such as *should, would, must* etc.

<*comparableVerb*>: These verbs are used to compare a human being with the *comparable* that we have described. Generally, these verbs are auxiliary verbs, which are *is, are, was, and were*.

For the above marking phase we have lexicons for these kinds of words or phrases.

An example: *You need to get a life John!*

*You should know how to behave.*

Two sentences. After marking the first sentence it will become like the following list:

*You/PRP*

*need/VBP*

*to/TO*

*\*<phrase>get-a-life/VB*

*./.*

After marking the second sentence:

*You/PRP*

*<modifier>should/MD*

*<evaluative>know/VB*

*how/WRB*

*to/TO*

*<attribute>behave/VB*

*./.*



For every clause we are marking it and storing in the memory.

### 3.2.3 Tree Annotation

Each clause is given as an input to the Stanford dependency parser. It gives an output as a bracketed form. For example: *You should know how to behave*. The output is like following:

```
nsubj(know-3, You-1)
aux(know-3, should-2)
ccomp(know-3, how-4)
aux(behave-6, to-5)
dep(how-4, behave-6)
```

From this output we are building the tree. We keep four main properties of each node: *label*, *tag*, *word no* and *edge from parent*. Suppose we are building the node for the word *You* at the first line of the above output:

```
label: You
word no: 1
edge from parent: nsubj
These are the basic properties.
```

Now for each node, we also added some extra boolean properties:

```
insulted
factive
```

*comparable*

*comparableVerb*

*phrase*

*evaluative*

*negative*

*attributive*

*modified*

For some of the words in a sentence or a clause, all of the above boolean properties can be checked to be matched, since in section 3.2.2 we have marked some words with its tag, except the property ‘negative’. Suppose the word *know*, which is marked as *<evaluative>* in the ‘Marking Phase’, can set its ‘evaluative’ property to true while building a node for it.

### 3.2.4 Detection

This is the last part, in fact the main part. We have some predefined rules for the detection. The rules are described below. Each rule we are describing by giving an example sentence and its corresponding dependency tree. This section also described how we manipulated the stack by pushing frames. A single frame contains an ‘agent’ or an ‘experiencer’ with its corresponding event or verb, while traversing the tree.

While visiting a node we must have two information:

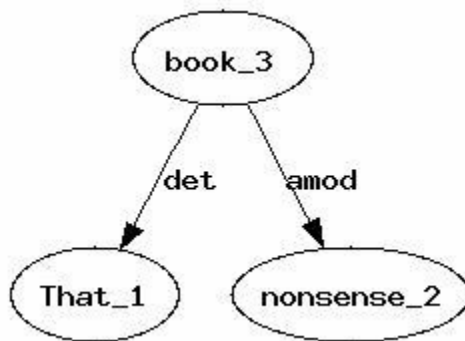
- The current root node.

- The relation, that means which sub tree we are traversing:  
suppose relation ‘nsubj’ indicates that we are traversing the subject part of the current root node, similarly ‘dobj’ indicates that we are traversing object part of the current root node.

The rules are:

1. If a dependency structure doesn't contain any verb as any of it's root, then only search for any insulting word. If found then set the ‘insulted’ property of the root node to true.

Ex: *That nonsense book.*



**fig –16**

The root node is ‘book’, it’s ‘insulted’ property will be set to true since the word ‘nonsense’ is found as it’s modifier.

2. Traverse the subject part of the tree. If found any insulting word or phrase, set the ‘insulted’ property of the current root to true. The subject will become ‘experiencer’, no matter what the root verb is (factive or nonfactive).

Ex: *Only coward says that great.*

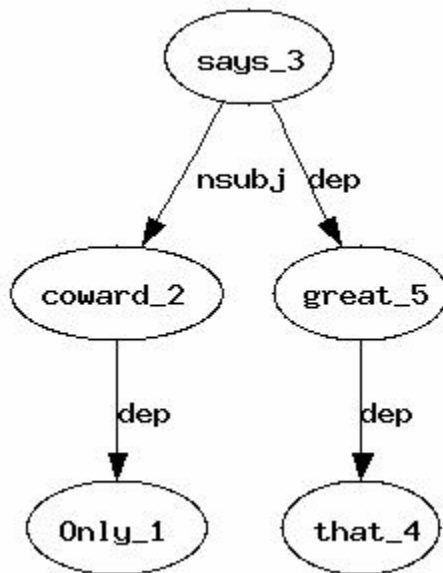


fig -17

In the figure, the root verb is *says* and the subject is *coward*, so the subject itself has an insulted property. The subject here is an experiencer although the verb is a speech event.

3. If the relation between the verb and an insulting word is “dobj” (direct object), or “iobj” (indirect object), “with” or “to” then set the root to be insulted and the subject will be an ‘experiencer’.

Ex: *Mary always says that nonsense.*

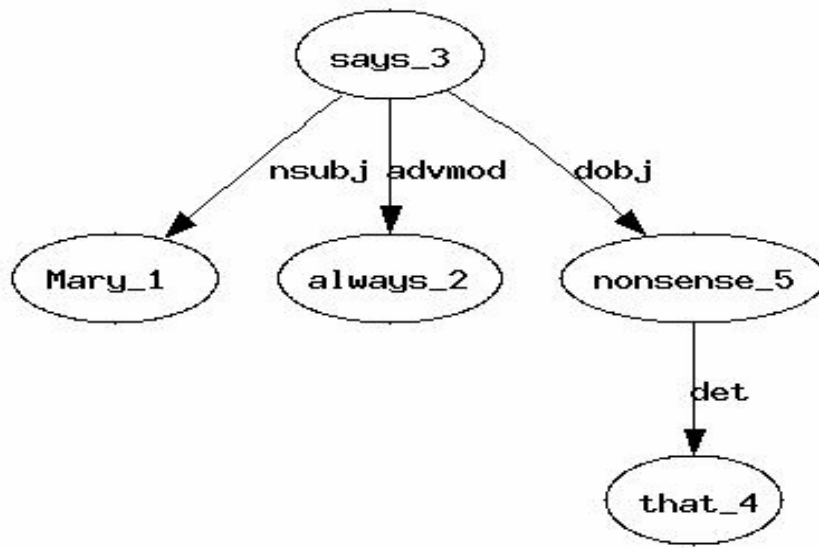


fig-18

The word ‘nonsense’ is the direct object of the verb ‘says’, so subject ‘Mary’ will be an experiencer, not agent.

4. If the root verb has the negative modifier, then check its children. If any of its children has its label “only” then root will be insulted, otherwise not.

Ex: *He is not only an idiot*

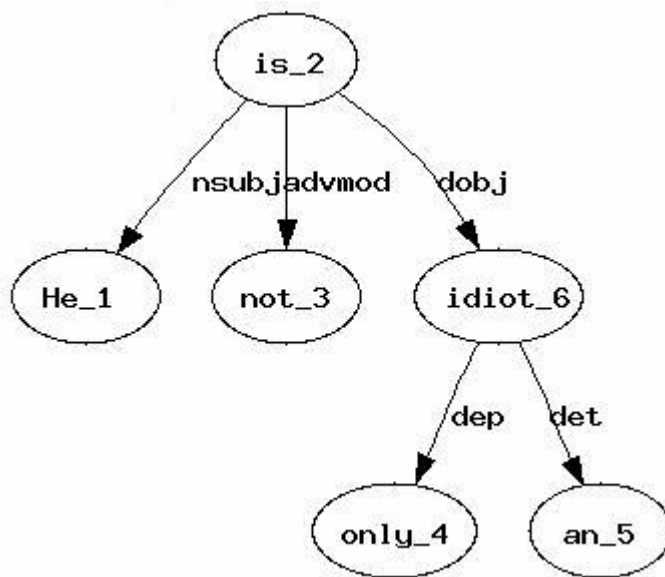


fig-19

The insulting word 'idiot' has a child which label is 'only'. So, the negative verb 'is' has no more an negative impact on 'idiot', root of this tree will be insulted as well.

Ex: *He is not an idiot.*

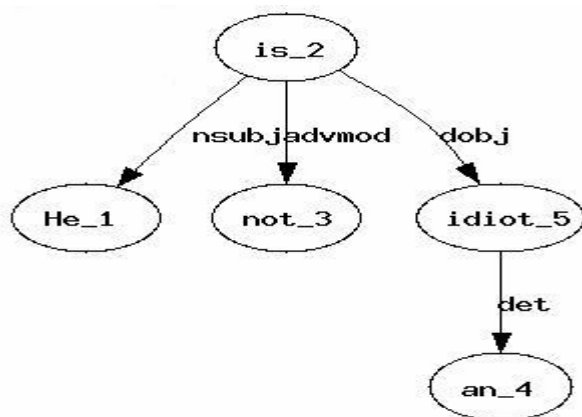
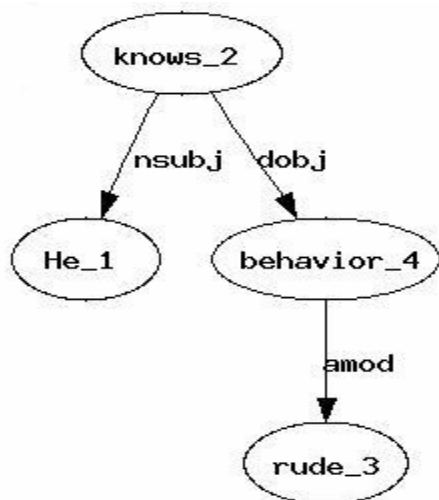


fig-20

Figure shows that the verb 'is' has a negative modifier 'not'. So, the negative property of the root verb will be set to true. Since, verb becomes negative it has a negative impact on the insulting word 'idiot'. So, the insulted property of the root will be false.

5. If the root is 'evaluative' but not 'negative', set the root to be insulted.

Ex: *He knows rude behavior.*

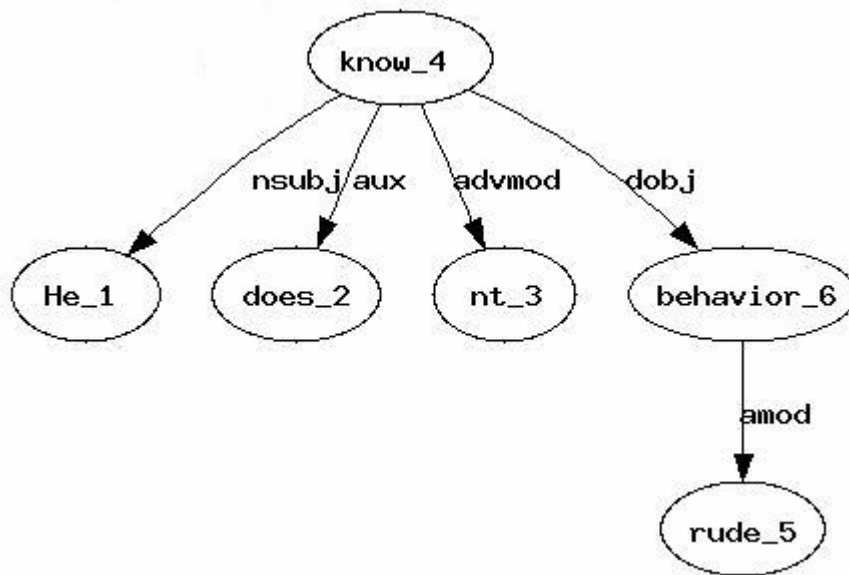


**fig -21**

The root verb is *know* which is evaluative and not negative because it doesn't have any negative node as a modifier, the root will be insulted since 'rude' is an insulted node.

6. If the root is evaluative and also negative, it will not be insulted.

Ex: *He doesn't know rude behavior.*



**fig –22**

Since the root node is ‘evaluative’ and also ‘negative’, it has the negative impact on the insulted node ‘rude’. The root node will not be ‘insulted’.

7. If the property of an insulted node is ‘comparable’ then check its property ‘edge from parent’. If that is ‘as’, ‘like’ or ‘to’ then it is an insult. If that is not then check whether it is ‘dobj’ and the root verb is ‘comparable’ verb, set the root’s insulted property to true.

*Ex: He played like a donkey.*



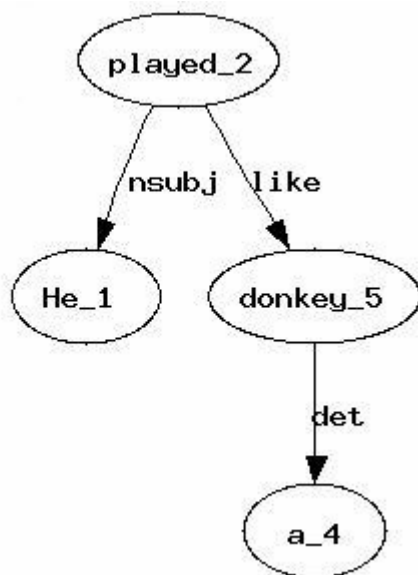


fig -23

Here, the ‘edge from parent’ of the node ‘donkey’ is ‘like’.

Ex: *He is a donkey.*

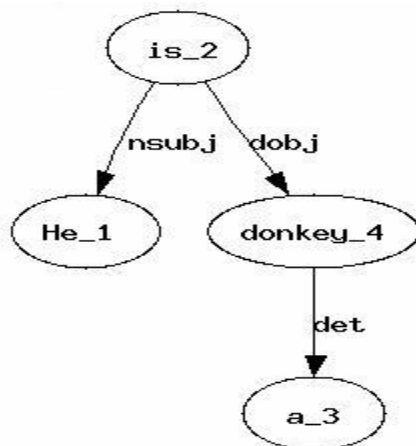


fig -24

Here, the ‘edge from parent’ of the node ‘donkey’ is ‘dobj’.

8. If the property of an insulted node is 'phrase' then check only the root verb whether it's 'factive' property is true. If that is not true then set it is an insulted.

Ex: *Mary confirmed that he should get a life.*

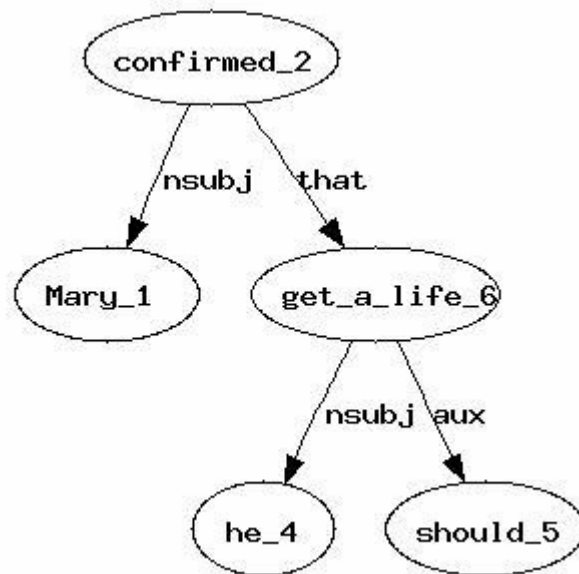


fig -25

The root node is 'confirmed' and its 'factive' property is false, because it is not a speech event. So, its 'insulted' property will be true because it is an insult.

Another ex: *Mary said that he should get a life.*

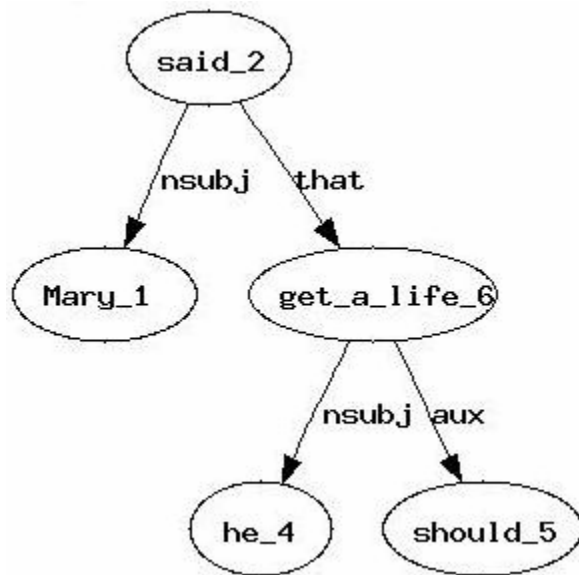


fig -26

The root node 'said' is 'factive' since it is a speech event. So, it is not an insult.

9. If the property of the node is 'attributive' then we got sequentially two checking. First checking is whether the root node is 'evaluative' or 'comparableVerb'. If these checking return true then next checking is whether the root node is 'negative' or it is 'modified'. If these also return true then set 'insulted' property of this root to be true.

Ex: *John doesn't know any behavior.*

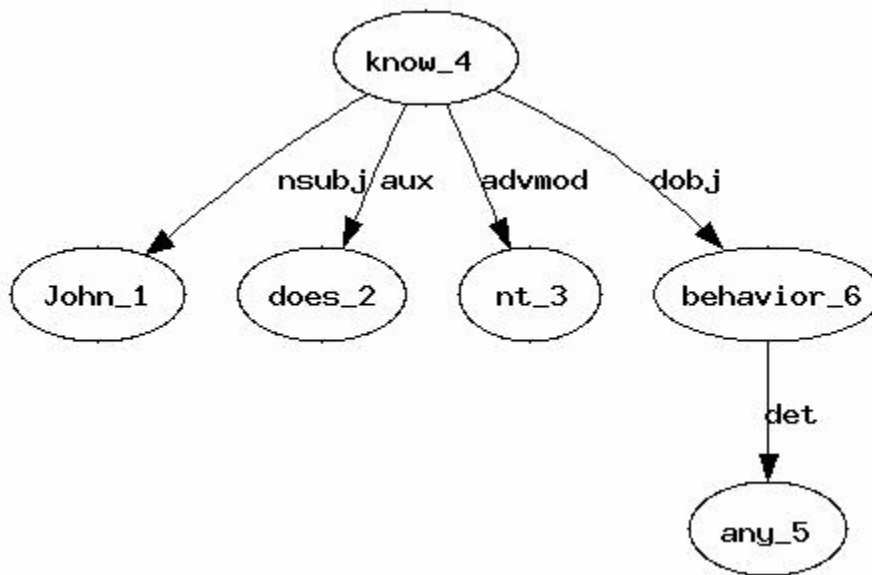


fig -27

The property of the node ‘behavior’ is attributive. Then check the root node. The root is an ‘evaluative’ verb, next it is a negative since it has a negative modifier “n’t”. This sentence is a flame or an insult (negative evaluation of someone’s personal attribute).

*Ex: John should know behavior.*

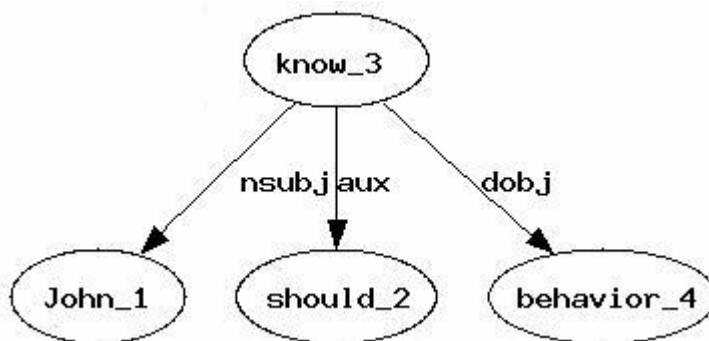
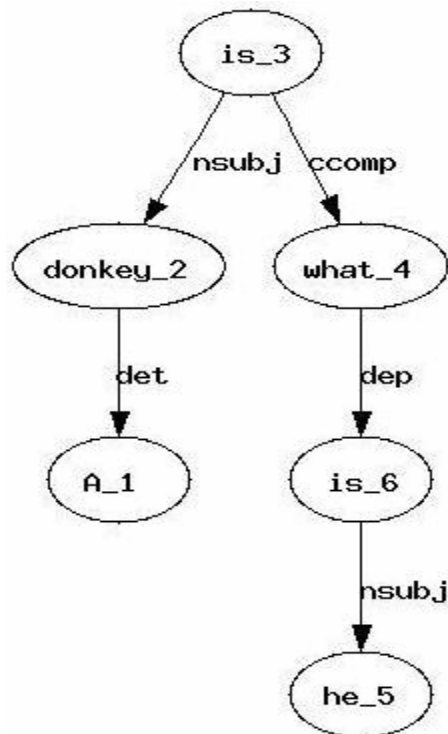


fig -28

Here, the root verb is ‘evaluative’ and it has auxiliary modifier ‘should’. So, the root is ‘modified’ and the sentence is an insult (a doubt, whether *John* knows behavior or not).

10. If the tag of the current node is ‘proper noun’ or ‘pronoun’, see whether property of the subject was ‘comparable’ and the root verb is ‘comparableVerb’. If these conditions are true the root of this tree will be insulted.

Ex: *A donkey is what he is.*



In this figure once we have reached at the node ‘he’ which is pronoun we need to check what was the main subject. Since, the subject here was ‘comparable’ and already been visited, and the root verb is also ‘comparable verb’ we can set

the root as an insulted.

**fig -29**

11. When the current node’s ‘get edge from parent’ contains the string ‘subjpass’ then this subject will be considered as an ‘experiencer’.

Ex: *John was told as an idiot by Mary.*

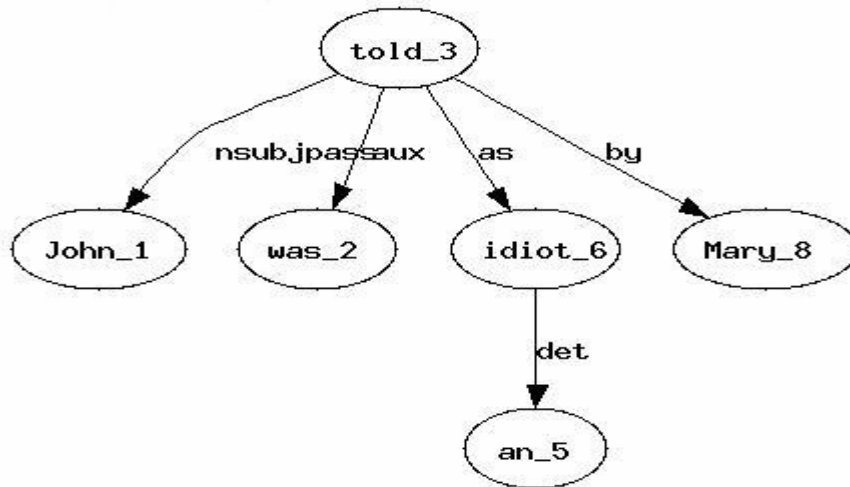
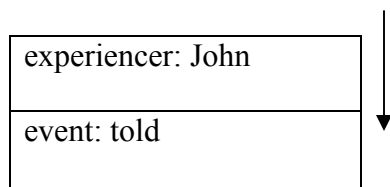


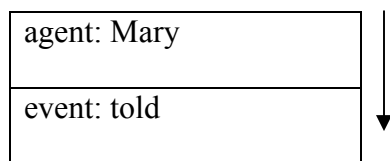
fig -30

12. If the tag of the current node is 'proper noun' or 'pronoun' and the 'edge from parent' is 'by', then the last subject of the stack has to be changed. Before making that change, check whether the root verb is 'factive'. If it is then the last subject will be an 'agent' other it will be an 'experiencer' and the subject will be changed to the current node's label.

In the above example as soon as we visit the node 'was' we will push 'John' as an experiencer into the stack with the event 'told'. Next when we visit 'Mary' the last subject which was 'John' as an experiencer will be changed to 'Mary' which will be an 'agent'.



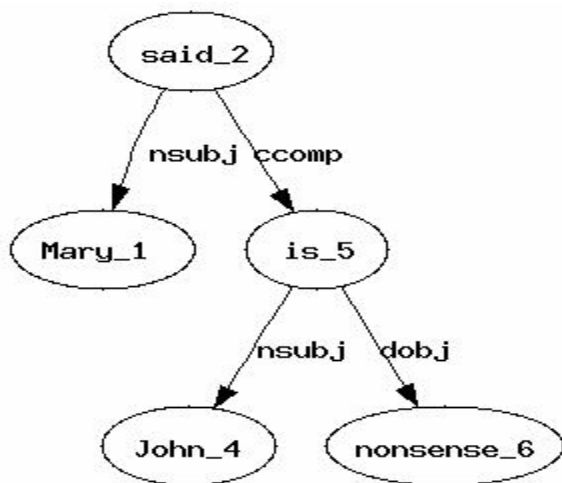
**fig –31:** stack before change



**Fig – 32:** stack after change

- 13.If ‘edge from parent’ of the current node is ‘that’ or ‘ccomp’ and any of its child node’s ‘edge from parent’ is ‘nsubj’ then the current root will be changed to this current node. Before changing the current root push the previous subject with that corresponding verb. Subject will be pushed as an ‘agent’ or an ‘experiencer’ depends on whether the verb is ‘factive’ or ‘non-factive’.

Ex: *Mary said John is nonsense.*



**fig –33**

In above figure node 'is' is the nested sub root of 'said' since it's 'edge from parent' is 'ccomp' and it has child 'John' and the relation is 'nsubj'. So, this node will become the current root, before that 'Mary' with it's event 'said' will be pushed into the stack. Subject 'Mary' will be pushed as 'agent' since the verb 'factive' property of the verb 'said' is true.

Now, if the sentence is: *Mary believes that John is nonsense*. The subject 'Mary' is now an experiencer since the verb 'believes' is not a factive event. Here, the author of this sentence expressing his/her personal opinion by talking about Mary's belief state. A factual information is what 'Mary' said about 'John', not what she believes about him. So, this sentence is considered as an insult.

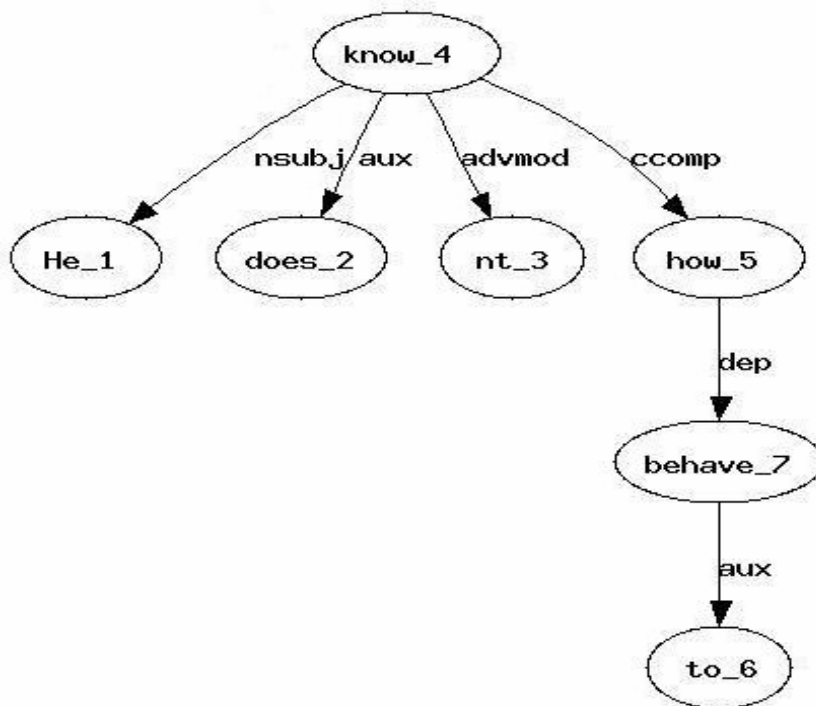


fig -34



This figure shows that the node ‘how’, which has its ‘edge from parent’ is ‘ccomp’ but it has no child as ‘nsubj’. So, the current root will be not be changed.

- 14.If the ‘insulted’ property of the root of a sub tree is set to true, then it’s immediate parent root’s ‘insulted’ property will be set to true if that is not ‘factive’.

The example is **fig-33** at rule no. **13**, where the root verb ‘is’ of a sub tree is insulted, but it will not set the ‘insulted’ property of it’s parent root ‘said’, since ‘said’ has it’s ‘factive’ property as true.

- 15.In case of the property of the current node is ‘ccomp’ or ‘that’ and both of the parent and its child has their ‘evaluative’ property is true, then set ‘modified’ property for of them to true.

*Ex: He must have to know how to behave.*

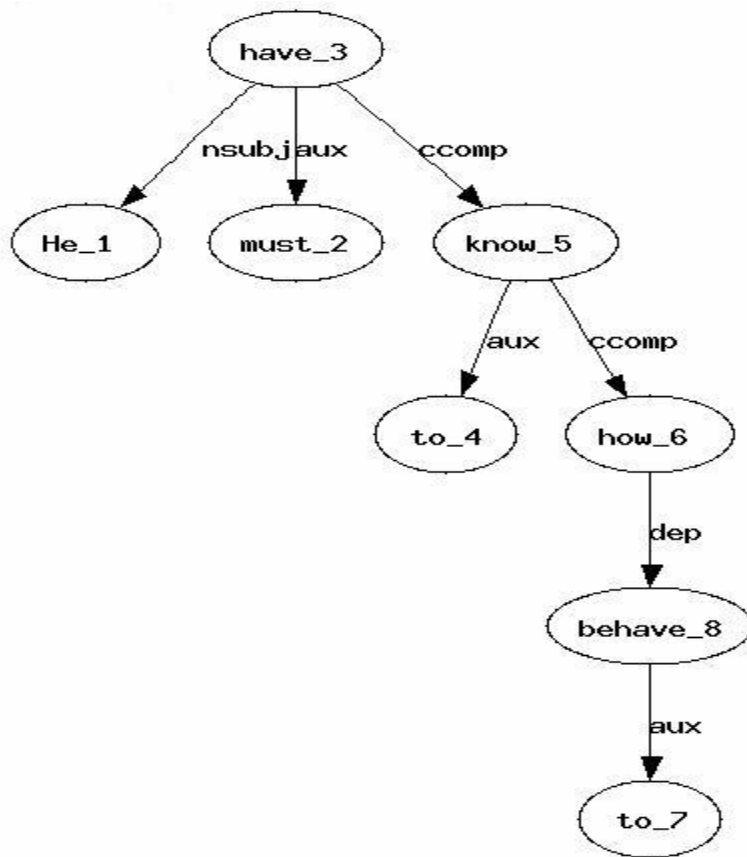
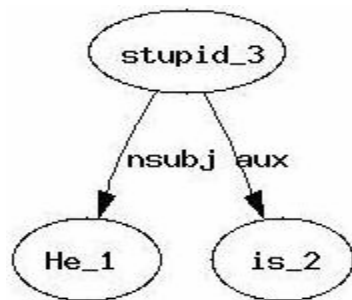


fig -35

Above figure, the node 'know' is the nested root of 'have' and both of their 'evaluative' property is true. So, we can set both node's 'modified' property to true (since both of them is evaluated by each other).

- 16.If current node's tag is verb (VB) and current root's tag is adjective (JJ) then push the subject with it's event which is the current node.

Ex: *He is stupid.*



In this tree when we will visit the node 'is' we can push the subject that is 'he' as an experiencer with its event 'is' into the stack.

**fig -36**

17.It's a final check. When we have finished traversing the whole tree, check if there is any frame yet to be pushed. If there is, then push it into the stack.

Now, tree traversing has been completed. The root node of the tree has its 'insulted' property as a true or as a false, according to the rules described above. A sentence is being considered as an insult if any of its clause bears insulting message. To decide a sentence whether it is an insult or not, following steps have to be executed, once we have got the root of a tree has its 'insulted' property is true:

- First check, currently no scope is open. Then check subject of bottom of the stack is an 'agent' and its immediate top subject is an 'experiencer'. If that is true then see whether those are the same, if same then change that 'agent' to an 'experiencer'.

An example: *Mary said John is an idiot, Mary doesn't know any behavior.*

After separated by <,>

*Mary said John is an idiot*

<,>*Mary doesn't know any behavior.*

agent: Mary
event: said
experiencer: John
event: is



**fig –37:** stack after the first clause

agent: Mary
event: said
experiencer: Mary
event: know



**fig – 38:** stack after the second clause

**fig- 38** shows that agent ‘Mary’ and experiencer ‘Mary’ are the same (they are the same person), the agent ‘Mary’ will become an experiencer.

This step will not be executed if a scope is open. Consider a sentence: *Mary said, “Mary is an idiot.”* The ‘Mary’ inside the scope of punctuation mark could be an another ‘Mary’.

- Now check whether the stack is empty or bottom of the stack contains an ‘experiencer’. If that is true, annotate the sentence as an insult.

The processing part here described is for each clause (or that could be a simple sentence). So, this processing will be repeated for each clause (or for a sentence) until the end of the document.

## Chapter 4: Result

This chapter shows a snapshot of the output of our program. All the sentences are here arbitrarily taken. The input text has three paragraphs. The output shows exactly at which paragraph and at which line (sentence) an insulting message is found.

### Input:

She said, "Lisa doesn't know any behavior." Get a life John! You should be punished for your shameless work.. Mary knows that John is an idiot. "He should know the well behavior. Otherwise I will sue him," she shouted. You sick idiotic liberals, she added there. A donkey is what he is. A donkey does what he does.

According to John, Lisa is stupid. He told that stupid Lisa cannot do this. Actually, John told that because he is nonsense. John told yesterday that shut up you shameless. Shut up you shameless! She didn't show any rude behavior. According to him, Mary didn't learn any courtesy. Actually, we think that Lisa didn't learn any courtesy. John expressed that Mary said that Lisa believes that he is stupid and doesn't know how to behave with a person. John knows that Lisa doesn't know how to behave with a person.

A dog is barking. A dog doesn't know any manner. He played that shot just like a stupid coward. Only coward and foolish could reply that as a great, Mary replied. That idiot said he is a good boy. That good boy said he is an

idiot. "Well, listen to me you bozos, do you know what's are you talking about?" she shouted with that guy.

Output:

[Para: 1 Line: 2] Get a life John!

[Para: 1 Line: 3] You should be punished for your shameless work.

[Para: 1 Line: 4] Mary knows that John is an idiot.

[Para: 1 Line: 8] A donkey is what he is.

[Para: 2 Line: 3] Actually, John told that because he is nonsense.

[Para: 2 Line: 5] Shut up you shameless!

[Para: 2 Line: 8] Actually, we think that Lisa didn't learn any courtesy.

[Para: 2 Line: 9] John expressed that Mary said that Lisa believes that he is stupid and doesn't know how to behave with a person.

[Para: 2 Line: 10] John knows that Lisa doesn't know how to behave with a person.

[Para: 3 Line: 2] A dog doesn't know any manner.

[Para: 3 Line: 3] He played that shot just like a stupid coward.

[Para: 3 Line: 5] That idiot said he is a good boy.

The output shows that the program has annotated exactly the sentences, which have classified as an insulting. Sentences, which bear insulting word or phrases but are speeches of a person, did not come at the output.



# Chapter 5: Conclusion, Limitations and The Future Work

## 5.1 Conclusion

We present a new and an efficient approach for distinguishing flames and information by interpreting the basic meaning that a sentence gives us. However, we are not only annotating flames but also distinguishing. From statistical analysis it has been revealed that, more than 60% of insulting messages are posted as a direct insult and direct insulting messages always contain insulting words or phrases. From psychological point of view, if these messages are categorized and restrict a user to send these kinds of messages, then a human intension to post or exchange of abusive messages can be significantly reduced. Moreover, this automated program can help a user to retrieve only the factual information. This can be a new topic for research since the strategies described here have the similarities of information retrieval mechanism from a syntactical structure along with semantic information.

## 5.2 Limitations

The limitations are:

- This can classify and distinguish insulting messages only bearing insulting words or phrases. The messages, which imply insult as insulting manner, cannot be categorized.

- Our preprocessing part is not yet been full proved. We still have some errors. For some complicated sentences it can be failed to process.
- We didn't yet handle any wrong input such misplacing of comma, unmatched punctuation marks etc.
- Since Stanford parser is a probabilistic parser, it is not guaranteed that all of its output is right. For those cases the program also gives the wrong output.

### **5.3 Future Work**

- Involving world knowledge to make it more efficient.
- Pragmatic analysis.
- Adding learning features, such as 'supervised learning'.
- Can be extended for detecting personal opinions or emotions.
- Make it for other languages such as 'bangla'. In that case we need 'bangla' dependency parser.

## References

[1] wikipedia

[www.en.wikipedia.org](http://www.en.wikipedia.org)

[2] Ellen Spertus. *Smokey: Automatic recognition of hostile messages (1997)*.

[people.mills.edu/spertus/Smokey/smokey.pdf](http://people.mills.edu/spertus/Smokey/smokey.pdf)

[3] J. Wiebe, T. Wilson, R. Bruce, M. Bell, M. Martin. *Learning Subjective Language (2002)*

<http://citeseer.ist.psu.edu/context/2488514/0>

[4] Melanie Martin. *Annotating Flames in Usenet Newsgroups (2002)*.

[www.cs.csustan.edu/~mmartin/pubs/martin\\_poster.pdf](http://www.cs.csustan.edu/~mmartin/pubs/martin_poster.pdf)

[5] Stanford Parser

<http://nlp.stanford.edu/software/lex-parser.shtml>

[6] OpenNLP 1.3.0

[http://aye.comp.nus.edu.sg/portal/RPNLPIR/opennlp\\_tools\\_1.3.0\\_1.2.0.html](http://aye.comp.nus.edu.sg/portal/RPNLPIR/opennlp_tools_1.3.0_1.2.0.html)

[7] T. Wilson, J. Wiebe. *Annotating opinions in the world press (2003)*

<https://rrc.mitre.org/pubs/sigdial03final.pdf>

[8] J. Wiebe, T. Wilson, C. Cardie. *Annotating expressions of opinions and emotions in language (June, 2005)*.

[www.cs.pitt.edu/~wiebe/pubs/papers/lre05withappendix.pdf](http://www.cs.pitt.edu/~wiebe/pubs/papers/lre05withappendix.pdf)

[9] Theresa Wilson, Janyce Wiebe. *Annotating attributions and private states (2005)*.

[www.cs.pitt.edu/~twilson/pubs/acl05wkshop.pdf](http://www.cs.pitt.edu/~twilson/pubs/acl05wkshop.pdf)

[10] M. Arthur Munson. *Automatic annotation of speech events and explicit private state in newswire (17, May, 2004)*.

[www.cs.cornell.edu/~mmunson/publications/docs/sw\\_ons.pdf](http://www.cs.cornell.edu/~mmunson/publications/docs/sw_ons.pdf)

- [11] Janyce Wiebe. *Instructions for annotating opinions in newspaper articles (2002)*  
[nrrc.mitre.org/NRRC/publications.htm](http://nrrc.mitre.org/NRRC/publications.htm)
- [12] T. Wilson, P. Hauffmann, S. Somasundaran, J. Kessler, J. Wiebe, Y. Choi, C. Cardie, E. Riloff, S. Patwardhan *Opinion finder: A system for subjectivity analysis (2005)*  
[www.cs.cornell.edu/home/cardie/papers/hlt-emnlp05-demo.pdf](http://www.cs.cornell.edu/home/cardie/papers/hlt-emnlp05-demo.pdf)
- [13] Soo-Min Kim, E. Hovy. *Automatic detection of opinion bearing words and sentences (2005)*.  
[www.isi.edu/~skim/Download/Papers/2005/ijcnlp\\_cameraready\\_letter.pdf](http://www.isi.edu/~skim/Download/Papers/2005/ijcnlp_cameraready_letter.pdf)
- [14] T. Wilson, J. Wiebe and R. Hwa. *Just How Mad Are You? Finding strong and weak opinion clauses (2004)*.  
[www.cs.pitt.edu/~twilson/pubs/aaai04.pdf](http://www.cs.pitt.edu/~twilson/pubs/aaai04.pdf)
- [15] J. Wiebe, R. Bruce, M. Bell, M. Martin, T. Wilson. *A corpus study of evaluative and speculative language (2001)*.  
<http://citeseer.ist.psu.edu/wiebe01corpus.html>
- [16] J. Wiebe, E. Breck, C. Buckley, C. Cardie, P. Davis, B. Fraser, D. Litman, D. Pierce, E. Riloff, T. Wilson, D. Day, M. Maybury. *Recognizing and organizing opinions expressed in the world press (2003)*.  
[www.cs.cornell.edu/home/cardie/papers/aaai-ss-overall-03.pdf](http://www.cs.cornell.edu/home/cardie/papers/aaai-ss-overall-03.pdf)
- [17] C. Ovesdotter Alm, D. Roth, R. Sproat. *Emotios from text: machine learning for text-based emotion prediction (2005)*.  
[l2r.cs.uiuc.edu/~danr/Papers/AlmRoSp05.pdf](http://l2r.cs.uiuc.edu/~danr/Papers/AlmRoSp05.pdf)
- [18] Sid. *Identifying Subjective Agents in Text (2004)*.  
[www.cs.utah.edu/~sidd/documents/nlp100404.pdf](http://www.cs.utah.edu/~sidd/documents/nlp100404.pdf)
- [19] Jonathan Ian Ellis. *An exploration of human emotion perception from short texts (April, 25, 2005)*.