SARCASM DETECTION ON TWITTER DATA

Manasa Kashyap Harinath   Sravanthi Avasarala   Siddhi Khandge

Department of Computer Science
University of Central Florida

***Abstract*— The goal of our project is to implement a Natural language solution for detecting the sarcasm on Twitter data. The end result of our project is detection these sarcastic sentences with more accuracy by using three different classifiers. Detecting the sarcasm from the raw data directly depends upon how well the model has trained and learnt. This concept of detecting the tone of the statement comes under the category of sentiment analysis. Sentiment analysis is an emerging tool which s currently used in various areas. The traditional approach of detecting sarcasm includes Bayesian model averaging, pattern-based approach etc. The drawback of these approaches is poor accuracy and limited to less dataset. In this paper we have restricted our tool to be applicable to dataset from twitter. The algorithm developed and used in this paper uses three different classifiers i.e. SVM, logistic regression, random forest ad comparisons were made among them. We plan to implement MVME algorithm and augment the matrix produced by MVME algorithm to classifiers and produce the results.**

***Keywords*— Sentiment analysis, sarcasm detection, SVM, logistic regression, random forest, MVME.**

I. INTRODUCTION

Cognitively speaking, sarcasm is one of the complicated forms of Human interaction. Therefore it is hard for a system to understand, identify the gap between its literals and analyze the complete statement. Human brain can easily understand raw data from previous experiences, but that is not the case for Artificial Intelligence. Detecting such type of sentiments in any platform like face book, twitter etc is very important for modern business needs. Sentiment analysis also knows as opinion mining or emotion AI is a field that aims to determine "the attitude of a speaker or writer on some topic". The main process of sentiment analysis is to determine the polarity of the sentence. Here polarity describes the adjectives like happy, angry, sarcastic etc. There are three main categories in which sentimental analysis has been divided i.e. knowledge based techniques, statistical methods and hybrid approaches. Knowledge based techniques classify based on text by different categories of clearly affect words such as happy, sad, peace etc. Statistical methods classify based on machine learning techniques such as Bag of words, support vector machines etc. Hybrid approaches classify on both machine learning and elements from knowledge representation. The paper concentrates on statistical based sentimental analysis.

The word "Sarcasm" is a French word which means "tear flesh" or "grind teeth". The literal meaning is different for each and every sentence. The study has been extensively concentrated in psychological and behavioral sciences, theories explaining how, why and when sarcasm is expressed according to the context which has been established. For example machine learning can't tell whether a smiley face is used to indicate happiness, sad, humor etc. The most common way through which one can analyze, learn any sentence in machine learning is by feeding huge sets of data into computational systems and make them to learn, understand the data. For any sentence sarcasm can be detected by its lexical, pragmatic features of that statement. All these texts should be inter-related to present world so that identifying the classes for a target word would be much easier and more accurate.

Another most fascinating task is the necessity to teach the machine what to do after detecting sarcasm. In recent studies it was proved that experimenting this sarcasm technique through deep learning methods would increase the accuracy to great extent.

In this paper we will discuss the related work, background, problem formulation, classifiers, technical approach, evaluation and results.

II. RELATED WORK

The term sarcasm has been well addressed in studies like psychology, medical, linguists etc. Sarcasm detection is always a challenging field to sentiment analysis because it might need to detect a positive attribute in a negative sentiment. Liu (Liu, 2010) had captured this problem and had done many researches in this context. Furthermore automatic detection of sarcasm pushes us into much more challenging path (Lee, 2008). The attention over the sentimental analysis has been initially started with speech and context related works (D. Davidov, 2010) .Detection of sarcasm for lexical features were identified by (Kreuz, 2007). The first approach for detecting sarcasm in Italian tweets was presented by (Saggion, 2014). Twitter (dataset) is an online social networking which enables users to send or read 140 character-text messages which are called tweets. Opinion mining (sentiment analysis) can be implemented on any type of datasets such as Wikipedia, twitter, Amazon product reviews etc. (Ghosh, 2015) was the first person who had experimented on twitter data set with 900 tweets using two different classifiers i.e. SVM (support vector machine) and SMO (sequential minimal optimization). Befit and Frank (A. Bifet, 2010) discusses the challenges of capturing Twitter data streams. Socher (Socher, 2013) had proposed a model which is based on basic neural bag of words to more structured, recursive and time delay neural networks which are used to train word embeddings. (Riloff E, 2013) had identified sarcasm between positive sentiment and negative sentiment by using boot strap algorithm with SVM classifier. For data set of 3000 tweets they have obtained an F-measure of about 0.48.

III. BACK GROUND

In this paper we have used the data set from own twitter account by using twitter API configuration. For this implementation we have converted all the words to vectors from the raw text to learn the model. The python library called gensim (generate similar) is used to generate a short list of most similar articles of a given article. This library is helpful in producing vectors with deep learning by using word2vec model. Word2vec is efficient for predicting model learning word embeddings from raw text. Word embedding represents a parameterized function mapping words in some language to either high dimensional or low dimensional vectors. For example for a feed-forward neural network, the words are taken from vocabulary as input and embedded into lower dimensional space, which in turn leads as weights of first layer. There are different techniques which can be implemented as embedding for words such as word2vec, Global Vector representation, weighted textual matrix representation etc. In this paper we have mainly concentrated on word2vec words embedding. Vector space model represents a continuous sequence of words which are mapped to nearby points. VSMs have a strong, rich history in NLP. Here all methods depend in some or other way on the Distribution Hypothesis which states that all the words which appears in the same context share same meaning among them which is more elaborately explained by Baroni (Zamparelli, 2010). Word2vec is basically composed into two different categories. The first is Continuous Bag of Word (CBOW) and the later one is Skip-gram model. The first model predicts the target word from the source context words. For example in CBOW model if the source context is: It might rain ----, then it would predict the target word here as: heavily. Skip-gram model is inverse of CBOW model. Here it would predict the source context words from target word. In this paper we had concentrated on CBOW model.

IV. Problem Formulation

Most sentiment analysis models are concerned with classifying a text, based on the positive or negative sentiment associated with that text. Sarcasm is an expression used to convey contempt or expression of irony. Sentiment analysis models are not modeled to encounter sarcasm. Hence, the urge the build a sarcasm detection model. Sarcasm detection has received much attention in the field of linguistics, cognitive research, and psychology as it improves performance of the sentiment analysis model as it gives deep insight about the exact sentiment associated with the text. Social media data such as Twitter exhibit higher level of sarcasm. Hence, we extract data from the twitter and we aim to detect the level of sarcasm in certain number of tweets containing a target word. In our project, the target word used is 'mature' and number of tweets taken into consideration is 1000. We further discuss various machine learning techniques which are applied on these tweets.

V. CLASSIFIERS

1. Support Vector Machine

SVMs are the supervised learning methods used for classification and regression. It was developed in 1990 and has gained a huge popularity since then. SVMs are known to be one of the best classifiers which is

Max Margin Classifier
Support Vector Classifier
Support Vector Machine

Figure 1: Historical appearance in the literature

Support Vector Machine is a classifier which is formally defined by separating hyper plane. For instance, if given datasets which are labeled, the SVM categories the new examples using optimal hyper plane. The primary approach of the SVM algorithm is based on finding this hyper plane which provides the largest minimum distance to the training examples. This distance is most popularly known as margin within SVM theory. This optimized hyper plane maximizes the margin of the training data.

Computation of Optimal Hyper plane:

Given below is the notation used to define the hyper plane,

$$f(x) = \beta_0 + \beta^T x \qquad (1)$$

Where, β_0 is the bias and β is the weight.

We can optimize the hyper plane by scaling the values ofβ_0 and β. We thus obtain infinite representations of hyperplane and the one chosen is,

$$|\beta_0 + \beta^T x| = 1 \qquad (2)$$

Where x represents the training examples closest to the hyper plane. Ideally, the training examples which are closest to the hyper plane are called **Support Vectors.** This representation is known as Canonical hyper plane.

The distance between the point x and the hyper plane (β, β_0) is given by,

$$distance = \frac{|\beta_0 + \beta^T x|}{||\beta||} \qquad (3)$$

Generally, for the canonical hyper plane, the numerator is equal to one and the distance to the support vectors is,

$$distance_{supportvectors} = \frac{|\beta_0 + \beta^T x|}{||\beta||} = \frac{1}{||\beta||} \qquad (4)$$

Let us denote the Margin by M which is twice the distance to the closest examples,

$$M = \frac{2}{||\beta||} \qquad (5)$$

Essentially, maximizing M is equivalent to minimizing the function, $L(\beta)$ subject to some constraints. These constraints model the requirement for the hyperplane to classify the training examples x_i. Hence,

$$min_{\beta,\beta_0} L(\beta) = \frac{1}{2}||\beta||^2 \qquad (6)$$

This can be written as,

$$y_i(\beta^T x_i + \beta_0) \geq 1 \quad \forall i, \qquad (7)$$

Where, y_i represents each of the labels of the training examples.

2. Random Forrest classifier

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The accuracy of the classifier has significantly increased as the result of growing an ensemble of decision trees and allowing them to vote for the most popular class. To generate these ensembles, random vectors are generated that plays an important role in the growth of every tree in the ensemble. Essentially, for kth tree in an ensemble, a random vector θ_k is chosen. This random vector is not dependent on previously chosen random vectors, $\theta_1, \theta_2, \dots \dots . \theta_{k-1}$. But, these random vectors are of the same distribution. The resulting classifier is represented by, $H(x, \theta_k)$ where x is an input vector. After the ensemble of trees is created using the random vectors, each tree casts a vote for a class which is considered as the most popular.

3. Logistic Regression

Logistic regression is a predictive analysis algorithm used to model dichotomous (binary) outcome variables. Logistic regression model is used to estimate the probability of a binary response based on one or more independent variable. The predictor variables can be either continuous or categorical. The logistic function which is also called the sigmoid function takes any real number as input and the output always takes values between zero and one which can be interpreted as probability. The logistic function is given by the following equation:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

We can then express t as follows:

$$t = \beta_0 + \beta_1 x$$

Where x is the predictor variable, t is the dependent variable and $\beta_0$and $\beta_1$are the hypothesis parameters. In this project, we use logistic regression to predict if a given tweet sarcastic (1) or not sarcastic (0). As logistic regression gives output in the form of predictions it is well suited for sarcasm detection.

VI. TECHNICAL APPROACH

Our implementation aims to detect sarcasm in twitter data using three different classifiers, Logistic regression classifier, Random Forrest classifier and Support vector machine classifiers. We compare the performances of these three classifiers based on accuracy, Sensitivity, Specificity and F1 score. In this section, we discuss about the twitter dataset, preprocessing of the data set, Word2vec, MVME algorithm and classification process.

1. Datasets

In our experiment, the dataset used, possess 1000 tweets consisting of target word, ’Mature’. To extract all the sarcastic tweets, we used twitter search API. This API allows us to search tweets by a string or by hash tags. We then separated all the extracted tweets as sarcastic and non-sarcastic tweets represented by 0 and 1 respectively. The reason for having a smaller dataset is the unavailability of tweets and the restrictions implemented by the Twitter API. According to policies of twitter API, tweets that are 5 years older are not available for download for experimental purpose. Twitter API serves 100 requests for every 15 minutes. The data set consists of two types of tweets based on how the target word was used. First, there are tweets which had target word mentioned in a literal sense. These tweets don’t contain any hash tags denoting that they are sarcastic. Due to this, they are considered as non-sarcastic tweets represented by 0. On the contrary, there are tweets in which the target word is used sarcastically. These tweets contained ‘#sarcasm’ or ‘#sarcastic’, which are represented by ‘1’. The tags used in the tweet give us an intuition of the way the target word is used in the language. Due to lack of infrastructure and with the aim of the project in mind, it was deemed unfeasible to choose a very large dataset. Therefore, our project was implemented using only one target word. Since classification of 1000 tweets takes huge amount of time, increasing the number of tweets to be processed would further increase the execution time. Hence, we consider that it is optimum to process 1000 tweets.

2. Preprocessing

Before passing that dataset set to the classifier, we need to prepare the data such that it meets the needs of the classifier. This step is called as preprocessing. This step involves, cleaning of data, removal of unnecessary spaces and punctuations, removal of least frequent words etc. The raw tweet text is tokenized using a library, TweetTokenizer, which is included in the NLTK package. TweetTokenizer is designed specifically for tokenizing tweets. For instance, consider a string, string Process = '@Messi: I hate Messi for being so good!'. Calling method tokenize, tokenize (s1) method would result in, [':', 'I', 'hate', 'Messi', 'for', 'being', 'so', 'good', '!']. For normalization purpose, we remove the target word from the tweets. But, for classification process, we still retain the target word. In the next step, we remove all the hash tags from the tweets. Furthermore, we replace all the numerical tokens with the common token, 2. We then remove all the username mentions i.e., anything which starts with '@' because this would hardly contribute in the classification. All the words are converted into lower case except if all the letters in the word are in uppercase. For instance, 'Good' is replaced by 'good' whereas 'GOOD' is retained as it is. Finally, all the punctuations are removed since they don't contribute in the classification.

3. Vector Representation

It is the process of representing the words into vectors (weights). There are several ways to represent data as vectors depending on the problem and assumptions in the model. For example if we consider any sentence like "I love ice creams", each word of that sentence is assigned a unique binary number. These binary numbers are very useful because we can merge those encodings to get a vector representation of entire textual document.

I= [1, 0, 0, 0]
Love= [0, 1, 0, 0]
Ice= [0, 0, 1, 0]
creams= [0, 0, 0, 1]

Here the words are represented as a vector of dimensions which is equal to the size of vocabulary (4). Such a representation is called one-hot encoding. One main limitation in one-hot encoding is, it can't perform any meaningful comparisons among the words. In order to avoid the limitation we assume the distribution which states that words which appear in same context are semantically closer than the words. A representation where we embedded words in vector space is called word embeddings. So instead of mapping words in one dimensional line one-hot encoding, here the words were distributed across all the dimensions to capture the semantic properties of the words.

I= [0.73, 0.34, 0.52]
love= [0.65, 0.73, 0.1]
ice= [0.73, 0.69, 0.4]
creams= [0.87, 0.79, 0.9]

From above example we can observe that the distribution for words I and creams were quite closer to each other and words love and ice were a bit closer to each other. This form of extracting the relationships or similarities among the words is called word embeddings which was proposed by Mikolov (Mikolov, 2013). There are several methods through which we can obtain this word embedding representation. Word2vec, global vector representation, weighted textual matrix representation. In this paper we have implemented by using word2vec implementation by gensim library.

4. Word2vec

In order to quantify the text, word2vec uses sliding window. Each sliding window there will be a target word for which all other will be preceding around that word. One main factor in this implementation is the window size and main fold learning techniques. The goal of the manifold learning techniques is to 'learn' the low dimensional manifold. The complete concept of word2vec belongs to shadow learning. Here in this paper we have pre-processed the tweets before and were fed into wrod2vec. The main purpose of pre-processing is to remove any noises that would add while converting into vectors. This is an algorithm which takes in all the terms (with repetitions or folds) from a particular document, divides them into sentences and outputs the vectorial representation for each term. The word2vec requires some parameters for initialization. They are size, min-count, content window size and number of parallel threads. All of these parameters have their own default vector values, but in our implementation we have used our own vector values. Here we have used a minimum count as 1 which means the terms which occur less than min-count number of times that are ignored in calculations and size as 400 which denotes the number of dimensions present in vectorial forms. Window is the range in which the terms which are associated with the sentences will occur. The default value for window is 4, but since our data set is much large we have considered 10. Here we have also declared our number of parallel threads as num-workers of value 4. By all these parameters we have trained our word2vec model as have saved the model as “modelWord2vec_"+targetWord.

5. Kernel Matrix

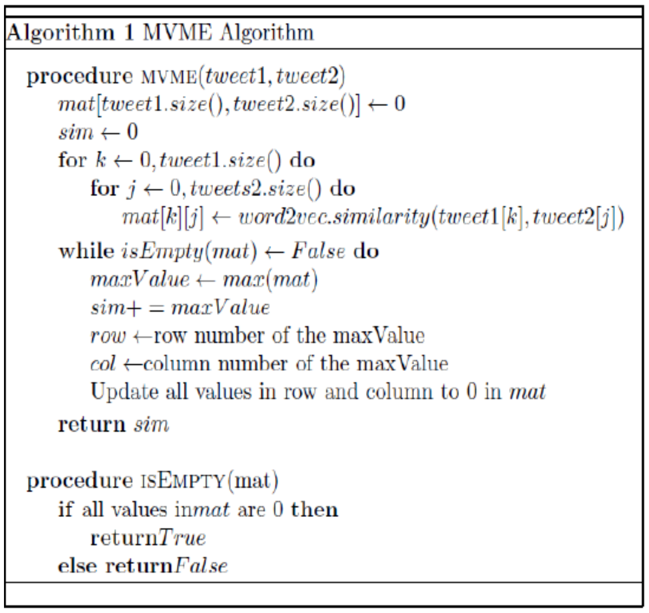
First the preprocessed data (tweets) is split into 80:20 as training and testing set. Then for each pair of tweets the similarity is calculated using the Maximum Valued Matrix Element (MVME) algorithm and saved in the kernel matrix. Basically, the kernel matrix is a 2D array where each value represents the similarity between the two words in the training dataset. This matrix is then mapped to a dictionary and is used as an input for the SVM classifier. The same matrix is then calculated for the test data using the MVME algorithm to calculate the accuracy of the model.

Classification

Before the classification, the dataset is divided into 80:20 ratio, which is the training by testing ratio. This functionality is achieved using train_test_split method available in the sklearn package. Additionally, the train_test_split method accepts the training datasets in the form of data frames. The feature matrix is then converted into data frames using pandas library. All the classifiers are trained with the training data set. Before classification, the test dataset is subjected to preprocessing and then is tested against the classifier and the accuracy and other performance metrics are recorded. All the classifiers implementations are imported from ‘sklearn’ python package.

6. MVME algorithm

MVME algorithm is used to calculate the similarity between two tweets by measuring the word to word similarities for each word pair formed by the inner product between the tweets [1]. The algorithm takes two input parameters that are the two tweets. First, we load the word2vec model that was trained in the previous process. Next, we fill up the similarity matrix ‘*mat*’ where the value ‘mat_{ij}’ represents the similarity value between ith word from tweet 1 and jth word from tweet 2. After filling this matrix, we calculate the similarity coefficient- ‘*sim’* between the two given tweets. For this purpose, we first find the maximum value in ‘*mat’* and its corresponding row and column number- ‘*r’* and ‘*c’*. We add this value to the similarity coefficient ‘*sim’*and mark all values across the row ‘*r’* and the column ‘*c’* as zero. Next we find the maximum value in this modified kernel matrix and continue the same process till all coefficients in the matrix ‘*mat’* are zero, implying the matrix being empty. Finally, this summed similarity value *sim*is returned by the function.



```
Algorithm 1 MVME Algorithm
procedure MVME($tweet1, tweet2$)
    $mat[tweet1.size(), tweet2.size()] \leftarrow 0$
    $sim \leftarrow 0$
    for $k \leftarrow 0, tweet1.size()$ do
        for $j \leftarrow 0, tweets2.size()$ do
            $mat[k][j] \leftarrow word2vec.similarity(tweet1[k], tweet2[j])$
    while $isEmpty(mat) \leftarrow False$ do
        $maxValue \leftarrow max(mat)$
        $sim+ = maxValue$
        $row$ ←row number of the maxValue
        $col$ ←column number of the maxValue
        Update all values in row and column to 0 in $mat$
    return $sim$

procedure ISEMPTY(mat)
    if all values in$mat$ are 0 then
        return$True$
    else return$False$
```

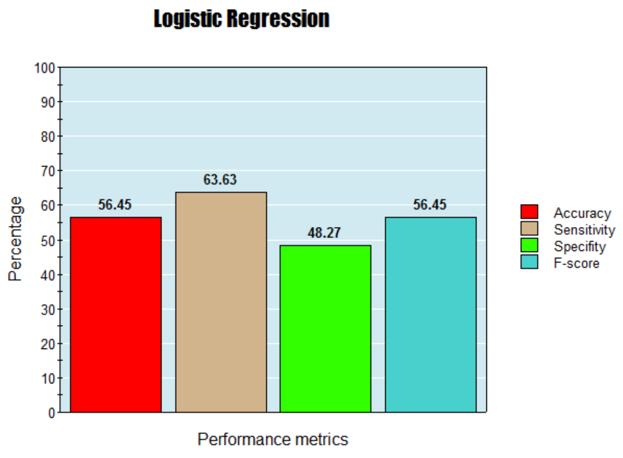
MVME Algorithm

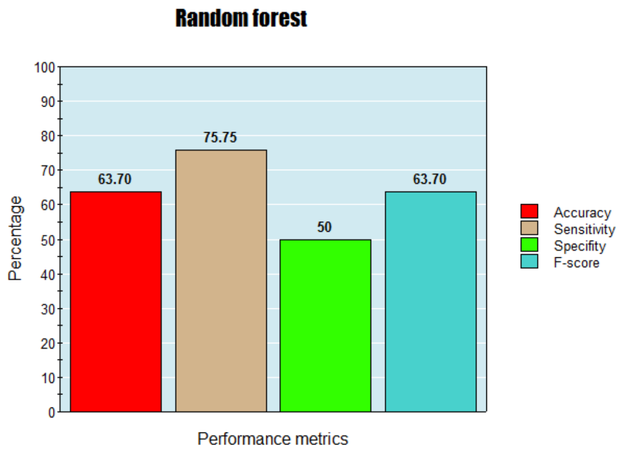
The purpose of using MVME algorithm is that a Support Vector Machines (SVM) classifier with a modified kernel using word embeddings achieves a 7-10% F1 improvement over a strong lexical baseline. The MVME algorithm introduced by

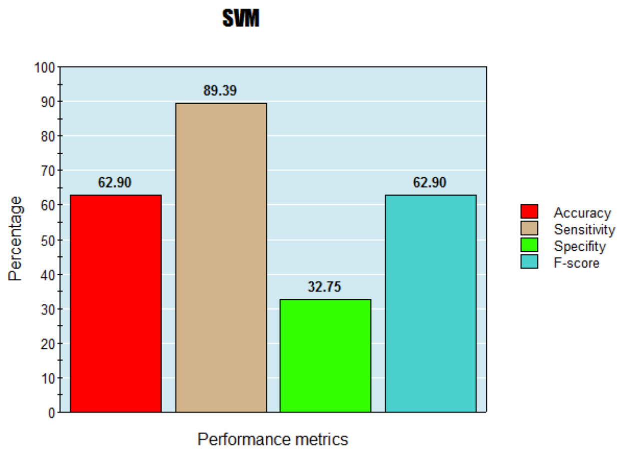
Islam and Inkpen (Inkpen, 2008), has been presented to be useful for computing the similarity of short texts such as the tweets. The notion behind the MVME algorithm is that it finds a one to-one “word alignment” between two statements based on the pair wise word similarity between them. It is only the aligned words that contribute towards the overall similarity score.

VII. Evaluation and Results

This project compares the performances of three classifiers used to detect the sarcasm in the twitter data. We have evaluated these three algorithms on 1000 tweets containing the target word, ‘mature’. The data set, as discussed contains both sarcastic and non-sarcastic tweets. The metrics and graphical representation of performances of all the three algorithms are listed below.







It is observed from the tables and graphs that Random Forrest classifier performs better with the dataset in our approach. Whereas, the performances of SVM and Logistic Regression classifiers were almost similar.

VIII. Future Works

To further improve the accuracy of sarcasm detection in tweets, number of tweets processed can be increased considerably. Furthermore, the dataset designed by Ghosh can be used as the dataset for the experimentation. The dataset consists of 2 million tweets and 37 target words. The classification can be experimented using different target words in the dataset. Various other word embedding models. In our experiment, the word embedding model used is word2Vec, we can instead use GloVe or experiment with other Distributional Semantic models and compare their performances with the twitter data. Additionally, convolution neural network models can be used to analyze the sentiment in the twitter dataset. Emojis in the tweets can also provide deep insights on determining the sarcasm in the twitter dataset. This feature of determining the sarcasm considering the emojis can also be implemented. This implementation can further be extended in pun detection in the tweets.

IX. CONCLUSION

In this paper, we presented the performance analysis of three classification algorithms used to detect sarcasm in the twitter data. We tabulated the performance metrics and graphical representation of the same is shown. Twitter feeds, also called as Tweets, can be analyzed to figure out if the user intends to use sarcasm. This further improvises the efficiency of the sentiment analysis. The scope of our project is to detect sarcasm in twitter data using various classification techniques and comparison of the performances of these classifiers.

X. REFERENCES

1) A. Bifet, E. F. (2010). Sentiment knowledge discovery in twitter streaming data . *13th International Conference on Discovery Science, Springer* , 1-15.

2)D. Davidov, O. T. (2010). Semi-supervised recognition of sarcastic sentences in twitter and amazon. *In Proceedings of the Fourteenth Conference on Computational Natural Language Learning* , 107–116.

3)Ghosh, D. G. (2015). Sarcas-tic or Not: Word Embeddings to Predict the Literal or Sarcastic Meaning of Words. . *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* , 1003-1012.

4)Inkpen, A. I. (2008). Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data* .

5)Kreuz, R. J. (2007). Lexical influences on the perception of sarcasm. *In Proceedings of the Workshop on Computational Approaches to Figura- tive Language* , 1-4.

6)Lee, B. P. (2008). Opinion mining and sentiment analysis. *Foundations and trends in information retrieval* , 1–13.

7)Liu, B. (2010). Sentiment analysis and sub-jectivity. Handbook of natural language processing 2. 627–666.

8)Mikolov, T. (2013). Efficient Estimation of Word Representations in Vector Space. 3-10.

Riloff E, Q. A. (2013). Sarcasm as contrast between a positive sentiment and negative situation. *In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* , 704-714.

9)Saggion, F. R. (2014). *Italian irony detection in twitter: a First approach.* Retrieved 2014, from In The First Italian Conference on Computational Linguis-tics: https://www.researchgate.net/publications/301845949 Automatic Sarcasm Detection A Survey

10)Socher, R. P. (2013). Recursive deep models for semantic compositionality over a sentiment. *In Conference on Empirical Methods in Natural Language Processing* .

11)Zamparelli, M. B. (2010). *In Proceedings of the 2010 Conference on Empirical.* Retrieved 2010, from Association for Computational Linguistics, Cambridge, MA,: http://www.aclweb.org/anthology/D10-1115