ELiRF-UPV at SemEval-2017 Task 7: Pun Detection and Interpretation

Lluís-F. Hurtado, Encarna Segarra, Ferran Pla, Pascual Carrasco, José-Ángel González
Universitat Politècnica de València
Camí de Vera sn, 46022, València
{lhurtado|esegarra|fpla|pascargo|jogonba2}@dsic.upv.es

Abstract

This paper describes the participation of ELiRF-UPV team at task 7 (subtask 2: homographic pun detection and subtask 3: homographic pun interpretation) of SemEval2017. Our approach is based on the use of word embeddings to find related words in a sentence and a version of the Lesk algorithm to establish relationships between synsets. The results obtained are in line with those obtained by the other participants and they encourage us to continue working on this problem.

1 Introduction

Pun is a figure of speech that consists of a deliberate confusion of similar words or phrases for rhetorical effect, whether humorous or serious. In (Giorgadze, 2014), the author analyzed, from a linguistic point of view, the pun as one of the categories of wordplay and its manifestation in one-liner jokes in English. Pun is a way of using the characteristics of the language to cause a word, a sentence or a discourse to involve two or more different meanings. Therefore, humorous or any other effects created by puns depend upon the ambiguities of these words.

Pun detection is closely related to the Word Sense Disambiguation (WSD) problem, but in this case we need to select two senses of the pun (Miller and Gurevych, 2015; Miller and Turkovi, 2016).

The interpretation of puns has been subject of study in theoretical linguistics, and has led to a small but growing body of research in computational linguistics. In the task 7 of the SemEval 2017 competition, organizers proposed three challenges (subtasks): pun detection, pun location and pun interpretation (Miller et al., 2017).

In this work, we present a proposal for two subtasks: homographic pun location (subtask2), and homographic pun interpretation (subtask3).

Our proposal for both subtasks lies in the hypothesis that the two senses of the pun in the sentence are possible thanks to the coexistence of the pun with other words in that sentence that are semantically close to the pun. According to this hypothesis, our method for pun detection consists of finding pairs of words more semantically related in the sentence. In addition, our method for pun interpretation is based on the detection of words in the sentence, different from the pun, that help to find the two senses of the pun. The selection of these words is also based on the criterion of the semantic proximity to the pun.

2 Subtask 2: Pun location process

Pun localization consists of identifying which word is the pun given a sentence that contains a pun. Our proposal is based on two hypotheses: i) to find the most semantically related pair of words (one of these words should be the pun); ii) the pun should be at the end of the sentence.

Our approach to the pun location process is made following the Algorithm 1. As a previous step, the sentence is processed in order to eliminate punctuation marks and stop words, and to convert uppercase to lowercase. As a result of this process a set of semantically relevant tokens is obtained. This process removes from the sentences those tokens without semantics. Each token is represented by its embedding obtained from a pre-trained word embedding model (Mikolov et al., 2013) trained on part of Google News dataset (3 million words). The embedding dimension was fixed to 300.

For all the pairs of tokens, the cosine distance of their corresponding embedding representation

Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017), pages 440–443,
Vancouver, Canada, August 3 - 4, 2017.

is calculated. The pairs are ranked according to this distance and the pair of less distance is selected. Finally, the pun selection is performed applying two heuristics:

- First, we assume that consecutive words in a sentence are semantically close, but the words that help the pun to be interpretable are not placed next to the pun. Therefore, we do not consider those pairs that correspond to consecutive words in the sentence.
- Second, we assume that the words that help the pun to be interpretable are placed before the pun in the sentence. Therefore, we selected as pun the word in the pair that is situated closer to the end of the sentence.

Algorithm 1: Selection of the pun of a sentence, task2

Input: s, the sentence that contains a pun
Result: w_k, the word in s that we guess is the pun

```
begin
    k, b ← -1, ∞
    t ← remove_stopwords(tokenize(s))
    foreach w_i ∈ t do
        e_i ← embedding(w_i)
        foreach w_j ∈ t: i < j − 1 do
            e_j ← embedding(w_j)
            d ← cosine_distance(e_i, e_j)
            if d < b then
                b, k ← d, j
    if k > −1 then
        return w_k
    else
        return w_{|t|−1}
```

Table 1 shows the results for the subtask 2 (Homographic pun location). Although our results present a wide room for improvement (0.4462 for $F1$), they are in line with those obtained by other participants. We achieved the fourth place in the competition being the best result 0.6631 for $F1$.

In order to test the two heuristics applied in the pun selection, we additionally computed some statistics comparing our results with those of the gold standard.

We assumed that the words that help the pun to be interpretable are placed before the pun in the sentence in most of the cases. The number of pairs of tokens selected by our approach that contain the pun is 767. In 702 of these pairs (91,5%), the pun was the second component of the pair, and, only in 65 (8,5%) the pun was the first component. These percentages confirm the goodness of this heuristic for subtask 2.

$F1$	0.4462
recall	0.4462
precision	0.4462
coverage	1.0000

Table 1: Results of subtask 2: Homographic pun location.

We also assumed that the words that help the pun to be interpretable are not placed next to the pun; therefore, we did not consider as candidates the consecutive words. If this heuristic is not applied, the number of pairs of tokens selected by our approach that contains the pun is 672, fewer than 767 pairs in case the heuristics was applied. In these 672 pairs, there are 580 where the pun is the word selected by our approach, and in 92 pairs, the selected word was the first component of the pair, that is more than the 65 pairs in case the heuristics was applied.

3 Subtask 3: Pun interpretation process

The process of pun interpretation is described by Algorithm 2. The interpretation process of our proposal is made following several steps:

- Selection of the two words semantically closest to the pun.

  In a similar way that stated for subtask 2 (Section 2), the sentence is processed in order to eliminate punctuation marks and stop words, and uppercase are converted to lowercase.

  Given the set of tokens, a sorted list of pairs of different tokens is generated, where, the first component of the pair is the pun w_p and the second component is any of the other tokens in the sentence whenever is not consecutive to the pun. For each pair of tokens, the cosine distance of their corresponding embedding representation is calculated.

  We selected the two first pairs in the above sorted list, $(w_p, w_1), (w_p, w_2)$, that is, we selected the two words in the sentence most

Algorithm 2: Selection of the two synsets of the pun on a sentence, task3

Input: s, the sentence that contains a pun
w_p, the word in the sentence, at position p, that is the pun
Result: (sy_1, sy_2), the two synsets of the pun w_p in the sentence s

Function *get_closest_words (s, w_p)*
  $t \leftarrow$ remove_stopwords(tokenize(s))
  $w_1, w_2, b_1, b_2 \leftarrow$ null, null, ∞, ∞
  $e_p \leftarrow$ embedding_representation(w_p)
  foreach $w_i \in t \mid (i < p-1) \vee (i > p+1)$ **do**
    $e_i \leftarrow$ embedding_representation(w_i)
    $d \leftarrow$ cosine_distance(e_p, e_i)
    if *($d < b_1$) $\wedge$ ($d < b_2$)* **then**
      $b_1, b_2, w_1, w_2 \leftarrow d, b_1, w_i, w_1$
    else if $d < b_2$ **then**
      $b_2, w_2 \leftarrow d, w_i$
  return *(w_1, w_2)*

Function *get_context ($synset$)*
  $d \leftarrow$ get_definition($synset$)
  $w \leftarrow \{w_i \in$tokenize(lemmatize(d)) $\mid w_i \notin$ stopword_list$\}$
  foreach $e_i \in$ *get_examples($synsets$)* **do**
    $w \leftarrow w \cup \{w_i : w_i \in$tokenize(lemmatize($e$)) $\mid w_i \notin$ stopword_list$)\}$
  return w

Function *synset_similarity(sy_1, sy_2)*
  $c_1 \leftarrow$ get_context(sy_1)
  $c_2 \leftarrow$ get_context(sy_2)
  return $\|c_1 \cap c_2\|$

begin
  w_i, w_j = get_closest_words(s, w_p)
  $sy_1, b \leftarrow null, -\infty$
  foreach $sy_p \in$ *synsets(w_p)* **do**
    foreach $sy_i \in$ *synsets(w_i)* **do**
      $s \leftarrow$synset_similarity(sy_p, sy_i)
      if $s > b$ **then**
        $sy_1, b \leftarrow sy_p, s$
  $sy_2, b \leftarrow null, -\infty$
  foreach $sy_p \in$ *synsets(w_p)* **do**
    foreach $sy_j \in$ *synsets(w_j)* $\mid sy_j \neq sy_1$ **do**
      $s \leftarrow$synset_similarity(sy_p, sy_i)
      if $s > b$ **then**
        $sy_2, b \leftarrow sy_p, s$
  return *(sy_1, sy_2)*

closely related to the pun from a semantic point of view. The cosine distance of (w_p, w_1) is the smallest and the cosine distance of (w_p, w_2) is the next smaller one.

In Algorithm 2, this step corresponds to the *get_closest_words* function.

- Generation of a bag-of-words per synset.

  For each synset of the pun (w_p) and for each synset of both closest words (w_1, w_2), we obtain a bag-of-words that includes: i) all the lemmas in the gloss of the synset; ii) the own name of the synset; and iii) the lemmas in all the example sentences. Before getting the

lemmas, the sentences are processed in order to convert to lowercase and to eliminate punctuation marks and stop-words.

This step corresponds to the *get_context* function in Algorithm 2.

- Synsets selection.

  The final goal of the subtask is to select one pair of synsets (sy_1, sy_2) of the pun that represent its two different meanings in the sentence. Our hypothesis is that one synset of the pair (sy_1) is related to one synset of w_1 and the other synset of the pair (sy_2) is related to one synset of w_2. In a similar way that *Lesk algorithm* (Lesk, 1986), we used as measure of similarity between two synsets, the overlapping between the bags-of-words of both synsets.

  In this way, we select the first synset (sy_1) of the pun that maximizes the overlapping with one synsets of w_1. After that, we select other synset of the pun (sy_2, $sy_2 \neq sy_1$) that maximizes the overlapping with one synset of w_2.

Table 2 shows the results of subtask 3 (Homographic pun interpretation). Our results are low, but are in line with the results of the rest of the participants. We achieved 0.0996 for $F1$ (the third place), being the best result 0.1557.

$F1$	0.0996
recall	0.0978
precision	0.1014
coverage	0.9646

Table 2: Results of subtask 3: Homographic pun interpretation.

As in the subtask 2, we calculated some statistics comparing our results with those of the gold-standard. The number of correct pairs of synsets was 127 of the 1252 analyzed sentences, however, there were 255 additional sentences for which one synset was correct.

4 Conclusions

In this work, we have presented our participation at task 7 (subtask 2: homographic pun detection and subtask 3: homographic pun interpretation) of SemEval2017. Our approach is based on the use of word embeddings to find related words in a sentence and a version of the Lesk algorithm to establish relationships between synsets. We achieved the fourth place in subtask 2 (Homographic pun location) and the third place in subtask 3 (Homographic pun interpretation).

The results obtained are in line with those obtained by the other participants and they encourage us to continue working on this problem.

As future work we plan to adapt state-of-the-art WSD techniques to tackle with the pun interpretation problem.

Acknowledgements

This work has been partially funded by the Spanish MINECO and FEDER founds under project ASLP-MULAN: Audio, Speech and Language Processing for Multimedia Analytics, TIN2014-54288-C4-3-R.

References

Meri Giorgadze. 2014. Linguistic features of pun, its typology and classification. *European Scientific Journal* .

Michael Lesk. 1986. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation*. ACM, New York, NY, USA, SIGDOC '86, pages 24–26. https://doi.org/10.1145/318723.318728.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *CoRR* abs/1301.3781. http://arxiv.org/abs/1301.3781.

Tristan Miller and Iryna Gurevych. 2015. Automatic disambiguation of english puns. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2015)*. page 719729.

Tristan Miller, Christian F. Hempelmann, and Iryna Gurevych. 2017. SemEval-2017 Task 7: Detection and interpretation of English puns. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*.

Tristan Miller and Mladen Turkovi. 2016. Towards the automatic detection and identification of english puns. *European Journal of Humour Research* 4(1):59–75.