Sarcasm Detection in Twitter Data using SVM Classification

Chinmay Jain
Rochester Institute of Technology
Rochester, NY, USA
caj4430@rit.edu

Abstract

In this modern age of social media, people find it convenient to voice their opinion through social media messages, blogs or post. Social media data has become an area of study in the fields of cognitive science, psychology, computational linguistics amongst various others. Sarcasm is a way of communication where the user hides the real meaning by using words in contrasting sense. Finding sarcasm in social media text can be challenging because it is natural to relate the sense of the word in its literal sense. Hence sarcasm detection can be viewed as word sense disambiguation task where the sense to be classified are sarcastic or literal. This is called as Literal/Sarcastic Sense Disambiguation (LSSD) problem. This paper recreates the experiments performed by Ghosh and his colleagues using the algorithm described by them in the same publication.

Keywords: SVM classification, Social media data analysis, Natural Language Processing, Sarcasm detection, Word sense disambiguation

1 Introduction

Sarcasm can be defined as use of irony to mock or convey contempt [2]. Despite each tweet being less than or equal to 140 characters, the amount of information they carry collectively is so much more than expected. Detection of sarcasm in Twitter data can be very helpful. Recently, analysis of tweets has been found useful in predicting the results of presidential elections in the USA, opinion of the users about the corruption scandal in FIFA governing body and various other socio-political topics. These analysis generalize the public opinion on a topic and predict a rough estimate of the output. Detection of sarcasm can also be considered as a sentiment analysis formulation, where the true sentiment of the user is used for detection of sarcasm. People tend to use sarcasm for various reasons, one of which is humor. Humans have a natural tendency to use humor in their day to day social interactions [5]. It helps them socialize with others by use of laughter. Sarcasm can be blunt or funny and can be used to cause discomfort to others. Sarcasm also helps in transmission of messages in a covert way. Hence detection of sarcasm is important and necessary.

Current approaches try to detect sarcasm by considering the complete utterance of the sentence [3][4] whereas in these experiments we try to detect sarcasm for a particular target word. Sarcasm would be detected by classifying the use this target word in the sentence as sarcastic or literal. Support Vector Machines (SVM) are used to perform the task of classification. For the SVM classification I augment my own kernel matrix, which will be explained in further sections. This kernel matrix capture the similarity coefficients between two tweets based upon the word2vec embedding of the words in each tweet. The entire algorithm will be discussed in details in section 4.

The remainder of the paper is structured as follow: Section 2 talks about the dataset used in the experiment. The experiments are described in Section 3. The results of the experiments are discussed in Section 4. A brief conclusion as well as the future scope is discussed in Section 5.

2 Dataset

The dataset used for the experiment was designed by Ghosh and his colleagues while performing their experiments for the *'Sarcastic or Not: Word Embeddings to Predict the Literal or Sarcastic Meaning of Words'*[1] published as part of the Empirical Methods for Natural Language Processing conference 2015. However, despite of having the complete dataset with about 2 million tweets, my

own experimental dataset consist of only 406 tweets i.e. about 0.02% of the original dataset.

The original dataset consist of 37 target words, however for my experiments I choose only two target words. The target words I selected are 'mature' and 'joy'. Both the words had consistent performance across all the implementations. The classification accuracies achieved were high and not much difference is observed in multiple runs. In my dataset I have 339 tweets for the target word 'joy' and 68 tweets for the target word 'mature'. Second reason for having a smaller dataset is the unavailability of tweets and the restrictions implemented by the Twitter API. According to Twitter policies, tweets that are older than 5 years of their posting dates are not available for download for experimental purpose. However a user can query his own history and use those tweets for his experiments. Twitter API serves 100 request for every 15 minutes. That being said, it would approximately take 50 hours to download the complete dataset. Due to lack of infrastructure and with the aim of the project in mind, it was deemed unfeasible to go down this approach. Hence I selected few target words that performed consistently well in the original experiments and used them for my experiments.

Target Word	Mature	Joy
Sarcastic	24	126
Literal	20	108
Sentiment	24	104
Total	68	338

Table 1 Data distribution for target word - 'Mature' and 'Joy'

The data consists of three types of tweet and this classification is based upon the way the target word is used in the tweets. First, we have the tweets where the target word is used in its most natural way i.e. the literal way. These tweets were not labelled or hash tagged by the user. Next, we have tweets that use the target in a sarcastic sense. These tweets were selected on the basis of two criteria; 1. They have the target word in them and 2. They were labelled #sarcastic or #sarcasm by the user. This leads us to the third types of tweets available in the dataset. These are the tweets that contain the target word and are labelled with sentiments. These sentiments depicts the use of the target word in a positive or negative way. The tags used in the tweet gives us an intuition of the way the target word is used in the language.

The tweets collected are labelled with the three classes. The classes are 'S', 'L' and 'L_{sent}'. 'S' denotes the sarcastic tweets, 'L' denotes the target word being used in the literal sense and 'L_{sent}' represents tweets that were labelled with sentiments. Hence the classification task now simplifies to S vs L and S vs L_{sent} for each target word. The distribution of the data is as shown in *table 1*.

3 Methodology

We perform the binary classification to detect the use of sarcasm by probing the target word and its usage in the tweet. For this purpose we need to train for classifier model. However before we pass our data to the classifier we need to prepare the data so as meet the needs of the classifier. For this purpose we need to preprocess the data. In preprocessing, we clean each tweet and create a list of words for each tweet. So in whole, the training data is a list of lists. Each instance in the dataset is in the form-*'t c tweet'*, where 't' is the target word, '*c*' is the class of the tweet and '*tweet*' being the raw text of the tweet. The distribution of training instances across all the classes was equal and is as shown in *figure 1*

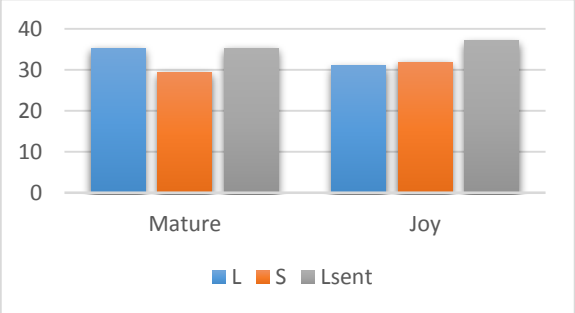

Figure 1 Graph showing percentage distribution of training data

3.1 Preprocessing

Each tweet was preprocessed to extract the features that will be used for classification later. First the raw text of the tweets is tokenized using the NLTK's TweetTokenizer [6]. There are lot options available in NLTK framework for tokenization, but I used this tokenizer as it is designed specifically to tokenize Twitter data. Next I remove all the hash tags from the tweets. This is to normalize the sarcastic tweets as the classifier then tries to look for '#sarcasm' to classify the sarcastic tweets. For a general classification implementation, we would also remove the target word from the tweets, but for our experiments we keep the target word. Then we replace all numeric tokens

in the tweets with a common token- the number '2'. Next we align the case of all the tokens in the tweet. By aligning the case, we change the case of all the tokens to lower case except if the token consist of all the letters in uppercase. For example, 'Never' would be aligned to 'never' whereas 'NEVER' would be kept as it is. Now we need to remove all the usernames mentioned in the tweet. The reason behind this is to remove all the named entities that do not help us capture the relationship between the words and the target word. Next I removed all the punctuations from the tweets. The code even cleaned punctuation marks like the ellipsis so as to capture word to target relation efficiently.

3.2 Word2Vec

Next we need to train the word vectors so as to capture the similarity between the words present in the tweets. I used the word2vec [7] model implemented by Gensim [8]. The list of tweets that was preprocessed is fed into the word2vec model to generate the word embedding for all the words in the bag of words. This trained model is then saved for future purpose. I am training my own vectors for these experiments, however there are pre trained vectors that are available as open source for general purpose use.

The parameters used for training the vectors are minimum word count as 1, context window size of 10 words, number of parallel threads for training the model is 4 and downsampling was 0.0001. The time taken to train the vectors for the target word depend directly on the size of the training data provided.

3.3 Kernel Matrix

Kernel matrix is a data structure that consist of the kernel distance between each pair of training instances. This kernel matrix is then augmented in to Support Vector Classifier from the Scikit / SK Learn [9] library in python. The kernel distance between two tweets is calculated using the Maximum Valued Matrix Element (MVME) algorithm designed by Islam and Inkpen [10]. 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. However the MVME algorithm was modified for my experiments. The algorithm used is shown in *figure 2*.

The algorithm takes two input parameter, those being the two tweets in question. First we load the word2vec model that we trained in the previous step. Next we fill up the similarity matrix '*mat*' where the value 'mat_{ij}' represents the similarity value between i^{th} word from tweet 1 and j^{th} word from tweet 2. After filling this matrix we move on to calculate the similarity coefficient-'*sim*' between the tweets. To do so 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 and mark all values across the row '*r*' and the column '*c*' as zero. We now find the maximum value in this modified matrix and continue the same process till all coefficients in the matrix '*mat*' are zero, signifying the matrix being empty. We returned this summed similarity value as the similarity coefficient between the two tweets.

3.4 Classification

After the kernel matrix has been filled using the MVME algorithm, we augment this matrix into the SVM classifier of the libSVM library and trained the classifier using supervised learning technique. The classification model was cross validated with 5 fold verification before being tested. The data was split in 80:20 ratio for training and testing task. Because of the limited size of the dataset, I replicated the training data to increase the size of training data. Then I preprocessed the testing data and use this classifier to calibrate the accuracy of the classifier. The results of classification will be discussed in the next section.

Algorithm 1 MVME Algorithm

```
procedure MVME(tweet1, tweet2)
    mat[tweet1.size(), tweet2.size()] ← 0
    sim ← 0
    for k ← 0, tweet1.size() do
        for j ← 0, tweets2.size() do
            mat[k][j] ← word2vec.similarity(tweet1[k], tweet2[j])
    while isEmpty(mat) ← False do
        maxValue ← 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 inmat are 0 then
        returnTrue
    else returnFalse
```

Figure 2 MVME Algorithm

4 Results and Discussion

Following the training of the SVM classification, the model was tested against the test data for its classification accuracy. The results obtained are shown in *table 2*.

Target Word	Mature	Joy
Data	56.3%	46.2%
Data X2	42.9%	35.2%
Data X3	36.8%	35.2%
Ghosh et. al. 2015[1]	99%	97%

Table 2 Classification accuracy comparison

The training accuracies shown in *table 2* are the average accuracies after performing 5 iterations of the experiments. As seen in the table the classification accuracy dropped after replication of data. This is consistent with the fact that since the distribution of training data was unequal, the trained model is over fitted to the training data and hence it gave lower accuracies. It can also be inferred that the target word 'joy' outperformed the target word 'mature' in all runs of the experiment. That is down to the size of data available for each target word. More the data, more accurate the word similarities captured by word2vec and hence better similarity of calculations. That being said, the correctness of the algorithm implemented and the classification model used is apt to the original research conducted in the same field.

The classification accuracies achieved are significantly lower than the classification accuracies achieved by Ghosh et. al. 2015[1]. This is down to the fact that the training data used in this set of experiments is significantly lower than the size of training data used by Ghosh et.al. 2015[1].

5 Future Work

The task of sarcasm detection was reformulated as the Literal Sarcastic Sense Disambiguation problem (LSSD). LSSD can be viewed as variant of the Word Sense disambiguation problem. The experimental setup used is more efficient than the results show. For future experiment, the dataset has to be increased considerably, at least up to 80% of the original dataset. Experimentation with different target words would definitely be on agenda for next set of experiments. Another facet to improve upon would the word embeddings used. In this experiment, word2vec is used, we can experiment with other embeddings like Global Vector representation [11] or Weighted Textual Matrix Factorization [12]. It would be also interesting to see how target words perform when the trained vectors are added with vectors obtained by training the opposite target word and the summed vectors are used for classification.

6 Acknowledgment

Sincere thank you to Debanjan Ghosh and his colleagues for sharing their dataset and guidance during the course of this project.

Reference

1. Ghosh, D., Guo, W., & Muresan, S. (n.d.). Sarcastic 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.
2. Oxford Dictionary.
3. Dmitry Davidov, Oren Tsur, and Ari Rappoport. 2010. Semi-supervised recognition of sarcastic sentences in twitter and amazon. *In Proceedings of the Fourteenth Conference on Computational Natural Language Learning, CoNLL '10.*
4. Diana Maynard and Mark A Greenwood. 2014. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. *In Proceedings of LREC.*
5. David Dunning, Self-Insight: Road Blocks and Detours on the Path to Knowing Thyself: *(Kruger, Gordon, Kuban) (page 129).*
6. NLTK Book
7. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. *Efficient estimation of word representations in vector space.*
8. Rehurek, R., & Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, 45-50.
9. Scikit-learn: machine learning in Python - scikit-learn 0.16.1 documentation

   http://scikit-learn.org/stable/
10. Aminul Islam and Diana Inkpen. 2008. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD), 2(2):10.*
11. Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empirical Methods in Natural Language Processing (EMNLP 2014).*
12. Weiwei Guo and Mona Diab. 2012b. Modeling sentences in the latent space. *In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, pages 864–872. Association for Computational Linguistics.*