Digital Communications and Networks 2 (2016) 108–121

HOSTED BY

Contents lists available at ScienceDirect

Digital Communications and Networks

journal homepage: www.elsevier.com/locate/dcan

Sarcastic sentiment detection in tweets streamed in real time: a big data approach

S.K. Bharti*, B. Vachha, R.K. Pradhan, K.S. Babu, S.K. Jena

Department of Computer Science & Engineering, National Institute of Technology, Rourkela 769008, India

A R T I C L E I N F O

Article history:
Received 20 February 2016
Received in revised form
16 May 2016
Accepted 15 June 2016
Available online 12 July 2016

Keywords:
Big data
Flume
Hadoop
Hive
MapReduce
Sarcasm
Sentiment
Tweets

A B S T R A C T

Sarcasm is a type of sentiment where people express their negative feelings using positive or intensified positive words in the text. While speaking, people often use heavy tonal stress and certain gestural clues like rolling of the eyes, hand movement, etc. to reveal sarcastic. In the textual data, these tonal and gestural clues are missing, making sarcasm detection very difficult for an average human. Due to these challenges, researchers show interest in sarcasm detection of social media text, especially in tweets. Rapid growth of tweets in volume and its analysis pose major challenges. In this paper, we proposed a Hadoop based framework that captures real time tweets and processes it with a set of algorithms which identifies sarcastic sentiment effectively. We observe that the elapse time for analyzing and processing under Hadoop based framework significantly outperforms the conventional methods and is more suited for real time streaming tweets.

1. Introduction

With the advent of smart mobile devices and the high-speed Internet, users are able to engage with social media services like Facebook, Twitter, Instagram, etc. The volume of social data being generated is growing rapidly. Statistics from Global WebIndex shows a 17% yearly increase in mobile users with the total number of unique mobile users reaching 3.7 billion people [1]. Social networking websites have become a well-established platform for users to express their feelings and opinions on various topics, such as events, individuals or products. Social media channels have become a popular platform to discuss ideas and to interact with people worldwide. For instance, Facebook claims to have 1.59 billion monthly active users, each one being a friend with 130 people on average [2]. Similarly, Twitter claims to have more than 500 million users, out of which more than 332 million are active [1]. Users post more than 340 million tweets and 1.6 billion search queries every day [1].

With such large volumes of data being generated, a number of challenges are posed. Some of them are accessing, storing, processing, verification of data sources, dealing with misinformation and fusing various types of data [3]. However, almost 80% of generated data is unstructured [4]. As the technology developed, people were given more and more ways to interact, from simple text messaging and message boards to other more engaging and engrossing channels such as images and videos. These days, social media channels are usually the first to get the feedback about current event and trends from their user base, allowing them to provide companies with invaluable data that can be used to position their products in the market as well as gather rapid feedback from customers.

When an event commences or a product is launched, people start tweeting, writing reviews, posting comments, etc. on social media. People turn to social media platforms to read reviews from other users about a product before they decide whether to purchase it or not. Organizations also depend on these sites to know the response of users for their products and subsequently use the feedback to improve their products. However, finding and verifying the legitimacy of opinions or reviews is a formidable task. It is difficult to manually read through all the reviews and determine which of the opinions expressed are sarcastic. In addition, the common reader will have difficulty in recognizing sarcasm in tweets or product reviews, which may end up misleading them.

A tweet or a review may not state the exact orientation of the user directly, i.e., it may be sarcastically expressed. Sarcasm is a kind of sentiment which acts as an interfering factor in any text

* Corresponding author.
E-mail addresses: sbharti1984@gmail.com (S.K. Bharti), bakhtyarvachha@gmail.com (B. Vachha), ramkrushna768@gmail.com (R.K. Pradhan), ksathyababu@nitrkl.ac.in (K.S. Babu), skjena@nitrkl.ac.in (S.K. Jena).

Peer review under responsibility of Chongqing University of Posts and Telecommunications.

http://dx.doi.org/10.1016/j.dcan.2016.06.002
2352-8648/

that can flip the polarity [5]. For example, 'I love being ignored #sarcasm'. Here, "love" expresses a positive sentiment in a negative context. Therefore, the tweet is classified as sarcastic. Unlike a simple negation, sarcastic tweets contain positive words or even intensified positive words to convey a negative opinion or vice versa. This creates a need for the large volumes of reviews, tweets or feedback messages to be analyzed rapidly to predict their exact orientation. Moreover, each tweet may have to pass through a set of algorithms to be accurately classified.

In this paper, we propose a Hadoop-based framework [6] that allows the user to acquire and store tweets in a distributed environment [7] and process them for detecting sarcastic content in real time using the MapReduce [8] programming model. The mapper class works as a partitioner and divides large volume of tweets into small chunks and distributes them among the nodes in the Hadoop cluster. The reducer class works as a combiner and is responsible for collecting processed tweets from each node in the cluster and assembles them to produce the final output. Apache Flume [9,10] is used for capturing tweets in real time as it is highly reliable, distributed and configurable. Flume uses an elegant design to make data loading easy and efficient from several sources into the Hadoop Distributed File System (HDFS) [11]. For processing these tweets stored in the HDFS, we use Apache Hive [12]. It provides us with an SQL-like language called HiveQL to convert queries into mapper and reducer classes [12]. Further, we use natural language processing (NLP) techniques like POS tagging [13], parsing [14], text mining [15,16] and sentiment analysis [17] to identify sarcasm in these processed tweets.

My paper compares and contrasts the time requirements for our approach when run on a standard non-Hadoop implementation as well as on a Hadoop deployment to find the improvement in performance when we use Hadoop. For real time applications where millions of tweets need to be processed as fast as possible, we observe that the time taken by the single node approach increases much higher than the Hadoop implementation. This suggests that for higher volumes of data it is more advantageous to use the proposed deployment for sarcasm analysis.

The contributions of this paper are as follows:

1. Capturing and processing real time tweets using Apache Flume and Hive under the Hadoop framework.
2. We propose a set of algorithms to detect sarcasm in tweets under the Hadoop framework.
3. We propose another set of algorithms to detect sarcasm in tweets.

The rest of this paper is organized as follows. Section 2 presents related work for capturing and processing data acquired through the Twitter streaming API followed by sarcasm analysis of the captured data. Section 3 explains preliminaries of this research paper. The proposed scheme is described in Section 4. Section 5 presents the performance analysis of the proposed schemes. Finally, the conclusion and recommendations for future work are drawn in Section 6.

2. Related work

In this section the literature survey is done on two folds. At first, capturing and preprocessing of the real time tweets are surveyed and then literature on sarcasm detection follows.

2.1. Capturing and preprocessing of tweets in large volume

Rapid adaption and growth of social networking platforms enable users to generate data at an alarming rate. Storing and processing of such large data sets become a complex problem. Twitter is one such social networking platform that generates data continuously. In the existing literature, most of the researchers used Tweepy (An easy-to-use Python library for accessing the Twitter API) and Twitter4J (a java library for accessing the Twitter API) for aggregation of tweets from Twitter [5,18–22]. The Twitter Application Programming Interface (API) [23] provides a streaming API [24] to allow developers to obtain real time access to tweets. Befit and Frank [25] discuss the challenges of capturing Twitter data streams. Tufekci and Zeynep [26] examined the methodological and conceptual challenges for social media based big data operations with special attention to the validity and representativeness of big data analysis of social media. Due to some restrictions placed by Twitter on the use of their retrieval APIs, one can only download a limited amount of tweets in a specified time frame using these APIs and libraries. Getting a larger amount of tweets in real time is a challenging task. There is a need for efficient techniques to acquire a large amount of tweets from Twitter. Researchers are evaluating the feasibility of using the Hadoop ecosystem [6] for the storage and processing [22,27–29] of large amounts of tweets from Twitter. Shirahatti et al. [27] used Apache Flume [10] with the Hadoop ecosystem to collect tweets from Twitter. Ha et al. [22] used Topsy with the Hadoop ecosystem for gathering tweets from Twitter. Furthermore, they analyzed the sentiment and emotion information for the collected tweets in their research. Taylor et al. [28] used the Hadoop framework in applications in the bioinformatics domain.

2.2. Sarcasm sentiment analysis

Sarcasm sentiment analysis is a rapidly growing area of NLP with research ranging from word, phrase and sentence level classification [5,18,19,30] to document [31] and concept level classification [21]. Research is progressing in finding ways for efficient analysis of sentiments with better accuracy in written text as well as analyzing irony, humor and sarcasm within social media data. Sarcastic sentiment detection is classified into three categories based on text features used for classification, which are lexical, pragmatic and hyperbolic as shown in Fig. 1.

2.2.1. Lexical feature based classification

Text properties such as unigram, bigram, n-grams, etc. are classified as lexical features of a text. Authors used these features to identify sarcasm, Kreuz et al. [32] introduced this concept for the first time and they observed that lexical features play a vital role in detecting irony and sarcasm in text. Kreuz et al. [33], in their subsequent work, used these lexical features along with syntactic features to detect sarcastic tweets. Davidov et al. [30] used pattern-based (high-frequency words and content words) and punctuation-based methods to build a weighted k-nearest

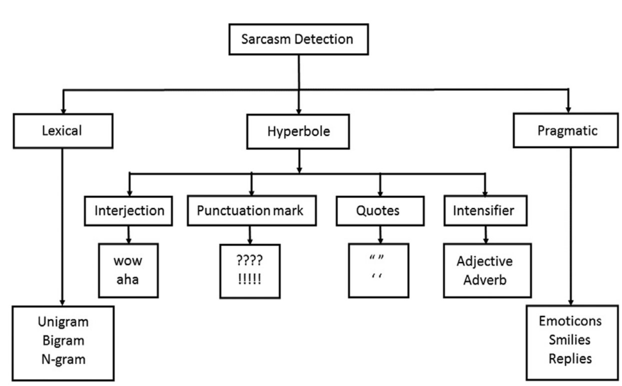

Fig. 1. Classification of sarcasm detection based on text features used.

neighbor (kNN) classification model to perform sarcasm detection. Tsur et al. [34] observed that bigram based features produce better results in detecting sarcasm in tweets and Amazon product reviews. González-Ibánez et al. [18] explored numerous lexical features (derived from LWIC [35] and WordNet affect [36]) to identify sarcasm. Riloff et al. [5] used a well-constructed lexicon based approach to detect sarcasm and for lexicon generation they used unigram, bigram and trigram features. Bharti et al. [19] considered bigram and trigram to generate bags of lexicons for sentiment and situation in tweets. Barbieri et al. [37] considered seven lexical features to detect sarcasm through its inner structure such as unexpectedness, the intensity of the terms or imbalance between registers.

2.2.2. Pragmatic feature based classification

The use of symbolic and figurative text in tweets is frequent due to the limitations in message length of a tweet. These symbolic and figurative texts are called pragmatic features (such as smilies, emoticons, replies, @user, etc.). It is one of the powerful features to identify sarcasm in tweets as several authors have used this feature in their work to detect sarcasm. Pragmatic features are one of the key features used by Kreuz et al. [33] to detect sarcasm in text. Carvalho et al. [38] used pragmatic features like emoticons and special punctuations to detect irony from newspaper text data. González-Ibánez et al. [18] further explored this feature with some more parameters like smilies and replies and developed a sarcasm detection system using the pragmatic features of Twitter data. Tayal et al. [39] also used the pragmatic feature in political tweets to predict which party will win in the election. Similarly, Rajadesingan et al. [40] used psychological and behavioral features on users' present and past tweets to detect sarcasm.

2.2.3. Hyperbole feature based classification

Hyperbole is another key feature often used in sarcasm detection from textual data. A hyperbolic text contains one of the text properties, such as intensifier, interjection, quotes, punctuation, etc. Previous authors used these hyperbole features and achieved good accuracy in their research to detect sarcasm in tweets. Utsumi and Akira [41] discussed extreme adjectives and adverbs and how the presence of these two intensifies the text. Most often, it provides an implicit way to display negative attitudes, i.e., sarcasm. Kreuz et al. [33] discussed the other hyperbolic terms such as interjection and punctuation. They have shown how hyperbole is useful in sarcasm detection. Filatova and Elena [31] used the hyperbole features in document level text. According to them, phrase or sentence level is not sufficient for good accuracy and considered the text context in that document to improve the accuracy. Liebrecht et al. [42] explained hyperbole features with examples of utterances: 'Fantastic weather' when it rains is identified as sarcastic with more ease than the utterance without a hyperbole ('the weather is good' when it rains). Lunando et al. [20] declared that the tweet containing interjection words such as wow, aha, yay, etc. has a higher chance of being sarcastic. They developed a system for sarcasm detection for Indonesian social media. Tungthamthiti et al. [21] explored concept level knowledge using the hyperbolic words in sentences and gave an indirect contradiction between sentiment and situation, such as raining, bad weather, which are conceptually the same. Therefore, if 'raining' is present in any sentence, then one can assume 'bad weather'. Bharti et al. [19] considered interjection as a hyperbole feature to detect sarcasm in tweets that starts with an interjection.

Based on the classification, a consolidated summary of previous studies related to sarcasm identification is shown in Table 1. It provides types of approaches used by previous authors (denoted as A1 and A2), various types of sarcasm occurring in tweets (denoted as T1, T2, T3, T4, T5, T6, and T7), text features (denoted as F1, F2, and F3) and datasets from different domains (denoted as D1, D2, D3, D4, and D5), mostly from Twitter data. The details are shown in Table 2.

From Table 1, it is observed that only Bharti et al. [19] have worked for sarcasm type T2 and T3. Lunando et al. [20] discussed that tweets with interjections are classified as sarcastic. Further, Rajadesingan et al. [40] are the only authors who worked for sarcasm type T4. Most of the researchers identified sarcasm in

Table 1
Previous studies in sarcasm detection in text.

Study	Approaches			Types of sarcasm							Type of feature						Domains				
	A1		A2	T1	T2	T3	T4	T5	T6	T7	F1	F2	F3				D1	D2	D3	D4	D5
	A11	A12											F31	F32	F33	F34					
Kreuz et al.(1995)	✓			✓							✓	✓		✓			✓				
Utsumi et al. (2000)	✓			✓							✓			✓			✓				
Verma et al. (2004)	✓		✓	✓							✓						✓				
Bhattacharyya et al. (2004)	✓		✓	✓							✓						✓				
Kreuz et al. (2007)	✓			✓							✓	✓		✓			✓				
Chaumartin et al. (2007)			✓	✓							✓						✓				
Carvalho et al. (2009)	✓			✓							✓						✓				
Tsur et al. (2010)		✓		✓							✓							✓			
Davidov et al. (2010)		✓		✓							✓				✓		✓	✓			
González-Ibánez (2011)	✓			✓							✓	✓					✓				
Filatova et al. (2012)	✓		✓	✓							✓			✓				✓			
Riloff et al. (2013)	✓		✓	✓							✓						✓				
Lunando et al. (2013)	✓					✓					✓		✓				✓				
Liebrecht et al. (2013)	✓			✓							✓			✓			✓				
Lukin et al. (2013)	✓		✓	✓							✓						✓				
Tungthamthiti et al. (2014)	✓			✓							✓			✓			✓				
Peng et al. (2014)	✓										✓	✓				✓	✓				
Raquel et al. (2014)	✓			✓							✓					✓	✓				
Kunneman et al. (2014)	✓			✓							✓	✓	✓	✓			✓				
Barbieri et al. (2014)	✓			✓							✓						✓				
Tayal et al. (2014)	✓			✓							✓	✓					✓				
Pielage et al. (2014)	✓			✓							✓						✓	✓	✓	✓	✓
Rajadesingan et al. (2015)	✓			✓			✓				✓	✓					✓				
Bharti et al. (2015)			✓	✓	✓	✓					✓		✓				✓				

Table 2
Types, features and domains of sarcasm detection.

Types of Approaches used in sarcasm detection	
A1	Machine learning based
A11	Supervised
A12	Semi-supervised
A2	Corpus based
Types of sarcasm occur in text	
T1	Contrast between positive sentiment and negative situation
T2	Contrast between negative sentiment and positive situation
T3	Tweet starts with an interjection word
T4	Likes and Dislikes contradiction – behavior based
T5	Tweet contradicting universal facts
T6	Tweet carries positive sentiment with antonym pair
T7	Tweet contradicting time dependent facts
Types of features	
F1	Lexical – unigram, bigram, trigram, n-gram, #hashtag
F2	Pragmatic – smilies, emoticons, replies
F3	Hyperbole – Interjection, Intensifier, Punctuation Mark, Quotes
F31	Interjection – yay, oh, wow, yeah, nah, aha, etc.
F32	Intensifier – adverb, adjectives
F33	Punctuation Mark – !!!!!, ????
F34	Quotes – " " , ' '
Types of domains	
D1	Tweets of Twitter
D2	Online product reviews
D3	Website comments
D4	Google Books
D5	Online discussion forums

tweets in type T1. None of the authors worked on sarcasm types T5, T6 and T7 until now. In this work, we consider these research gaps as challenges and propose a set of algorithms to tackle them.

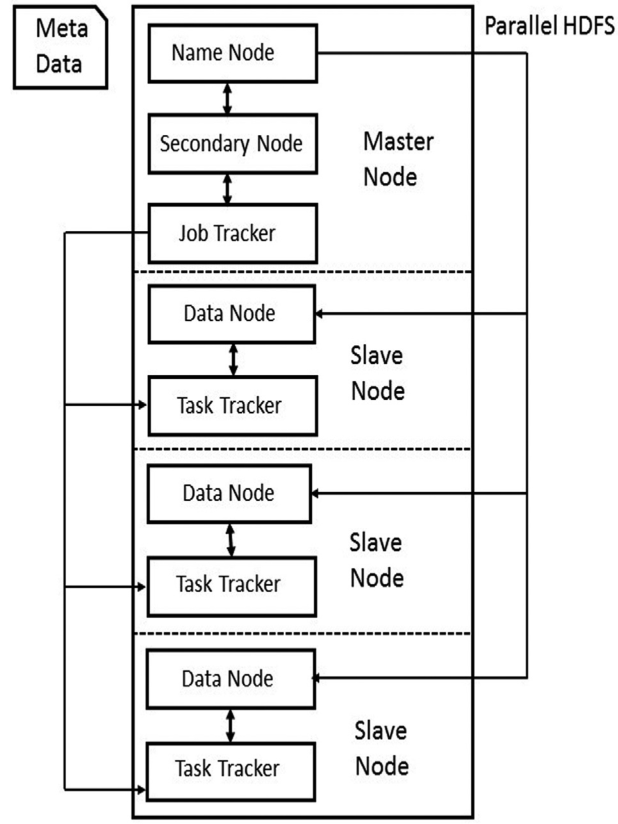

Fig. 3. Parallel HDFS architecture.

3. Preliminaries

This section describes the overall framework for capturing and analyzing tweets streamed in real time. In addition, the architecture of Hadoop HDFS followed by POS tagging, parsing and sentiment analysis of the given phrase or sentence are elaborated.

3.1. Framework for sarcasm analysis in real time tweets

The proposed system uses the Hadoop framework to process and store the tweets streamed in real time. These tweets are retrieved from Twitter using the Twitter streaming API (Twitter4j) as shown in Fig. 2. The Flume module is responsible for communicating with the Twitter streaming API and retrieving tweets matching certain criteria, trends or keywords. The tweets retrieved from Flume are in JavaScript Object Notation (JSON) format which is passed on to the HDFS. Oozie is a module in Hadoop that provides the output from one stage as the input to the next. Oozie is used to partition the incoming tweets into blocks of tweets, partitioned on an hourly basis. These partitions are passed onto the Hive module, which then parses the incoming JSON tweets into a format suitable for consumption by the sarcasm detection engine (SDE). These parsed tweets are stored again in the HDFS and later retrieved by SDE for further processing and attainment of final sentiment summarization.

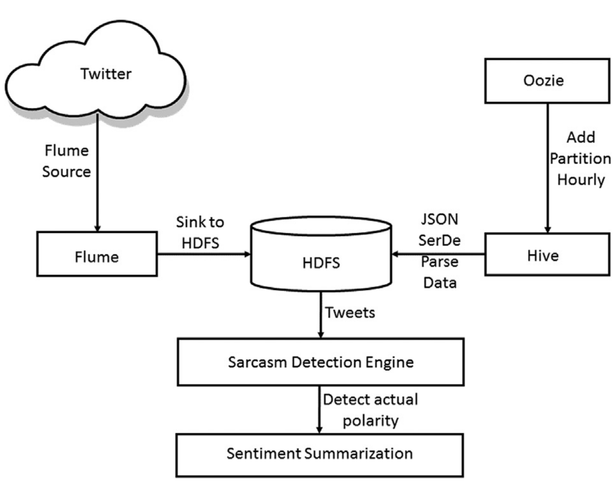

Fig. 2. System model for capturing and analyzing sarcasm sentiment in tweets.

3.2. Parallel HDFS

To increase the throughput of a system and handle the massive volume of tweets, the parallel architecture of HDFS that is used is shown in Fig. 3. The overall file system consists of a metadata file, master node and multiple slave nodes that are managed by the master node.

A metadata file contains two subfiles, namely, fsimage and edits file. The fsimage contains the complete state of the file system at a given instance of time and the edits file contains the log of changes to the file system after the most recent fsimage was made. The master node contains three entities, namely, name node, secondary name node and data node. All three entities in the name node can communicate with each other. The name node is responsible for the overall functioning of the file system. A secondary name node is responsible for updating and maintaining of the name node as well as managing the updates to the metadata. The Job tracker is a service in Hadoop that interfaces between the name node and the task trackers and matches the jobs with the closest available task tracker.

The Slave node contains two entities, namely data node and

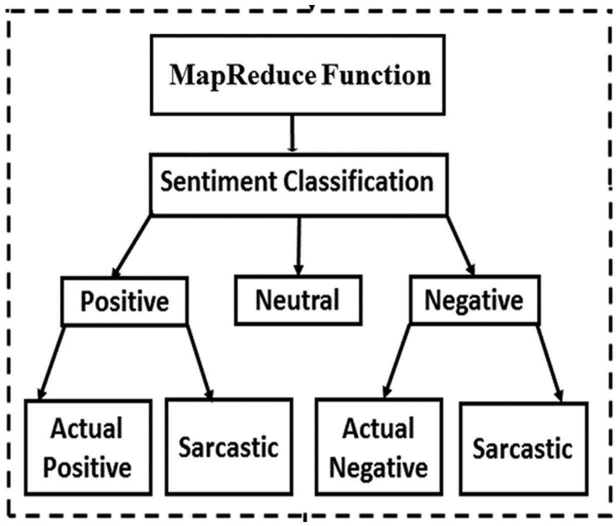

Fig. 4. Sarcasm detection engine.

task tracker. Both entities can communicate with each other within the slave node. The data node is responsible for handling the data blocks and providing the services for storage, and retrieval of the data as requested by the name node. The task tracker is responsible for processing the input according to user requirements and returning the output.

In the parallel HDFS architecture, the name node communicates with the various data nodes in the slave nodes while simultaneously the job tracker in the name node coordinates with the task trackers on the slaves in parallel, resulting in a high rate of output which is fed into the SDE.

3.3. Sarcasm detection engine

To identify the sentiment of a given tweet, it passes through the MapReduce functions for sentiment classification. The tweet is classified into either a negative, positive or neutral, based on the detection engine. Fig. 4 depicts an automated SDE which takes tweets as an input and produces the actual sentiment of the tweet as an output. Once the tweet is classified as either positive or negative, further checks are required to confirm if it has an actual positive/negative sentiment or a sarcastic sentiment.

3.4. Parts-of-speech tagging

Parts-of-speech (POS) tagging divides sentences or paragraphs into words and assigning corresponding parts-of-speech information to each word based on their relationship with adjacent and related words in a phrase, sentence, or paragraph. In this paper, a Hidden Markov Model (HMM) based POS tagger [13] is used to identify the correct POS tag information of given words. For example: POS tag information for the sentence "Love has no finite coverage" is love-NN, has-VBZ, no-DT, finite-JJ, and coverage-NN. Where NN, JJ, VBZ and DT denote the notations for noun, adjective, verb and determiner, respectively. The Penn Treebank tag [43] set notations are used to assign a tag to the particular word. It is a brown corpus style of tagging having 44 tags.

3.5. Parsing

Parsing is a process of analyzing grammatical structure, identifying its parts of speech and syntactic relations of words in sentences. When a sentence is passed through a parser, the parser divides the sentence into words and identifies the POS tag

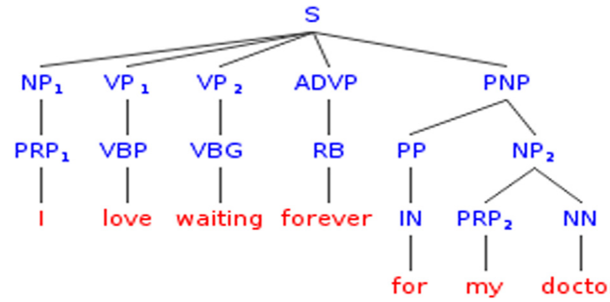

Fig. 5. Parse tree for a tweet: I love waiting forever for my doctor.

information. With the help of the POS information and syntactic relation, it forms units like subject, verb, and object, then determines the relations between these units and generates a parse tree. In this paper, a python based package called TEXTBLOB has been used for parsing. An example of parsing for text "I love waiting forever for my doctor" is I/PRP/B-NP/O, love/NN/I-NP/O, waiting/VBG/B-VP/O, forever/RB/B-ADVP/O, for/IN/B-PP/B-PNP, my/PRP$/BNP/ I-PNP, doctor/NN/I-NP/I-PNP. With the help of the parse data, two examples of parse trees are shown in Figs. 5 and 6.

3.6. Sentiment analysis

Sentiment analysis is a mechanism to recognize one's opinion, polarity, attitude and orientation of any target like movies, individuals, events, sports, products, organizations, locations, services, etc. To identify sentiment in given phrase, we use pre-defined lists of positive and negative words such as Sentiwordnet [44]. It is a standard list for positive and negative English words. Using the Sentiwordnet lists along with Eqs. (1)–(3), we find the sentiment score for a given phrase or sentence:

$$PR = \frac{PWP}{TWP} \tag{1}$$

$$NR = \frac{NWP}{TWP} \tag{2}$$

$$Sentiment\ Score = PR - NR \tag{3}$$

where PR is the positive ratio, NR the negative ratio, PWP the number of positive words in a given phrase, NWP the number of negative words in a given phrase, and TWP the total words in given phrase.

4. Proposed scheme

There is an increasing need for automatic techniques to capture and process real time tweets and analyze their sarcastic sentiment. It provides useful information for market analysis and risk management applications. Therefore, we propose the following

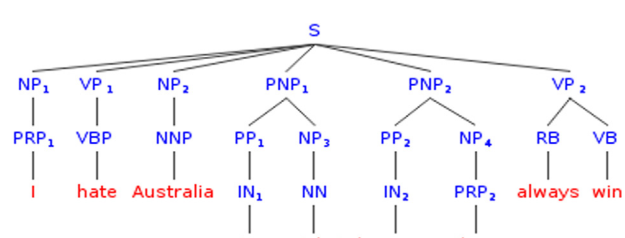

Fig. 6. Parse tree for a tweet: I hate Australia in cricket because they always win.

approaches to sarcasm detection in tweets:

- Capturing and processing real time tweets using Flume and Hive.
- An HMM-based algorithm for POS tagging.
- MapReduce functions for three approaches to detect sarcasm in tweets:
  1. *Parsing_based_lexicon_generation_algorithm.*
  2. *Interjection_word_start.*
  3. *Positive_sentiment_with_antonym_pair.*
- Other approaches to detect sarcasm in tweets:
  1. *Tweet_contradicting_universal_facts.*
  2. *Tweet_contradicting_time_dependent_facts.*
  3. *Likes_dislikes_contradiction.*

4.1. Capturing and processing real time streaming tweets using flume and hive

The Twitter Streaming API returns a constant stream of tweets in JSON format which is then stored in the HDFS as shown in Fig. 2. To avoid issues related to security and writing code that requires complicated integration with secure clusters, we prefer to use the existing components within Cloudera Hadoop [29]. This allows us to directly store the data retrieved by the API into the HDFS. We use Apache Flume to store the data in the HDFS. Flume is a data ingestion system that is defined by setting up channels in which data flows between sources and sinks. Each piece of data is an event and each such event goes through a channel. The Twitter API does the work of the source here and the sink is a system that writes out the data to the HDFS. Along with the data capture, the Flume module allows us to set up custom filters and keyword-based searches that allow us to further narrow down the tweets to just the ones relevant to our requirements.

Once the data from the Twitter API is fed into the HDFS, the data must be pre-processed to convert the tweets stored in JSON format into usable text for the SDE. We make use of the Oozie module for handling the work flow, which is scheduled to run at periodic intervals. We configure Oozie to partition the data in the HDFS on the basis of hourly retrievals and load the last hour's data into the hive as shown in Fig. 2. The hive is another module in Hadoop that allows one to translate and load data with the help of the Serializer–Deserializer. This allows us to convert the JSON tweets into a query-able format an we then add these entries back into the HDFS for processing by the SDE.

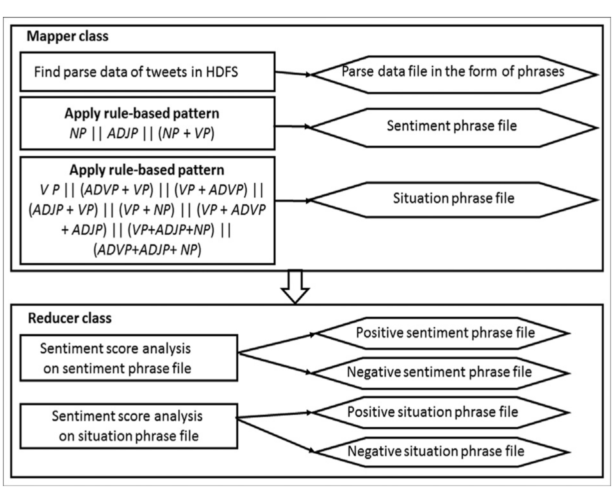

Fig. 7. Procedure to obtain sentiment and situation phrase from tweets

4.2. HMM-based POS tagging

In this paper, an HMM-based POS tagger is deployed to evaluate accurate POS tag information for the Twitter dataset as shown in Algorithms 1 and 2. Algorithm 1 trains the system using 500,000 pre-tagged (according to the Penn Tree Bank style) American English words from the American National Corpus (ANC) [45,46]. Algorithm 2 evaluates the POS tag information of words in the given dataset.

Algorithm 1. POS_training.

```
Data: dataset := Annotated corpus
Result: WT:= dictionary variable with
        ⟨key, value⟩ pair for each word with its
        tag in the corpus
        TT:= dictionary variable with
        ⟨key, value⟩ for bigram tag pair
        T:=dictionary variable with
        ⟨key, value⟩ pair for each tag with
        its occurrences
while sentence in corpus do
    while word in sentence do
        if word==first_word then
            previous_tag =$
            current_tag = POS tag of current word
            TT[previous_tag, current_tag]++
            T[current_tag]++
            WT[word, current_tag]++
        end
        else
            previous_tag =POS tag of previous
            word
            current_tag = POS tag of current word
            TT[previous_tag, current_tag]++
            T[current_tag]++
            WT[word, current_tag]++
        end
    end
end
```

Algorithm 2. POS_testing.

```
Data: dataset := corpus of real time streaming
      tweets
Result: Tags sequences of each tweet in dataset
while tweet in dataset do
    while word in tweet do
        APT = find_all_possible_tag(word)
        if word==first_word then
            argmax_{t ∈ APT}[TT($, t)/T($)] *
            [WT(word,t)/T(t)]
        end
        else
            P = tag of previous word
            argmax_{t ∈ APT}[TT(P, t)/T(P)] *
            [WT(word,t)/T(t)]
        end
    end
end
```

According to Algorithm 1, HMM uses pre-tagged American English words [45,46] as an input and creates three dictionary objects, namely WT, TT and T. WT stores the number of occurrence of each word with its associated tag in the training corpus. Similarly, TT stores the number of occurrence of the bi-gram tags in the corpus and T stores the number of occurrence of uni-gram tag. For each word in the sentence, it checks if the word is the starting word of the sentence or not. If a word is the starting word then it assumes the previous tag to be ‘ $’. Otherwise, the previous tag is the tag of the previous word in the respective sentence. It increases the occurrence of various tags through the dictionary objects WT, TT and T. Finally, it creates a probability table using the dictionary objects WT, TT and T.

Algorithm 2 finds all the possible tags of a given word (for tag evaluation) using the pre-tagged corpus [45,46] and applies Eq. (4) [47], if the word is the starting word of a respective sentence otherwise it applies Eq. (5) [47]. Next, it selects the tag whose probability value is maximum. For example: once you encounter a POS tag determiner (DT), such as ‘the’, maybe the probability that the next word is a noun is 40% and it being a verb is 20%. Once the model finishes its training, it is used to determine whether ‘can’ in ‘the can’ is a noun (as it should be) or a verb:

$$\underset{t \in APT}{\operatorname{argmax}}[TT(\$, t)/T(\$)] * [WT(word, t)/T(t)] \tag{4}$$

$$\underset{t \in APT}{\operatorname{argmax}}[TT(P, t)/T(P)] * [WT(word, t)/T(t)] \tag{5}$$

where APT is all possible tags

4.3. MapReduce functions for sarcasm analysis

Here, the Map function comprises three approaches to detect sarcasm. Each of the approaches is detailed below.

4.3.1. Parsing based lexicon generation algorithm

The MapReduce function, parsing based lexicon generation algorithm (PBLGA), is based on our previous study [19]. It takes tweets as an input from HDFS and parses them into the form of phrases such as noun phrase (NP), verb phrase (VP), adjective phrase (ADJP), etc. These phrases are stored in the phrase file for further processing. The phrase file is then subsequently passed onto the rule-based classifier to classify sentiment phrases and situation phrases as shown in the mapper part of Fig. 7 and stores it in the sentiment phrase file and situation phrase file. Then, the output of the mapper class (sentiment phrase file and situation phrase file) passes to the reducer class as an input. The reducer class calculates the sentiment score (as explained in Section 3.6) of each phrase in both the sentiment and the situation phrase file. Then, it gives output an aggregated positive or negative score for each phrase in terms of the sentiment and situation of the tweet. Based on whether the score is positive or negative, the phrases are stored in the corresponding phrase file as shown in the reducer class of Fig. 7. PBLGA generates four files, namely positive sentiment, negative sentiment, positive situation and negative situation files as an output. Furthermore, we use these four files to detect sarcasm in tweets with tweet structure contradiction between positive sentiment and negative situation and vice versa as shown in Algorithm 3.

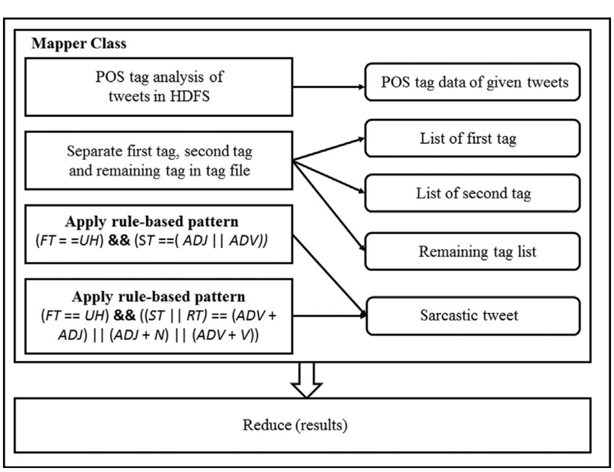

Fig. 8. Procedure to detect sarcasm in tweets that starts with interjection word.

Algorithm 3. PBLGA_testing.

```
Data: dataset := Tweets for testing, bags of
      lexicons
Result: classification:= sarcastic or non
        sarcastic
while tweets in dataset do
    count = 0
    sarcamsFlag = False;
    while words in tweet do
        if word==(any phrase in positive
          sentiment lexicons) && (count == 0)
          then
            count = 1;
            check negative situation lexicons
            continue;
        end
        if word == (any phrase in negative
          situation lexicons) && (count == 1)
          then
            sarcasmFlag = True
            break;
        end
        else
            if word == (any phrase in negative
              sentiment lexicons) && (count == 0)
              then
                count = 1
                check positive situation lexicons
                continue;
            end
            if word == (any phrase in negative
              situation lexicons) && (count == 1)
              then
                sarcasmFlag = True
                break;
            end
        end
        if sarcasmFlag==True then
            Given tweet is sarcastic
        end
        else
            Given tweet is not sarcastic
        end
    end
end
```

According to Algorithm 3, it takes testing tweets and four bags of lexicons generated using PBLGA. If the testing tweet matches with any positive sentiment from the positive sentiment file, it subsequently checks for any matches with negative situation against the negative situation file. If both checks match, the testing tweet is sarcastic and similarly, and it checks for sarcasm with a negative sentiment in a positive situation. Otherwise, the given tweet is not sarcastic. Both the algorithms are executed under the Hadoop framework as well as without the Hadoop framework to compare the running time.

4.3.2. Interjection word start

The MapReduce function for interjection word start (IWS) is also based on [19] as shown in Fig. 8. This approach is applicable for the tweets that start with an interjection word such as aha, wow, nah, uh, etc. In this approach, the tweet that is sent to the mapper is first parsed into its constituent tags using Algorithms 1 and 2. Then, the tags are separated as first tag, second tag and remaining tags of each tweet. The output of this stage gives us three lists: the list of the first tag, which stores the first tag of the tweet, the list of the second tag, which stores the second tag of the tweet and the list of remaining tags, which stores the remaining tags in the tweet. The lists are then passed to a rule based pattern as given in the mapper class of Fig. 8 that checks that if the first tag is an interjection, i.e., UH (interjection tag notation) and second tag is either adjective or adverb, the tweet is classified as sarcastic. Otherwise, it checks that if the first tag is an interjection and the remaining tags are either adverbs followed by adjectives, adjectives followed by nouns, or adverbs followed by verbs, the tweet is sarcastic else it is not sarcastic. If the pattern does not find any match in a given tweet, tweet is not sarcastic. The algorithm IWS also executes under the Hadoop framework as well as without the Hadoop framework to compare the running time.

4.3.3. Positive sentiment with antonym pair

The MapReduce function for positive sentiment with antonym pair (PSWAP) is a novel approach as shown in Fig. 9 to determine if the tweet is sarcastic or not. The tweet that is sent to the mapper is first parsed into its constituent tags using Algorithms 1 and 2. The output of this stage gives us a bag of tags which is then passed to a rule based classifier as given in the mapper class of Fig. 9 which looks for antonym pairs of certain tags such as noun, verb, adjective and adverb. If any antonym pair is found, it stores them in a separate file. The reducer class is responsible for generating a sentiment score using Eqs. (1)–(3) for the tweet contained in the file of antonym tweets and are sorted according to their sentiment score into positive and negative sentiment tweets. It then classifies

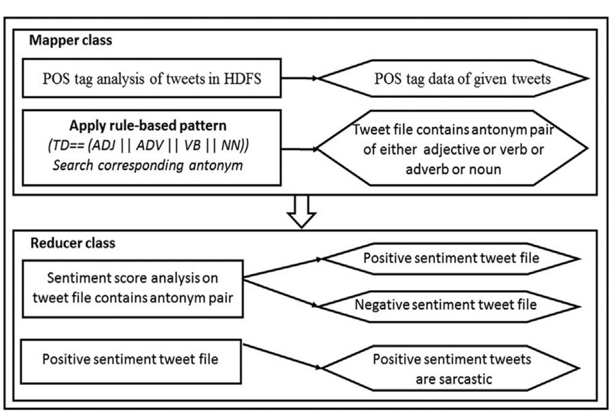

Fig. 9. Procedure to detect sarcasm in positive sentiment tweets with antonym pair.

all the positive sentiment tweets as sarcastic as shown in the reducer class of Fig. 9. In this approach, the antonym pairs of nouns, verbs, adjectives and adverbs are taken from NLTK wordnet [48]. The algorithm PSWAP is executed under the Hadoop framework as well as without Hadoop framework to compare the running time.

4.4. Other approaches for sarcasm detection in tweets

We propose three other novel approaches to identify sarcasm in three different tweet types, i.e., T4, T5 and T7 as shown in Table 2. Due to the unavailability of various aspects modeling these algorithms in the Hadoop framework is undone. However, the methods were implemented without the Hadoop framework. Each of the methods is described below.

4.4.1. Tweets contradicting with universal facts

Tweets contradicting with universal facts (TCUF) is based on universal facts. In this approach, universal facts are used as a feature to identify sarcasm in tweets as shown in Algorithm 4. For an example 'the sun rises in the east' is a universal fact. The corpus of universal fact sentences, Algorithm 4 takes as an input and generates a list of ⟨*key*, *value*⟩ pairs for every sentence in the corpus. To generate ⟨*key*, *value*⟩ pair, it finds triplets of (subject, verb, and object) values according to the Rusu_Triplets [49] method for every sentence. Furthermore, it combines the subject and verb together as key and object as value. The ⟨*key*, *value*⟩ pair for the sentence "the sun rises in the east" is ⟨⟨*sun*, *rises*⟩, *east*⟩.

Algorithm 4. Tweet_contradict_universal_facts.

Data: $dataset$:= Corpus of universal facts.
Result: $Result$:= A $\langle Key, Value \rangle$ pair
Notation: S: Subject, V: verb, O: Object, T: tweets, C: corpus, PF: parse file, TWP: tweet wise_parse_phrase, UFF: Universal_fact_file.
Initialization : $PF = \{ \phi \}$, $UFF = \{ \phi \}$
while T *in* C **do**
  | p = find_parsing (T) $PF \leftarrow PF \cup p$
end
while TWP *in* PF **do**
  | S = find_subject (TWP)
  | V = find_verb (TWP)
  | O = find_object (TWP)
  | Key ← (S + V)
  | Value ← O
  | $UFF \leftarrow \langle Key, Value \rangle$
end

Identifying sarcasm in tweets using universal facts is shown in Algorithm 5. It takes the universal facts ⟨*key*, *value*⟩ pair file and tests the tweets as input and extracts triplet values (subject, object, verb) from the test tweets using the Rusu_Triplets [49] method. Furthermore, we form ⟨*key*, *value*⟩ pairs of the testing tweet using the subject, verb, and object. If the ⟨*key*, *value*⟩ of the testing tweet is matched with any key in universal fact ⟨*key*, *value*⟩ pair file, it checks the value of the testing tweet along with the corresponding value in the universal fact ⟨*key*, *value*⟩ pair file. If both the ⟨*key*, *value*⟩ pairs are matched, the current testing tweet is not sarcastic. Otherwise, the tweet is sarcastic.

Table 3
Experimental environment.

Components	OS	CPU	Memory	HDD
Primary server	Ubuntu_14.04 × 64	Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz)	24 GB	1 TB
Secondary server	Ubuntu_14.04 × 64	Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz)	8 GB	1 TB
Data server 1	Ubuntu_14.04 × 64	Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz)	4 GB	20 GB
Data server 2	Ubuntu_14.04 × 64	Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz)	4 GB	20 GB
Data server 3	Ubuntu_14.04 x64	Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz)	4 GB	20 GB

Table 4
Datasets captured for experiment and analysis.

Datasets	No. of tweets (approx)	Extraction period (h)
Set 1	5,000	1
Set 2	51,000	9
Set 3	100,000	21
Set 4	250,000	50
Set 5	1,050,000	187

Algorithm 5. TCUF_testing_tweets.

Data: *dataset* := Corpus of tweets, UFF.
Result: *classification*:= Sarcastic or not sarcastic
Notation: S: Subject, V: verb, O: Object, T: tweets, C: corpus, PF: parse file, TWP: tweet wise_parse_phrase
Initialization : $PF = \{ \phi \}$
while T *in* C **do**
  p = find_parsing (T) $PF \leftarrow PF \cup p$
end
while TWP *in* PF **do**
  SarcasticFlag=1;
  S = find_subject (TWP)
  V = find_verb (TWP)
  O = find_object (TWP)
  Key ← (S + V)
  Value ← O
  forms $\langle Key, Value \rangle$ pair for all tweets in corpus
  if *(Key in UFF) && (Val==UFF[Key].Value)* **then**
    SarcasticFlag=0;
  end
  return SarcasticFlag;
end

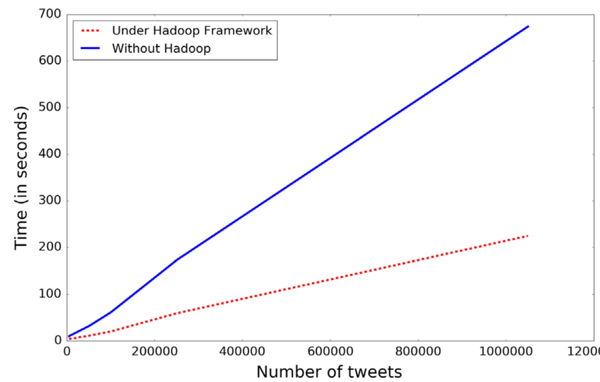

Fig. 10. Elapsed time for POS tagging under the Hadoop framework vs without the Hadoop framework.

4.4.2. Tweets contradicting with time-dependent facts

Tweets contradicting with time-dependent facts (TCTDF) are based on temporal facts. In this approach, time-dependent facts (ones that may change over a certain time period) are used as a feature to identify sarcasm in tweets as shown in Algorithm 6. For instance, '@MirzaSania becomes world number one. Great day for Indian tennis' is a time-dependent fact sentence. After some time, someone else will be the number one tennis player. The newspaper headlines are used as a corpus for time-dependent facts. Algorithm 6 uses newspaper headlines as an input corpus and generates a list of ⟨*key*, *value*⟩ pairs for every headlines in the corpus. To generate a ⟨*key*, *value*⟩ pair, it finds the triplet of (subject, verb, and object) values according to the Rusu_Triplets [49] method for every sentence. Furthermore, it combines the subject and verb together as key and combines the object and time-stamp as value. The time-stamp is the news headline date. The ⟨*key*, *value*⟩ pair for the sentence 'Wow, Australia won the cricket world cup again in 2015' is ⟨(*Australia*, *won*), (*cricketworldcup*, 2015)⟩.

Algorithm 6. Tweet_contradict_time_dependent_facts.

Data: *dataset* := Corpus of universal facts.
Result: *classification*:= A $\langle Key, Value \rangle$ pair
Notation: S: Subject, V: verb, O: Object, T: tweets, C: corpus, PF: parse file, TWP: tweet wise_parse_phrase, UFF: Universal_fact_file.
Initialization : $PF = \{ \phi \}$, $UFF = \{ \phi \}$
while T *in* C **do**
  p = find_parsing (T) $PF \leftarrow PF \cup p$
end
while TWP *in* PF **do**
  S = find_subject (TWP)
  V = find_verb (TWP)
  O = find_object (TWP)
  Key ← (S + V)
  Value ← (O + TS)
  $UFF \leftarrow \langle Key, Value \rangle$
end

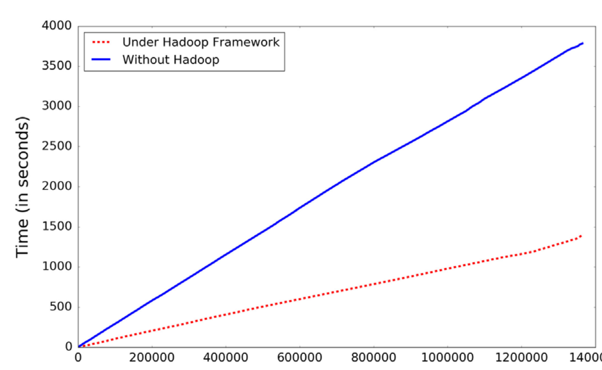

Fig. 11. Processing time to analyze sarcasm in tweets using PBLGA under the Hadoop framework vs without the Hadoop framework.

Identifying sarcasm in tweets using time-dependent facts is similar to TCUF as shown in Algorithm 7. The only difference is in the value of the ⟨*key*, *value*⟩ pair. While matching the ⟨*key*, *value*⟩ pair of the testing tweets with the ⟨*key*, *value*⟩ pair in the file to identify sarcasm using the TCTDF approach, one needs to match the object as well as the time-stamp together as the value. If both match, the current testing tweet is not sarcastic else it is sarcastic.

Algorithm 7. TCTDF_testing_tweets.

```
Data: dataset := Corpus of tweets, UFF.
Result: classification:= Sarcastic or not sarcastic
Notation: S: Subject, V: verb, O: Object, T:
tweets, C: corpus, PF: parse file, TWP: tweet
wise_parse_phrase
Initialization : PF = { ϕ }
while T in C do
  | p = find_parsing (T) PF ← PF ∪ p
end
while TWP in PF do
    SarcasticFlag=1;
    S = find_subject (TWP)
    V = find_verb (TWP)
    O = find_object (TWP)
    Key ← (S + V)
    Value ← (O + T)
    forms ⟨Key, Value⟩ pair for all tweets in
    corpus
    if (Key in UFF) &&
    (Val==UFF[Key].Value) then
      | SarcasticFlag=0;
    end
    return SarcasticFlag;
end
```

4.4.3. Likes – dislikes contradiction

Likes – dislikes contradiction (LDC) is based on the behavioral features of the Twitter user. It is given in Algorithm 8. Here, the algorithm observes a user's behavior using their past tweets. It analyzes the user's tweet history in the profile and generates a list behaviors for his likes and dislikes. To generate the likes and dislikes list of a particular user, one needs to crawl through all the past tweets from the user's Twitter account as an input for Algorithm 8. Next, the algorithm calculates the sentiment score of all the tweets in the corpus using Eqs. (1)–(3). Later it classifies the tweets as positive sentiment or negative sentiment using the sentiment score (if the sentiment score is >0.0, the tweet is positive). Otherwise the tweet is negative. Then both the positive and negative tweets are stored in separate files. From the positive sentiment tweet file, one needs to extract triplet value (subject, object, verb) for every tweet in the file using the Rusu_Triplets [49] method. If the subject value is a pronoun such as 'I' or 'We', 'object' value of that tweet is appended in the likes list. Otherwise, the 'subject' value of that tweet is appended in the likes list. Similarly, in the negative sentiment tweet file, one needs to extract triplet value (subject, object, verb) for every tweet in the file using the Rusu_Triplets [49] method. If the subject value is a pronoun such as 'I' or 'We', the 'object' value of that tweet is appended to the dislikes list. Otherwise, the 'subject' value of that tweet is appended in the dislikes list. For example: '@Modi is doing good job for India'. Given the tweet is positive as the word 'good' is present, the subject of this particular tweet is 'Modi'. Therefore, "Modi" is appended to the likes list of that particular user.

Algorithm 8. Likes_and_Dislikes_Contradiction.

```
Data: dataset := Tweets corpus of particular user.
Result: Result:= Likes and dislikes list of that
        user.
Notation: N: Noun, T: Tweets, C: Corpus,
PSTF: Positive_sentiment_tag_file, NSTF:
Negative_sentiment_tag_file, t: tag, TWT:
Tweet_wise_tag, FT: First_tag, pstf:
Positive_sentiment_tweet_file, nstf:
Negative_sentiment_tweet_file, LL: Likes_list,
DLL: Dislikes_list, PF: Parse file.
Initialization : pstf = { ϕ }, nstf = { ϕ }, NSTF
= { ϕ }, PSTF = { ϕ }, LF = { ϕ }, DLF = { ϕ }
while T in C do
  | SC = find_sentiment_score (T)
  | if SC >0.0 then
  |   | pstf ← pstf ∪ T
  | end
  | if SC <0.0 then
  |   | nstf ← nstf ∪ T
  | else
  |   | Discard the tweet.
  | end
end
while T in pstf do
  | k = find_postag (T)
  | PSTF ← PSTF ∪ k
  | x = find_parsing (T)
  | PF ← PSTF ∪ x
end
while TWT in PSTF do
  | t = find_subset (TWT)
  | FT = find_first_tag (t)
  | if (FT = pronoun) then
  |   | O = find_object (PF)
  |   | LL ← LL ∪ O
  | end
  | else
  |   | S = find_subject (PF)
  |   | LL ← LL ∪ S
  | end
end
while T in nstf do
  | k = find_postag (T)
  | NSTF ← NSTF ∪ k
  | x = find_parsing (T)
  | PF ← PSTF ∪ x
end
while TWT in NSTF do
  | t = find_subset (TWT)
  | FT = find_first_tag (t)
  | if (FT = PRP) then
  |   | O = find_object (PF)
  |   | DLL ← DLL ∪ S
  | end
  | else
  |   | S = find_subject (PF)
  |   | DLL ← DLL ∪ S
  | end
end
```

The method to identify sarcasm in tweets using behavioral features (likes, dislikes) is shown in Algorithm 9. The algorithm considers the testing tweets and the list of likes and dislikes as an input parameter for the particular user. While testing sarcasm in

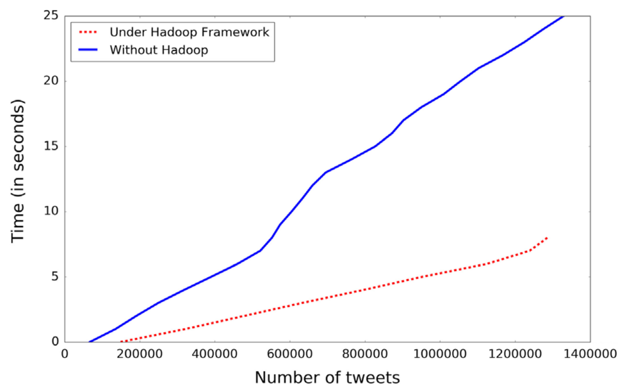

Fig. 12. Processing time to analyze sarcasm in tweets using IWS under the Hadoop framework vs without the Hadoop framework.

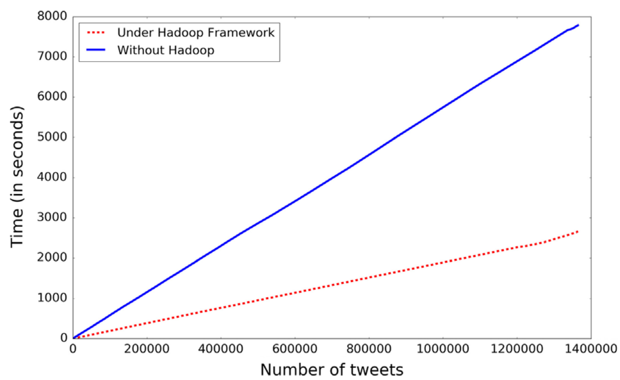

Fig. 13. Processing time to analyze sarcasm in tweets using PBLGA under Hadoop framework vs without Hadoop framework.

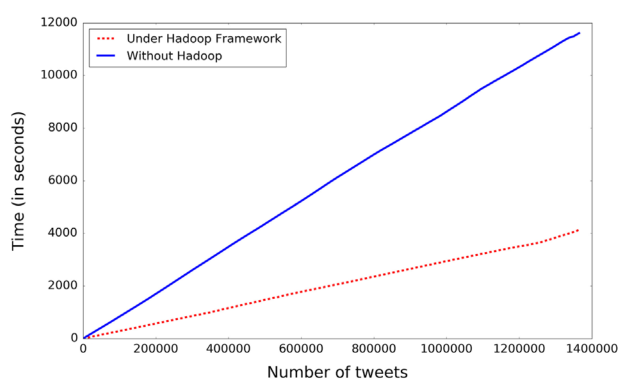

Fig. 14. Processing time to analyze sarcasm in tweets using PBLGA, IWS and PSWAP (combined approach) under the Hadoop framework vs without the Hadoop framework.

tweets, one needs to calculate the sentiment score of the tweet. Then, extract the triplet (subject, verb and object) of that tweet. If the tweet is positive and the subject is not a pronoun check the subject value in the likes list. If the subject value is found in the likes list, the tweet is not sarcastic. If it is found in the dislikes list, the tweet is sarcastic. Similarly, if the subject value is a pronoun and the tweet is positive the object value checks the likes list. If it is found the tweet is not sarcastic. If it is found in the dislikes list the tweet is sarcastic. In a similar fashion, one identifies sarcasm for negative tweets as well.

Algorithm 9. LDC_testing_tweets.

Data: *dataset* := Corpus of tweets, LL, DLL
Result: *Classification*:= sarcastic or not sarcastic.
Notation: N: Noun, T: Tweets, C: Corpus, $PSTF$: Positive_sentiment_tag_file, $NSTF$: Negative_sentiment_tag_file, t: tag, TWT: Tweet_wise_tag, FT: First_tag, $pstf$: Positive_sentiment_tweet_file, $nstf$: Negative_sentiment_tweet_file, LL: Likes_list, DLL: Dislikes_list.
Initialization : $pstf = \{\phi\}$, $nstf = \{\phi\}$, $NSTF = \{\phi\}$, $PSTF = \{\phi\}$, $LF = \{\phi\}$, $DLF = \{\phi\}$

```
while T in C do
    SC = find_sentiment_score (T)
    k = find_postag (T)
    x = find_parse (T)
    t = find_subset (k)
    FT = find_first_tag (t)
    if (FT = pronoun) then
        O = find_object (x)
    end
    else
        S = find_object (x)
    end
    if (SC >0.0) && (O in LL) then
        Tweet is not sarcastic.
    end
    if (SC >0.0) && (O in DLL) then
        Tweet is sarcastic.
    else
        Discard the tweet.
    end
    if (SC <0.0) && (O in LL) then
        Tweet is sarcastic.
    end
    if (SC <0.0) && (O in DLL) then
        Tweet is not sarcastic.
    else
        Discard the tweet.
    end
    if (SC >0.0) && (S in LL) then
        Tweet is not sarcastic.
    end
    if (SC >0.0) && (S in DLL) then
        Tweet is sarcastic.
    else
        Discard the tweet.
    end
    if (SC <0.0) && (S in LL) then
        Tweet is sarcastic.
    end
    if (SC <0.0) && (S in DLL) then
        Tweet is not sarcastic.
    else
        Discard the tweet.
    end
end
```

Table 5
Precision, recall and F-score values for proposed approaches.

Approach	*Precision*	*Recall*	*F – score*
PBLGA approach	0.84	0.81	0.82
IWS approach	0.83	0.91	0.87
PSWAP approach	0.92	0.89	0.90
Combined (PBLGA, IWS, and PSWAP) approach	0.97	0.98	0.97
LDC (first user's account)	0.92	0.72	0.81
LDC (second user's account)	0.91	0.77	0.84
LDC (third user's account)	0.92	0.73	0.82
TCUF approach	0.96	0.57	0.72
TCTDF approach	0.93	0.62	0.74

5. Results and discussion

This section describes the experimental results of the proposed scheme. We started with an experimental setup where a five node cluster is deployed under the Hadoop framework. Five datasets are crawled using Apache Flume and the Twitter streaming API. We also discuss the time consumption of the proposed approach under the Hadoop framework as well as without the Hadoop framework and made a comparison. We also discuss all the approaches with precision, recall and F-score measure.

5.1. Experimental environment

Our experimental setup consists of a five node cluster with the specifications as shown in Table 3. The master node consists of an Intel Xeon E5-2620 (6 core, v3 @ 2.4 GHz) processor with 6 cores running the Ubuntu 14.04 operating system with 24 GB of main memory. The remaining four nodes were virtual machines. All the VMs ran on a single machine. The secondary name node server is another Ubuntu 14.04 machine running on an Intel Xeon E5-2620 with 8 GB of main memory. The remaining three slave nodes responsible for processing the data consist of three Ubuntu 14.04 machines running Intel Xeon E5-2620 with 4 GB of main memory.

5.2. Datasets collection for experiment and analysis

The datasets for the experimental analysis are shown in Table 4. There are five sets of tweets crawled from the Twitter using the Twitter Streaming API and processed through Flume before being stored in the HDFS. In total, 1.45 million tweets were collected using keywords #sarcasm, #sarcastic, sarcasm, sarcastic, happy, enjoy, sad, good, bad, love, joyful, hate, etc. After preprocessing, approximately 156,000 tweets were found as sarcastic (tweets ending with #sarcasm or #sarcastic). The remaining tweets approximately 1.294 million were not sarcastic. Every set contained a different number of tweets. Depending on the number of tweets in each set, the crawling time (in hours) is given in Table 4.

5.3. Execution time for POS tagging

In this paper, POS tagging is an essential phase for all the proposed approaches. Therefore, we used Algorithms 1 and 2 to find POS information for all the datasets (approximately 1.45 million tweets). We deployed algorithms on both Hadoop as well as without the Hadoop framework and estimated the elapsed time as shown in Fig. 10. The solid line shows time taken (approx. 674 s) for POS tagging (approx. 10.5 million tweets) without the Hadoop framework, while the dotted line shows time (approx. 225 s) for POS tagging (approx. 10.5 million tweets) under the Hadoop framework. Tweets were in different sets and we ran the POS tag algorithm separately for each set. Therefore the graph in Fig. 10 shows the maximum time (674 s) for 10.5 million tweets.

5.4. Execution time for sarcasm detection algorithm

There are three proposed approaches, namely PBLGA, IWS and PSWAP, which are deployed under Hadoop framework to analyze the estimated time for sarcasm detection in tweets. We pass tagged tweets as an input to all three approaches. Therefore, the tagging time is not considered in the proposed approaches for sarcasm analysis. Then, we compared the elapsed time under the Hadoop framework vs without the Hadoop framework for all three approaches as shown in Figs. 11–13. PBLGA approach takes approx. 3386 s to analyze sarcasm in 1.4 million tweets without the Hadoop framework and takes approx. 1,400 s to analyze sarcasm in 1.4 million tweets under the Hadoop framework. The IWS approach takes approx. 25 s to analyze sarcasm in 1.4 million tweets without the Hadoop framework and takes approx. 9 s to analyze sarcasm in 1.4 million tweets under the Hadoop framework. The PSWAP approach takes approx. 7,786 s to analyze sarcasm in 1.4 million tweets without the Hadoop framework and takes approx. 2,663 s to analyze sarcasm in 1.4 million tweets under the Hadoop framework. Finally, we combined all three approaches and ran with 1.4 million tweets. Then, we compared the elapsed time under the Hadoop framework vs without the Hadoop framework for all three combined approaches as shown in Fig. 14 and it takes approx. 11,609 s to analyze sarcasm in 1.4 million tweets without the Hadoop framework (indicated with the solid line) and takes approx. 4,147 s to analyze sarcasm in 1.4 million tweets under the Hadoop framework (indicated with the dotted line).

5.5. Statistical evaluation metrics

There are three statistical parameters, namely *precision*, *recall* and *F-score*, which are used to evaluate our proposed approaches. *Precision* shows how much relevant information is identified correctly and *recall* shows how much extracted information is relevant. *F-score* is the harmonic mean of *precision* and *recall*. Eqs. 6, 7, and 8 shows the formula to calculate *precision*, *recall* and *F-score*, respectively:

$$Precision = \frac{T_p}{T_p + F_p} \tag{6}$$

$$Recall = \frac{T_p}{T_p + F_n} \tag{7}$$

$$F - Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{8}$$

where T_p is true positive, F_p is false positive, and F_n is false negative.

Experimental datasets consist of a mixture of sarcastic and non-sarcastic tweets. In this paper, we assume the tweets with the hashtag sarcasm or sarcastic (#sarcasm or #sarcastic) as sarcastic tweets. The datasets consist of a total of 1.4 million tweets. Among these tweets, 156,000 were sarcastic and the rest was non-sarcastic. Experimental results in terms of *precision*, *recall* and $F - score$ was the same under both the Hadoop and the non-Hadoop framework. The only difference was algorithm processing time due to the parallel architecture of HDFS. Experimental results are shown in Table 5.

5.6. Discussion on experimental results

Among the six proposed approaches, PBLGA and IWS were earlier implemented and discussed in [19] with a small set of test data (approx. 3,000 tweets for each experiment) and deployed in a non-Hadoop framework. In this work, we deployed PSWAP (novel approach) along with PBLGA and IWS in both a Hadoop and non-Hadoop framework to check the efficiency in terms of time. PBLGA generates four lexicon files, namely positive sentiment, negative situation, positive situation, and negative sentiment, using 156,000 sarcastic tweets. The PBLGA algorithm used 1.45 million tweets as test data. While testing, PBLGA checks each tweet's structure for the contradiction between positive sentiment and negative situation and vice versa to classify them as sarcastic or non-sarcastic. For 1.45 million tweets, PBLGA takes approx. 3386 s in the non-Hadoop framework and it takes approx. 1,400 s in the Hadoop framework. PBLGA consumes most of the time to access the four lexicon files for every tweet to meet the condition of tweet structure. IWS does not require any training set to identify tweets as sarcastic. Therefore, it takes the minimal processing time in both frameworks (25 s for the without Hadoop and 9 s for the Hadoop framework). PSWAP requires a list of antonym pairs for noun, adjective, adverb, and verb to identify sarcasm in tweets. Therefore, it takes approx. 7,786 s for 1.45 million tweets in the non-hadoop framework and approx. 2,663 s for 1.45 million tweets in the Hadoop framework. PSWAP consumes most of the time in searching antonym pairs for all four tags (noun, adjective, adverb, and verb) for every tweet. Finally, we combined all three approaches together and tested. In the combined approach, the *F*-score value attained is 97%, but execution time is more as it checks all three approaches sequentially for every tweet until each one is satisfied to detect sarcasm.

Three more novel algorithms were proposed, namely TCUF, TCTDF and LDC. These three algorithms are implemented using conventional methods with small datasets. Presently, there are no sufficient datasets available with us to deploy these algorithms under the Hadoop framework. TCUF requires a corpus of universal facts. The accuracy of this approach is dependent on the universal facts set. We crawled approximately 5,000 universal facts from Google and Wikipedia for experimentation. TCTDF requires a corpus of time-dependent facts. Accuracy of this approach is dependent on the time-dependent facts. Presently, we trained TCTDF with 10,000 news article headlines as time-dependent facts. LDC requires Twitter users' profile information and their past tweet history. In this work, we tested LDC using ten Twitter users profile and their past tweet history.

6. Conclusion and future work

Sarcasm detection and analysis in social media provides invaluable insight into the current public opinion on trends and events in real time. In this paper six algorithms, namely PBLGA, IWS, PSWAP, TCUF, TCTDF, and LDC, were proposed to detect sarcasm in tweets collected from Twitter. Three algorithms were run with and without the Hadoop framework. The running time of each algorithm was shown. The processing time under the Hadoop framework with data nodes reduced up to 66% on 1.45 million tweets.

In the future, sufficient datasets suitable for the other three algorithms namely LDC, TCUF and TCTDF need to be attained and deployed under the Hadoop framework.

References

[1] D. Chaffey, Global Social Media Research Summary 2016. URL ⟨http://www.smartinsights.com/social-media-marketing/social-media-strategy/new-global-social-media-research/⟩.
[2] W. Tan, M.B. Blake, I. Saleh, S. Dustdar, Social-network-sourced big data analytics, Internet Comput. 17 (5) (2013) 62–69.
[3] Z.N. Gastelum, K.M. Whattam, State-of-the-Art of Social Media Analytics Research, Pacific Northwest National Laboratory, 2013, pp. 1-9.
[4] P. Zikopoulos, C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Osborne Media, 2011.
[5] E. Riloff, A. Qadir, P. Surve, L. De Silva, N. Gilbert, R. Huang, Sarcasm as contrast between a positive sentiment and negative situation, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2013, pp. 704–714.
[6] Hadoop. URL ⟨http://hadoop.apache.org/⟩.
[7] S. Fitzgerald, I. Foster, C. Kesselman, G. Von Laszewski, W. Smith, S. Tuecke, A directory service for configuring high-performance distributed computations, in: Proceedings on High Performance Distributed Computing, IEEE, 1997, pp. 365–375.
[8] J. Dean, S. Ghemawat, Mapreduce: simplified data processing on large clusters, Commun. ACM 51 (1) (2008) 107–113.
[9] S. Hoffman, Apache Flume: Distributed Log Collection for Hadoop, Packt Publishing Ltd, 2013.
[10] Flume. URL ⟨http://flume.apache.org/⟩.
[11] K. Shvachko, H. Kuang, S. Radia, R. Chansler, The Hadoop distributed file system, in: Proceedings of 26th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2010, pp. 1–10.
[12] A. Thusoo, J.S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, R. Murthy, Hive: a warehousing solution over a map-reduce framework, Proc. VLDB Endow. 2(2) (2009) 1626–1629.
[13] S.M. Thede, M.P. Harper, A second-order hidden Markov model for part-of-speech tagging, in: Proceedings of the 37th Annual Meeting on Computational Linguistics, ACL, 1999, pp. 175–182.
[14] D. Klein, C.D. Manning, Accurate unlexicalized parsing, in: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics, ACL, 2003, pp. 423–430.
[15] K. Park, K. Hwang, A bio-text mining system based on natural language processing, J. KISS: Comput. Pract. 17 (4) (2011) 205–213.
[16] Q. Mei, C. Zhai, Discovering evolutionary theme patterns from text: an exploration of temporal text mining, in: Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, ACM, 2005, pp. 198–207.
[17] B. Liu, Sentiment analysis and opinion mining, Synth. Lect. Hum. Lang. Technol. 5 (1) (2012) 1–167.
[18] R. González-Ibánez, S. Muresan, N. Wacholder, Identifying sarcasm in twitter: a closer look, in: Proceedings of the 49th Annual Meeting on Human Language Technologies, ACL, 2011, pp. 581–586.
[19] S.K. Bharti, K.S. Babu, S.K. Jena, Parsing-based sarcasm sentiment recognition in twitter data, in: Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), ACM, 2015, pp. 1373–1380.
[20] E. Lunando, A. Purwarianti, Indonesian social media sentiment analysis with sarcasm detection, in: International Conference on Advanced Computer Science and Information Systems (ICACSIS), IEEE, 2013, pp. 195–198.
[21] P. Tungthamthiti, S. Kiyoaki, M. Mohd, Recognition of sarcasm in tweets based on concept level sentiment analysis and supervised learning approaches, in: 28th Pacific Asia Conference on Language, Information and Computation, 2014, pp. 404–413.
[22] I. Ha, B. Back, B. Ahn, Mapreduce functions to analyze sentiment information from social big data, Int. J. Distrib. Sens. Netw. 2015 (1) (2015) 1–11.
[23] Twitter streaming api. URL ⟨http://apiwiki.twitter.com/⟩, 2010.
[24] J. Kalucki, Twitter streaming api. URL ⟨http://apiwiki.twitter.com/Streaming-API-Documentation/⟩, 2010.
[25] A. Bifet, E. Frank, Sentiment knowledge discovery in twitter streaming data, in: 13th International Conference on Discovery Science, Springer, 2010, pp. 1–15.
[26] Z. Tufekci, Big questions for social media big data: representativeness, validity and other methodological pitfalls, arXiv preprint arXiv:1403.7400.
[27] A.P. Shirahatti, N. Patil, D. Kubasad, A. Mujawar, Sentiment Analysis on Twitter Data Using Hadoop.
[28] R.C. Taylor, An overview of the Hadoop/mapreduce/hbase framework and its current applications in bioinformatics, BMC Bioinform. 11 (Suppl 12) (2010) 1–6.
[29] M. Kornacker, J. Erickson, Cloudera Impala: Real Time Queries in Apache Hadoop, for Real. URL ⟨http://blog⟩. cloudera. com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real.
[30] D. Davidov, O. Tsur, A. Rappoport, Semi-supervised recognition of sarcastic sentences in twitter and amazon, in: Proceedings of the Fourteenth Conference on Computational Natural Language Learning, ACL, 2010, pp. 107–116.
[31] E. Filatova, Irony and sarcasm: Corpus generation and analysis using crowdsourcing, in: Proceedings of Language Resources and Evaluation Conference, 2012, pp. 392–398.
[32] R.J. Kreuz, R.M. Roberts, Two cues for verbal irony: hyperbole and the ironic

tone of voice, Metaphor Symb. 10 (1) (1995) 21–31.
[33] R.J. Kreuz, G.M. Caucci, Lexical influences on the perception of sarcasm, in: Proceedings of the Workshop on Computational Approaches to Figurative Language, ACL, 2007, pp. 1–4.
[34] O. Tsur, D. Davidov, A. Rappoport, Icwsm—a great catchy name: Semi-supervised recognition of sarcastic sentences in online product reviews, in: Proceedings of International Conference on Weblogs and Social Media, 2010, pp. 162–169.
[35] J.W. Pennebaker, M.E. Francis, R.J. Booth, Linguistic Inquiry and Word Count: Liwc 2001, vol. 71, no. 1, Lawrence Erlbaum Associates, Mahway, 2001, pp. 1–11.
[36] C. Strapparava, A. Valitutti, et al., Wordnet affect: an affective extension of wordnet, in: Proceedings of Language Resources and Evaluation Conference, vol. 4, 2004, pp. 1083–1086.
[37] F. Barbieri, H. Saggion, F. Ronzano, Modelling sarcasm in twitter a novel approach, in: Proceedings of the 5th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, 2014, pp. 50–58.
[38] P. Carvalho, L. Sarmento, M.J. Silva, E. De Oliveira, Clues for detecting irony in user-generated contents: oh...!! it's so easy;-), in: Proceedings of the 1st International CIKM Workshop on Topic-Sentiment Analysis for Mass Opinion, ACM, 2009, pp. 53–56.
[39] D. Tayal, S. Yadav, K. Gupta, B. Rajput, K. Kumari, Polarity detection of sarcastic political tweets, in: Proceedings of International Conference on Computing for Sustainable Global Development (INDIACom), IEEE, 2014, pp. 625–628.
[40] A. Rajadesingan, R. Zafarani, H. Liu, Sarcasm detection on twitter: a behavioral modeling approach, in: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, ACM, 2015, pp. 97–106.
[41] A. Utsumi, Verbal irony as implicit display of ironic environment: distinguishing ironic utterances from nonirony, J. Pragmat. 32 (12) (2000) 1777–1806.
[42] C. Liebrecht, F. Kunneman, A. van den Bosch, The perfect solution for detecting sarcasm in tweets# not, in: Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, ACL, New Brunswick, NJ, 2013, pp. 29–37.
[43] M.P. Marcus, M.A. Marcinkiewicz, B. Santorini, Building a large annotated corpus of English: the Penn treebank, Comput. Linguist. 19 (2) (1993) 313–330.
[44] A. Esuli, F. Sebastiani, Sentiwordnet: A publicly available lexical resource for opinion mining, in: Proceedings of Language Resources and Evaluation Conference, 2006, pp. 417–422.
[45] N. Ide, K. Suderman, The american national corpus first release, in: Proceedings of Language Resources and Evaluation Conference, Citeseer, 2004.
[46] N. Ide, C. Macleod, The american national corpus: a standardized resource of American English, in: Proceedings of Corpus Linguistics, 2001.
[47] E. Charniak, Statistical techniques for natural language parsing, AI Mag. 18 (4) (1997) 33–43.
[48] J. Perkins, Python Text Processing with NLTK 2.0 Cookbook, Packt Publishing Ltd, 2010.
[49] D. Rusu, L. Dali, B. Fortuna, M. Grobelnik, D. Mladenic, Triplet extraction from sentences, in: Proceedings of the 10th International Multiconference on Information Society—IS, 2007, pp. 8–12.

Santosh Kumar Bharti is currently pursuing his Ph.D. in Computer Science & Engineering from National Institute of Technology Rourkela, India. His research interest includes opinion mining and sarcasm sentiment detection.

Bakhtyar Vachha is currently pursuing his M.Tech in Computer Science & Engineering from National Institute of Technology Rourkela, India. His research interest includes network security and big data.

Ramkrushna Pradhan is currently pursuing his M.Tech duel degree in Computer Science & Engineering from National Institute of Technology Rourkela, India. His research interest includes speech translation, social media analysis and big data.

Korra Sathya Babu is working as an Assistant Professor in the Department of Computer Science & Engineering, National Institute of Technology Rourkela, India.

Sanjay Kumar Jena is working as Professor in the Department of Computer Science & Engineering, National Institute of Technology Rourkela, India.