Sarcastic or Not: Word Embeddings to Predict the Literal or Sarcastic Meaning of Words

Debanjan Ghosh[§] and **Weiwei Guo**[†] and **Smaranda Muresan**[‡]
[§]School of Communication and Information, Rutgers University, NJ, USA
[†]Department of Computer Science, Columbia University, NY, USA
[‡]Center for Computational Learning Systems, Columbia University, NY, USA
debanjan.ghosh@rutgers.edu, weiwei@cs.columbia.edu, smara@ccls.columbia.edu

Abstract

Sarcasm is generally characterized as a figure of speech that involves the substitution of a literal by a figurative meaning, which is usually the *opposite* of the original literal meaning. We re-frame the sarcasm detection task as a type of word sense disambiguation problem, where the *sense* of a word is either *literal* or *sarcastic*. We call this the Literal/Sarcastic Sense Disambiguation (LSSD) task. We address two issues: 1) how to collect a set of target words that can have either literal or sarcastic meanings depending on context; and 2) given an utterance and a target word, how to automatically detect whether the target word is used in the literal or the sarcastic sense. For the latter, we investigate several distributional semantics methods and show 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.

1 Introduction

Recognizing sarcasm is important for understanding people's actual sentiments and beliefs. For example, failing to recognize the following message as being sarcastic "I love that I have to go back to the emergency room", will lead a sentiment and opinion analysis system to infer that the author's sentiment is positive towards the event of "going to the emergency room". Current approaches have framed the sarcasm detection task as predicting whether a full utterance is sarcastic or not (Davidov et al., 2010; González-Ibáñez et al., 2011; Riloff et al., 2013; Liebrecht et al., 2013; Maynard and Greenwood, 2014).

We propose a re-framing of sarcasm detection as a type of word sense disambiguation problem: given an utterance and a target word, identify whether the *sense* of the target word is *literal* or *sarcastic*. We call this the Literal/Sarcastic Sense Disambiguation (LSSD) task. In the above utterance, the word "love" is used in a sarcastic, non-literal sense (the author's intended meaning being most likely the opposite of the original literal meaning - a negative sentiment, such as "hate"). Two key challenges need to be addressed: 1) how to collect a set of target words that can have a literal or a sarcastic sense, depending on context; and 2) given an utterance containing a target word, how can we determine whether the target word is used in its literal sense (e.g., "I *love* to take a nice stroll in the park every morning"), or in a sarcastic sense (e.g., "I *love* going to the dentist.").

To address the first challenge, we need to identify a set of words from sarcastic utterances, which have a figurative/sarcastic sense (e.g., "love" in the utterance "I love going to the dentist"). We propose a crowdsourcing task where Turkers in Amazon Mechanical Turk (MTurk) platform are given sarcastic utterances (tweets labeled with #sarcasm or #sarcastic hashtags) and are asked to re-phrase those messages so that they convey the author's intended meaning ("I *love* going to the dentist" can be rephrased as "I *hate* going to the dentist" or "I *don't like* going to the dentist"). [1] Given this parallel dataset, we use unsupervised alignment techniques to identify semantically opposite words (e.g., "love" ↔ "hate", "brilliant" ↔ "stupid", "never" ↔ "always"). The words from these pairs that appear in the original sarcastic utterances are then considered as our collection of target words (e.g., "love", "brilliant", "never") that can have both a sarcastic and a literal sense depending on the context (Section 2).

To address the second challenge, we compare several distributional semantics methods generally used in word sense disambiguation tasks (Sec-

[1] utterances and messages are used interchangeably.

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1003–1012,
Lisbon, Portugal, 17-21 September 2015. ©2015 Association for Computational Linguistics.

Target	Sense	Utterance
great	S	. . . starting off the new year great !!!!! sick in bed . . .
	L	. . . you don't need a record label to have great music . . .
	L_{sent}	. . . i'm in love with this song great job justin . . .
proud	S	yay something to be proud of 3rd poorest in the NATION . . .
	L	im filipino with dark brown eye and forever true and proud . . .
	L_{sent}	but i'm proud of all the beliebers AROUND THE WORLD . . .

Table 1: Examples of Targets and their Senses

tion 3). We show that using word embeddings in a modified SVM kernel achieves the best results (Section 4). To collect training and test datasets for each of the target words, we use Twitter messages that contain those words. For the *sarcastic sense* (S), we use tweets that contain the target word and are labeled with the #sarcasm or #sarcastic hashtags. For the *literal sense* (L), we collect tweets that contain the target word and are not labeled with the #sarcastic or #sarcasm hashtags. Table 1 shows examples of two targets words ("great" and "proud") and their sarcastic sense (S) and literal sense (L). In addition, for the *literal sense*, we also consider a special case, where the tweets are labeled with either positive or negative hashtags (e.g., #happy, #sad) as proposed by Gonzalez et al. (2011). We denote these sentiment tweets as L_{sent} (Table 1). Gonzalez et al. (2011) argue that it is harder to distinguish sarcastic from non-sarcastic messages where the non-sarcastic messages contain sentiment. Our results support this argument (97% F1 measure for the best result for S vs. L, compared to 84% F1 for the best result for S vs. L_{sent}; Section 4).[2]

2 Collection of Target Words

To collect a set of target words that can have either literal or sarcastic meaning depending on context, we propose a two step approach: 1) a crowdsourcing task to collect a parallel dataset of sarcastic utterances and their re-phrasings that convey the authors' intended meaning; and 2) unsupervised alignment techniques to detect semantically opposite words/phrases.

Crowdsourcing Task. Given a sarcastic message (SM), Turkers were asked to re-phrase the message so that the new message is likely to express the author's intended meaning (IM). Examples of an original sarcastic message (1) and three messages generated by the Turkers (2) is given below:

(1) [SM] I am so happy that I am going back to the emergency room.

(2) a. [IM_1] I don't like that I have to go to the emergency room again.
b. [IM_2] I am so upset I have to return to the emergency room.
c. [IM_3] I'm so unhappy that I am going back to the emergency room.

From the above examples, we can see that aligning the sarcastic message (SM) to the re-phrasings containing the author's intended meaning generated by the Turkers (IM_1, IM_2, IM_3) will allow us to detect that "happy" can be aligned to "don't like", "upset", and "unhappy". Based on this alignment, "happy" will be considered as a target word for the LSSD task.

We used 1,000 sarcastic messages collected from Twitter using the #sarcasm and #sarcastic hashtags. The Turkers were provided with detailed instructions of the task including a definition of *sarcasm*, the task description, and multiple examples. In addition, for messages that contain one or more sentences and where sarcasm is related to only a part of the message, the Turkers were instructed to consider the entire message in their rephrasing. This emphasis was added to avoid high asymmetry in the length between the original sarcastic message and the rephrasing of the intended meaning. For each original sarcastic message (SM), we asked five Turkers to do the rephrasing task. Each HIT contains 1 sarcastic message, and Turkers were paid 5 cents for each HIT. To ensure a high quality level, only qualified workers were allowed to perform the task (i.e., more than 90% approval rate and at least 500 approved HITs). In this way, we obtained a dataset of 5,000 SM-IM pairs.

Unsupervised Techniques to Detect Semantically Opposite Words/Phrases. We use two methods for unsupervised alignment. First, we use the co-training algorithm for paraphrase detection developed by Barzilay and McKeown (2001). This algorithm is used for two specific reasons. First, our dataset is similar in nature to the parallel

[2] The datasets used in the experiments is available at https://github.com/debanjanghosh/sarcasm_wsd.

monolingual dataset used in Barzilay and McKeown (2001), and thus lexical and contextual information from tweets can be used to extract the candidate targets words for LSSD. For instance, we can align the [SM] and [IM_3] (from the above examples), where except for the words happy and unhappy, the majority of the words in the two messages are anchor words and thus happy and unhappy can be extracted as paraphrases via co-training. To model contextual information, such as part of speech tagging for the co-training algorithm, we used Tweet NLP (Gimpel et al., 2011). Second, Bannard and Callison-Burch (2005) noticed that the co-training method proposed by-Barzilay and McKeown (2001) requires identical bounding substrings and has bias towards single words while extracting paraphrases. This apparent limitation, however, is advantageous to us because we are specifically interested in extracting target words. Co-training resulted in 367 extracted pairs of paraphrases.

We also considered a statistical machine translation (SMT) alignment method - IBM Model 4 with HMM alignment implemented in Giza++ (Och and Ney, 2000). We used Moses software(Koehn et al., 2007) to extract lexical translations by aligning the dataset of 5,000 SM-IM pairs. From the set of 367 extracted paraphrases using Barzilay and McKeown (2001)'s approach, we selected only those paraphrases where the lexical translation scores ϕ (resulted after running Moses) are $\geq$ 0.8. After filtering via translation scores and manual inspection, we obtained a set of 80 semantically opposite paraphrases. Given this set of semantically opposite words, the words that appear in the sarcastic messages were consider our target words for LSSD (70 target words after lemmatization). They range from verbs, such as "love" and "like", adjectives, such as "brilliant", "genius", and adverbs, such as "really".

3 Literal/Sarcastic Sense Disambiguation

Our Literal/Sarcastic Sense Disambiguation (LSSD) task is formulated as follows: given a candidate utterance (i.e., a tweet) that contains a target word t, identify whether the sense of t is *sarcastic* (S) or *literal* (L). In order to be able to solve this problem we need training and test data for each target word that consists of utterances where the target word is used either in the literal sense or the sarcastic sense.

love(26802), like(14995), great(14495), good(11624), really(9825), right(6771), fun(6603), best(6182), better(5960), glad(5748), yeah(5504), nice(4443), awesome(4196), excited(4027), always(3807), happy(3098), cool(2705), amazing(1952), favorite(1883), perfect(1792), wonderful(1749), wonder(1476), lovely(1424), super(1390), fantastic(1369), joy(1176), cute(1007), beautiful(981), sweet(800), hot(729), proud(703), shocked(645), interested(624), brilliant(576), genius(481), attractive(449), mature(427)

Table 2: Target words and # of training instances per sense

3.1 Data Collection

To collect training and test datasets for each of the target words, we use Twitter messages that contain those words. For the *sarcastic sense* (S), we use tweets that contain the target word and are labeled with the #sarcasm or #sarcastic hashtag. For the *literal sense* (L), we collect tweets that contain the target word and are not labeled with the #sarcastic or #sarcasm hashtags. In addition, for the *literal sense* we also consider a special case, where the tweets are labeled with either positive or negative sentiment hashtags (e.g., #happy, #sad). Thus, we consider two LSSD tasks: S vs. L and S vs. L_{sent}, and aim to collect a balanced dataset for each target word.

For the 70 target words (see Section 2), we collected a total of 2,542,249 tweets via Twitter API . We considered a setup where 80% of data is used for training, 10% for development, and 10% for test. We empirically set the number of minimum training instances for each sense of the target word to 400 without any upper restriction. This resulted in 37 target words to be used in the LSSD experiments. Table 2 shows all the target words and their corresponding number of *training* instances for each sense (S and L/L_{sent}). The size of training data ranges from 26,802 for the target word "love" to 427 for the word "mature". As we will see in the results sections, however, the size of the training data is not always the key factor in the LSSD task, especially for the methods that use word embeddings.

3.2 Learning Approaches

We consider two classical approaches used in word sense disambiguation tasks: 1) distributional approaches where each sense of a target word is represented as a context vector derived from the training data; and 2) classification approaches (S

vs. L; S vs. L_{sent}) for each target word.

3.2.1 Distributional Approaches

The Distributional Hypothesis in linguistics is derived from the semantic theory of language usage, i.e., words that are used and occur in the same contexts tend to purport similar meanings (Harris, 1954). Distributional semantic models (DSMs) use vectors that represent the contexts (e.g., co-occurring words) in which target words appear in a corpus, as proxies for meaning representations. Geometric techniques such as cosine similarity are then applied to these vectors to measure the similarity in meaning of corresponding words.

The DSMs are a natural approach to model our LSSD task. For each target word t we build two context-vectors that will represent the two senses of the target word t using the training data: one for the *sarcastic* sense S using the sarcastic training data for t ($\vec{v_s}$) and one for the *literal* sense L using the literal sense training data for t ($\vec{v_l}$).[3] Given a test message u containing a target word t, we first represent the target word as a vector $\vec{v_u}$ using all the context words inside u. To predict whether t is used in a literal or sarcastic sense in the test message u we simply apply geometric techniques (e.g., cosine similarity) between $\vec{v_u}$ and the two sense vectors $\vec{v_s}$ and $\vec{v_l}$, choosing the one with the maximum score.

To create the two sense vectors $\vec{v_s}$ and $\vec{v_l}$ for each of the target words t, we use the positive pointwise mutual information model (PPMI) (Church and Hanks, 1990). Based on t's context words c_k in a window of 10 words, we separately computed PPMI for sarcastic and literal senses using t's training data. The size of the context widow used in DSMs is generally between 5 and 10, and in our experiments we used a window of 10 words since tweets often include meaningful words/tokens at the end of the tweets (e.g., interjections, such as "yay", "ohh"; upper-case words, such as, "GREAT"; novel hashtags, such as "#notreally", "#lolol"; emoticons, such as ":("). We sorted the context words based on the PPMI scores and for each target word t we selected a maximum of 1,000 context words per sense to approximate the two senses of the target word (i.e., the vectors $\vec{v_s}$ and $\vec{v_l}$ for each target word t consist of a maximum of 1,000 words). Table 3 shows some target words and their corresponding context words that were selected based on high PPMI scores.

Targets	Senses	Context Vector
love	S	ignored, being, waking, work, sick, #not
	L	please, follow, ♡, her, :)
	L_{sent}	happy, family, blessed, cute, birthday
fun	S	work, tomorrow, homework, friday, sleep
	L	hope, join, girl, game, friend
	L_{sent}	#friends, #family, weekend, amazing, #christmas
joy	S	working, snow, waking, studying, sick
	L	yesterday, sweet, special, prayer, laughter
	L_{sent}	wishing, warmth, love, christmas, peace

Table 3: Target words and their context words

To predict whether t is used in a literal or sarcastic sense in the test message u we simply apply the cosine similarity to the $\vec{v_u}$ (vector representation of the target word t in the test message u) and the two sense vectors $\vec{v_s}$ and $\vec{v_l}$ of t, choosing the one with the maximum score. All vector elements are given by the tf-idf values of the corresponding words. This approach, denoted as the "PPMI baseline", is the baseline for our DSM experiments.

Context Vectors with Word Embedding: The above method considers that the context vectors $\vec{v_s}$ and $\vec{v_l}$ of each target word t contain the co-occurring words selected by their PPMI values. We enhance the representation of context vectors to represent each word in the context vector by its word embedding. We experiment with three different methods of obtaining word embeddings: Weighted Textual Matrix Factorization (WTMF) (Guo and Diab, 2012b); *word2vec* that implements the skip-gram and continuous bag-of-words models (CBOW) of Mikolov et al. (2013a), and GloVe (Pennington et al., 2014), a log-bilinear regression model based upon global word-word co-occurrence count in the training corpora.

After removing the tweets that are used as test sets, we build the three word embedding models in an unsupervised fashion with the remaining 2,482,763 tweets from our original data collection (Section 3.1). In each of the three models, each word w is represented by its d-dimensional vector $\vec{w}$ of real numbers, where d=100 for all of the embedding algorithms in our experiments. For the size of the embedding vectors, it is common to use

[3] In the remaining of this section we will only mention L and not L_{sent} for clarity and brevity.

100 or 300 dimensions, with larger dimensions for larger datasets. Our current dataset is smaller than the ones used in other applications of word embeddings (e.g., Pennington et al. (2014) have used billion tweets to create word embedding) so we opted for 100 dimensional vectors. Below are the short descriptions of the three word embedding models:

- *Weighted Textual Matrix Factorization (WTMF):* Low-dimensional vectors have been used in WSD tasks, since they are computationally efficient and provide better generalization than surface words. A dimension reduction method is Weighted Textual Matrix Factorization (WTMF) (Guo and Diab, 2012b), which is designed specifically for short texts, and has been successfully applied in WSD tasks (Guo and Diab, 2012a). WTMF models unobserved words, thus providing more robust embeddings for short texts such as tweets.

- *word2vec Representation:* We use both the Skip-gram model and the Continuous Bag-of-Words (CBOW) model (Mikolov et al., 2013a; Mikolov et al., 2013c) as implemented in the word2vec gensim python library. [4] Given a window size of n words around a word w, the skip-gram model predicts the neighboring words given the current word. In contrast, the CBOW model predicts the current word w, given the neighboring words in the window. We considered a context window of 10 words.

- *GloVe Representation:* GloVe (Pennington et al., 2014) is a word embedding model that is based upon weighted least-square model trained on global word-word co-occurrence counts instead of the local context used by word2vec.

Here, the LSSD task is similar to the baseline: to predict whether the target word t in the test message u is used in a literal or sarcastic sense, we simply use a similarity measure between the $\vec{v_u}$ (vector representation of the target word t in the test message u) and the two sense vectors $\vec{v_s}$ and $\vec{v_l}$ of t, choosing the one with the maximum score. The difference from the baseline is twofold: First, all vectors elements are word embeddings (i.e., 100-d vectors). Second, we use the *maximum-valued matrix-element* (MVME) algorithm introduced by Islam and Inkpen (2008), which has been shown to be particularly useful for computing the similarity of short texts. We modify this algorithm to use word embeddings ($MVME_{we}$). The idea behind the MVME algorithm is that it finds a one-to-one "word alignment" between two utterances (i.e., sentences) based on the pairwise word similarity. Only the aligned words contribute to the overall similarity score.

Algorithm 1 $MVME_{we}$

```
procedure $MVME_{we}(v_s, v_u)$
    $v_{s_{words}} \leftarrow v_s.elements()$
    $v_{u_{words}} \leftarrow v_u.elements()$
    $M[v_{s_{words}}.size(), v_{u_{words}}.size()] \leftarrow 0$
    for $k \leftarrow 0, v_{s_{words}}.size()$ do
        $c_k \leftarrow v_{s_{words}}[k]$
        $\vec{c_k} \leftarrow getEmbedding(c_k)$
        for $j \leftarrow 0, v_{u_{words}}.size()$ do
            $w_j \leftarrow v_{u_{words}}[j]$
            $\vec{w_j} \leftarrow getEmbedding(w_j)$
            $M[k][j] \leftarrow cosine(\vec{c_k}, \vec{w_j})$
        end for
    end for
    while True do
        repeat
            $max \leftarrow getMax(M)$
            $Sim \leftarrow Sim + max$
            $r_m, c_m \leftarrow getRowCol(M, max)$
    ▷ Remove $r_m$ row and $c_m$ column from M
            $remove(M, r_m, c_m)$
        until $max > 0$ Or $M.size() > 0$
    end while
    Return $Sim$
end procedure

procedure GETEMBEDDING(word)
    Return $we_{model}[word]$
end procedure
procedure GETROWCOL(M,max)
    $row, col \leftarrow M.indexOf(max)$
    Return $row, col$
end procedure
```

Algorithm 1 presents the pseudocode of our modified algorithm for word embeddings, $MVME_{we}$. Let the total similarity between $\vec{v_s}$ and $\vec{v_u}$ be Sim. For each context word c_k from $\vec{v_s}$ and each word w_j from $\vec{v_u}$, we compute a matrix where the value of the matrix element M_{jk}

[4] https://radimrehurek.com/gensim/models/word2vec.html

denotes the cosine similarity between the embedded vectors $\vec{c_k}$ and $\vec{w_j}$ [lines 5 -13]. Next, we first select the matrix cell that has the highest similarity value in M (max) and add this to the Sim score [lines 16-17]. Let the r_m and c_m be the row and the column of the cell containing max (maximum-valued matrix element), respectively. Next, we remove all the matrix elements of the r_m-th row and the c_m-th column from M [line 20]. We repeat this procedure until we have traversed through all the rows and columns of M or max = 0 [line 21].

3.2.2 Classification Approaches

The second approach for our LSSD task is to treat it as a binary classification task to identify the sarcastic or literal sense of a target word t. We have two classification tasks: S vs. L and S vs. L_{sent} for each of the 37 target words. We use the libSVM toolkit (Chang and Lin, 2011). Development data is used for tuning parameters.

SVM Baseline: The SVM baseline for LSSD tasks uses n-grams and lexicon-based binary-valued features that are commonly used in existing state-of-the-art sarcasm detection approaches (González-Ibáñez et al., 2011; Tchokni et al., 2014). They are derived from i) bag-of-words (BoW) representations of words, ii) LIWC dictionary (Pennebaker et al., 2001), and iii) a list of interjections (e.g., "ah", "oh", "yeah"), punctuations (e.g., "!", "?"), and emoticons collected from Wikipedia. *CMU Tweet Tokenizer* is employed for tokenization. [5] We kept unigrams unchanged when all the characters are uppercase (e.g., "NEVER" in *"A shooting in Oakland? That NEVER happens! #sarcasm"*) but otherwise words are converted to lower case. We also change all numbers to a generic number token "22". To avoid any bias during experiments, we removed the target words from the tweets as well as any hashtag used to determine the sense of the tweet (e.g., #sarcasm, #sarcastic, #happy, #sad).

SVM with $MVME_{we}$ Kernel: We propose a new kernel $kernel_{we}$ to compute the semantic similarity between two tweets u_r and u_s using the $MVME_{we}$ method introduced for the DSM approach, and the three types of word embeddings (WTMF, word2vec, and GloVe). The similarity measure in the kernel is similar to the algorithm $MVME_{we}$ described in Algorithm 1, but instead of measuring the similarity between the sense vectors of t ($\vec{v_s}$, $\vec{v_l}$) and the vector representation of t in test message ($\vec{v_u}$), now we measure the similarity between two tweets u_r and u_s. For each k-th index word w_k in u_r and l-th index word w_l in u_s we compute the cosine similarity between the embedded vectors of the words and fill up a similarity matrix M. We select the matrix cell that has the highest similarity, add this similarity score to the total similarity Sim, remove the row and column from M that has highest similarity score, and repeat the procedure (similar to Algorithm 1). We noticed that $MVME_{we}$ algorithm carefully chooses the best candidate word w_l in u_s for the w_k word in u_r since w_l is the most similar word to w_k. The algorithm continues the same procedure for all the remaining words in u_r and u_s. The final Sim is used as the kernel similarity between u_r and u_s. We augment this kernel $kernel_{we}$ into libSVM and during evaluation we run supervised LSSD classification for each target word t separately.

4 Results and Discussions

Tables 4 and 5 show the results for the LSSD experiments using distributional approaches and classification-based approaches, respectively. For brevity, we only report the average Precision (P), Recall (R), and F1 scores with their standard deviation (SD) (given by '±'), and the targets with maximum/minimum F1 scores. $w2v_{sg}$ and $w2v_{cbow}$ represent the skip-gram and CBOW models implemented in word2vec, respectively.

Table 4 presents the results of distributional approaches (Section 3.2.1). We observe that the word embedding methods have better performance than the PPMI baseline for both S vs. L and S vs. L_{sent} disambiguation tasks. Also, the average P/R/F1 scores for S vs. L are much higher than for S vs. L_{sent}. Since all tweets with L_{sent} sense were collected using sentiment hashtags (González-Ibáñez et al., 2011), they might be lexically more similar to the S tweets than the L tweets are and thus identifying the sense of a target word t between S vs. L_{sent} is a harder task. In Table 4 we also observe that the average F1 scores between WTMF, $w2v_{sg}$, $w2v_{cbow}$, and GloVe are comparable and between 84%-86%, with $w2v_{sg}$ and $w2v_{cbow}$ achieving slightly higher F1.

Table 5 outlines the LSSD experiments us-

[5] http://www.ark.cs.cmu.edu/TweetNLP/

Expr.	Senses	Avg. P	Avg. R	Avg. F1	Max. F1(Target)	Min. F1(Target)
$PPMI_{bl}$	S	73.5 ± 3.6	84.6 ± 6.0	78.5 ± 3.2	83.9(mature)	68.8(wonder)
	L	83.1 ± 5.0	70.6 ± 5.5	76.1 ± 3.4	82.7(love)	68.3(nice)
	S	67.8 ± 7.0	76.2 ± 13.6	70.4 ± 7.6	81.8(joy)	43.8(like)
	L_{sent}	74.2 ± 7.1	62.7 ± 12.8	66.9 ± 6.6	78.6(joy)	47.1(interested)
WTMF	S	83.0 ± 3.4	87.2 ± 5.4	84.9 ± 2.4	91.4(mature)	78.7(wonder)
	L	87.5 ± 4.4	82.7 ± 4.5	84.9 ± 2.2	90.5(mature)	80.6(nice)
	S	67.4 ± 5.5	86.5 ± 5.1	75.6 ± 3.9	84.4(joy)	65.8(interested)
	L_{sent}	82.1 ± 5.8	58.9 ± 9.7	68.1 ± 7.2	81.5(joy)	50.0(genius)
GloVe	S	83.7 ± 3.6	85.6 ± 5.6	84.5 ± 2.8	90.6(joy)	78.8(sweet)
	L	86.3 ± 4.6	84.0 ± 4.3	85.0 ± 2.5	89.6(joy)	79.2(like)
	S	70.7 ± 5.1	84.3 ± 5.0	76.8 ± 3.9	85.4(joy)	67.1(interested)
	L_{sent}	80.7 ± 5.4	64.7 ± 8.5	71.5 ± 6.1	84.0(joy)	54.7(hot)
$w2v_{sg}$	S	84.9 ± 3.3	87.0 ± 4.8	85.8 ± 2.6	90.9(mature)	80.7(like)
	L	87.5 ± 4.1	85.1 ± 4.0	86.2 ± 2.5	90.7(mature)	80.2(like)
	S	70.8 ± 4.8	85.7 ± 5.1	77.4 ± 4.0	86.7(joy)	68.1(interested)
	L_{sent}	82.2 ± 5.7	64.3 ± 7.8	71.9 ± 5.9	85.4(joy)	57.4(interested)
$w2v_{cbow}$	S	84.9 ± 3.2	86.7 ± 4.7	85.6 ± 2.5	90.9(mature)	80.7(like)
	L	87.3 ± 4.0	85.1 ± 3.8	86.1 ± 2.4	90.7(mature)	80.2(like)
	S	70.7 ± 4.8	85.8 ± 5.0	77.4 ± 4.0	86.4(joy)	68.6(attractive)
	L_{sent}	82.0 ± 5.6	64.0 ± 7.7	71.7 ± 5.8	85.0(joy)	58.7(interested)

Table 4: Evaluation of distributional approaches (PMI and word embedding) for LSSD experiments

Expr.	Senses	Avg. P	Avg. R	Avg. F1	Max. F1(Target)	Min. F1(Target)
SVM_{bl}	S	87.0 ± 3.3	85.6 ± 3.1	86.3 ± 2.7	91.7(yeah)	75.4(sweet)
	L	85.9 ± 2.8	87.1 ± 3.6	86.5 ± 2.8	91.8(yeah)	76.1(sweet)
	S	77.3 ± 4.6	78.2 ± 4.2	77.7 ± 3.8	85.5(love)	68.6(brilliant)
	L_{sent}	77.8 ± 3.7	76.7 ± 6.4	77.1 ± 4.7	85.8(love)	64.6(attractive)
$kernel_{WTMF}$	S	94.1 ± 2.2	94.6 ± 1.8	94.3 ± 1.8	97.3(brilliant)	88.3(joy)
	L	94.6 ± 1.8	94.0 ± 2.3	94.3 ± 1.9	97.2(mature)	87.9(joy)
	S	79.0 ± 4.6	78.8 ± 4.4	78.8 ± 3.8	84.8(mature)	61.0(genius)
	L_{sent}	78.8 ± 3.7	78.9 ± 4.9	78.8 ± 3.6	85.4(mature)	63.5(genius)
$kernel_{GloVe}$	S	95.7 ± 1.6	97.4 ± 1.7	96.5 ± 1.1	99.1(mature)	92.9(glad)
	L	97.4 ± 1.6	95.6 ± 1.7	96.5 ± 1.2	99.1(mature)	92.7(interested)
	S	79.5 ± 3.5	83.1 ± 3.0	81.2 ± 2.8	86.9(joy)	74.2(attractive)
	L_{sent}	82.2 ± 3.0	78.3 ± 4.4	80.2 ± 3.4	86.6(joy)	69.2(attractive)
$kernel_{w2v_{sg}}$	S	96.6 ± 1.1	98.5 ± 0.6	97.5 ± 0.4	99.2(cute)	93.8(interested)
	L	98.5 ± 0.7	96.5 ± 1.2	97.5 ± 0.5	99.2(cute)	93.5(interested)
	S	81.9 ± 3.8	88.1 ± 3.2	84.8 ± 3.0	88.8(love)	74.2(genius)
	L_{sent}	87.0 ± 3.2	80.2 ± 4.7	83.4 ± 3.5	88.8(love)	73.3(genius)
$kernel_{w2v_{cbow}}$	S	96.4 ± 1.0	98.2 ± 1.1	97.3 ± 0.6	99.1(mature)	93.8(interested)
	L	98.2 ± 1.1	96.3 ± 1.1	97.2 ± 0.7	99.1(mature)	93.5(interested)
	S	81.7 ± 3.8	88.6 ± 2.9	84.9 ± 2.8	89.5(love)	74.8(genius)
	L_{sent}	87.4 ± 2.9	79.9 ± 4.8	83.4 ± 3.4	89.2(love)	74.4(genius)

Table 5: Evaluation of classification approaches (SVM_{bl} and $kernel_{we}$) for LSSD experiments

ing the classification approaches (Section 3.2.2): SVM baseline (SVM_{bl}) and SVM using the $kernel_{we}$ with word embeddings ($kernel_{WTMF}$, $kernel_{GloVe}$, $kernel_{w2v_{sg}}$, and $kernel_{w2v_{cbow}}$). The classification approaches give better performance compared to the distributional approaches. The SVM_{bl} is around 7-8 % higher than the $PPMI_{bl}$ and comparable with the word embeddings used in distributional approaches (Table 4). In addition, our new SVM kernel method using word embeddings shows significantly better results when compared to the SVM_{bl} (and distributional approaches). For instance, for the S vs. L task, the average F1 is 96-97%, which is more than 10% higher than SVM_{bl}. Similarly, for S vs. L_{sent} task, F1 scores reported by the kernel using word2vec embeddings are in the range of 83%-84% compared to 77% given by the SVM_{bl}, showing an absolute increase of 7%. As stated earlier, MVME algorithm aligns similar word pairs found in its inputs and this performs well for short texts (i.e., tweets). Thus, the MVME algorithm combined with word embedding in $kernel_{we}$ results in very high F1. Among the word embedding models, word2vec models give marginally better results compared to GloVe and WTMF, and GloVe outperforms marginally WTMF. Similar to Table 4, here, the average F1 scores for S vs. L task are higher than the S vs. L_{sent} results.

In terms of the best and worst performing targets, SVM_{bl} prefers targets with more training data (e.g., “yeah”, “love” vs. “sweet”, “attractive”; see Table 2). In contrast, word embedding models for “joy” and “mature”, two targets with comparatively low number of training instances have achieved very high F1 using both distributional and classification approaches (Table 4 and 5). This can be explained by the fact that for words, such as “joy”, “mature”, “cute”, and “brilliant”, the contexts of their literal and sarcastic sense are quite different, and DSMs and word embeddings are able to capture the difference. For example, observe in the Table 3, negative sentiment words, i.e., “sick”, “working”, “snow” are the context words for targets “joy” and “love”, where as positive sentiment words, such as, “blessed”, “family”, “christmas”, and “peace” are the context words for L or L_{sent} senses. Overall, out of 37 targets, only 5 targets (“mature”, “joy”, “cute”, “love”, and “yeah”) achieved “maximum” F1 scores in various experimental settings (Tables 4 and 5) whereas targets such as “interested”, “genius”, and “attractive” achieved low F1 scores.

In terms of variance in results, SVM results show low SD (0-4%). For distributional approaches, SD is slightly higher (5-8%) for several cases.

5 Related Work

Two lines of research are directly relevant to our work: sarcasm detection in Twitter and application of distributional semantics, such as word embedding techniques to various NLP tasks. In contrast to current research on sarcasm and irony detection (Davidov et al., 2010; Riloff et al., 2013; Liebrecht et al., 2013; Maynard and Greenwood, 2014), we have introduced a reframing of this task as a type of word sense disambiguation problem, where the sense of a word is sarcastic or literal. Our SVM baseline uses the lexical features proposed in previous research on sarcasm detection (e.g., LIWC lexicon, interjections, pragmatic features) (Liebrecht et al., 2013; González-Ibáñez et al., 2011; Reyes et al., 2013). Our analysis of target words where the sarcastic sense is the opposite of the literal sense is related to the idea of “positive sentiment toward a negative situation” proposed by Riloff et al. (2013) and recently studied by Joshi et al. (2015). In our approach, we chose distributional semantic approaches that learn contextual information of targets effectively from a large corpus containing both literal and sarcastic uses of words and show that word embedding are highly accurate in predicting the sarcastic or literal sense of a word (Tables 4 and 5). This approach has the potential to capture more nuanced cases of sarcasm, beyond “positive sentiment towards a negative situation” (e.g., one of our target words was “shocked” which is negative). However, our current framing is still inherently limited to cases where sarcasm is characterized as a figure of speech where the author means the opposite of what she says, due to our approach of selecting the target words.

Low-dimensional text representation, such as WTMF, have been successful in WSD disambiguation research and in computing similarity between short texts (Guo and Diab, 2012a; Guo and Diab, 2012b). word2vec and GloVe representations have provided state-of-the-art results on various word similarity and analogy detection task (Mikolov et al., 2013c; Mikolov et al., 2013b; Pennington et al., 2014). Word embedding based models are also used for other NLP tasks such as dependency parsing, semantic role labeling, POS tagging, NER, question-answering (Bansal et al., 2014; Collobert et al., 2011; Weston et al., 2015) and our work on LSSD is a novel application of word embeddings.

6 Conclusion and Future Work

We proposed a reframing of the sarcasm detection task as a type of word sense disambiguation problem, where the sense of a word is its *sarcastic* or *literal* sense. Using a crowdsourcing experiment and unsupervised methods for detecting semantically opposite phrases, we collected a set of target words to be used in the LSSD task. We compared several distributional semantics methods, and showed that using word embeddings in a modified SVM kernel achieves the best results (an increase of 10% F1 and 8% F1 for S vs. L and S vs. L_{sent} disambiguation task, respectively, against a SVM baseline). While the SVM baseline preferred larger amounts of training data (best performance achieved on the targets words with higher number of training examples), the methods using word embeddings seem to perform well on target words where there might be an inherent difference in the contextual sarcastic and literal use of a target word, even if the training data was smaller.

We want to investigate further the nature and size of training data useful for the LSSD task. For example, to test the effect of larger training dataset, we utilized pre-trained word vectors from GloVe (trained with 2 Billion tweets, using 100 dimensions).[6] For S vs. L disambiguation, the average F1 was 88.9%, which is 7% lower than the result using GloVe on our training set of tweets (much smaller) designed for the LSSD task. This shows the training data utilized to create word embedding models in GloVe probably do not contain enough sarcastic tweets.

Regarding the size of the training data, recall that the unsupervised alignment approach had extracted 70 target words (Section 2), although we have used 37 target words as we did not have enough training data for the remaining targets. Thus, we plan to collect more training data for these targets as well as more target words (especially for the S vs. L_{sent} task). In addition, we plan to improve our unsupervised methods for detecting semantically opposite meaning (e.g., using the IM-IM dataset in addition to the SM-IM dataset).

One common criticism of research based on use of hashtags as gold labels is that the training utterances could be noisy. In other words, tweets might be sarcastic but not have #sarcasm or #sarcastic hashtags. We did a small manual validation on a dataset of 180 tweets from the L_{sent} class using 3 annotators (we asked them to say whether the tweet is sarcastic or not). For cases where all 3 coders agree none of them were considered sarcastic, while when only 2 coders agree 1 tweet out of 180 was considered sarcastic. In future, we plan to perform additional experiments to study the issue of noisy data. We hope that the release of our datasets will stimulate other studies related to the sarcasm detection problem, including addressing the issue of noisy data.

We also plan to study the effect of hyperparameters in designing the DSMs. Recently, Levy et al. (2015) have argued that parameter settings have a large impact on the success of word embedding models. We want to follow their experiments to study whether parameter tuning in PMI based disambiguation can improve its performance.

[6] Downloaded from http://nlp.stanford.edu/projects/glove/

Acknowledgements

This paper is based on work supported by the DARPA-DEFT program. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. The authors thank the anonymous reviewers for helpful comments.

References

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 597–604. Association for Computational Linguistics.

Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2014. Tailoring continuous word representations for dependency parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Regina Barzilay and Kathleen R McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*, pages 50–57. Association for Computational Linguistics.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Kenneth Ward Church and Patrick Hanks. 1990. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

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.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A Smith. 2011. Part-of-speech tagging for twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 42–47. Association for Computational Linguistics.

Roberto González-Ibáñez, Smaranda Muresan, and Nina Wacholder. 2011. Identifying sarcasm in twitter: A closer look. In *ACL (Short Papers)*, pages 581–586. Association for Computational Linguistics.

Weiwei Guo and Mona Diab. 2012a. Learning the latent semantics of a concept from its definition. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 140–144. Association for Computational Linguistics.

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.

Zellig S Harris. 1954. Distributional structure. *Word*, 10:146–162.

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.

Aditya Joshi, Vinita Sharma, and Pushpak Bhattacharyya. 2015. Harnessing context incongruity for sarcasm detection. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 757–762, Beijing, China, July. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

Omer Levy, Yoav Goldberg, and Ido Dagan. 2015. Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

CC Liebrecht, FA Kunneman, and APJ van den Bosch. 2013. The perfect solution for detecting sarcasm in tweets# not.

Diana Maynard and Mark A Greenwood. 2014. Who cares about sarcastic tweets? investigating the impact of sarcasm on sentiment analysis. In *Proceedings of LREC*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Franz Josef Och and Hermann Ney. 2000. Giza++: Training of statistical translation models.

James W Pennebaker, Martha E Francis, and Roger J Booth. 2001. Linguistic inquiry and word count: Liwc 2001. *Mahway: Lawrence Erlbaum Associates*, 71:2001.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12.

Antonio Reyes, Paolo Rosso, and Tony Veale. 2013. A multidimensional approach for detecting irony in twitter. *Language resources and evaluation*, 47(1):239–268.

Ellen Riloff, Ashequl Qadir, Prafulla Surve, Lalindra De Silva, Nathan Gilbert, and Ruihong Huang. 2013. Sarcasm as contrast between a positive sentiment and negative situation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 704–714. Association for Computational Linguistics.

Simo Tchokni, Diarmuid O Séaghdha, and Daniele Quercia. 2014. Emoticons and phrases: Status symbols in social media. In *Eighth International AAAI Conference on Weblogs and Social Media*.

Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.