# ALC report: comparative study over the COSET shared task at IberEval 2017

Luca Ambrosini and Giancarlo Nicolò

Univesitat Politècnica De València

**Abstract.** This paper describes our participation in the *Classification Of Spanish Election Tweets (COSET)* task at IberEval 2017. During the searching process for the best classification system, we developed a comparative study over possible combination of corpus preprocessing, text representations and classification models. After an initial models exploration, we focus our attention over specific neural models. Interesting insight can be drawn from the comparative study helping future practitioner tackling tweets classification problems to create system baseline for their work.

## 1 Introduction

Nowadays the pervasive use of social network as a mean of communication helps researchers to found useful insight over open problems in the field of Natural Language Processing. In this context, the *Twitter* social network has a huge role in text classification problems, because thanks to its *API* is possible to retrieve specific formatted text (i.e. a sentence of maximum 140 characters called tweet) from a huge real-time text database, where different users publish their daily statements.
This huge availability of data gives raise to the investigation of new text classification problem, with special interest in prediction problems related to temporal event that can influence statements published by social network users. An example of this problem category is the text classification related to general election, where the *Classification Of Spanish Election Tweets (COSET)* task at IberEval 2017 is a concrete example.
In COSET, the aim is to classify a corpus of political tweets in five categories related to specific political topics. This task can be analysed as a domain-dependent (i.e. political domain) constrained-text (i.e. tweet sentence) classification problem.
To tackle the above problem we built a classification system that can be decomposed in three main modules, each representing specific approach widely used in the NLP literature: text pre-processing, text representation and classification model. During the modules design, we explore different design combinations leading the system development to a comparative study over the possible modules interactions. Analysing the produced study interesting insight can be drawn to create system baseline for tweet classification problem.
In the following sections we firstly describe the COSET task (section 2), then we report the development process of the classification system and its module design (section 3), after that, the evaluation of deployed systems over provided corpus is analysed (section 4), finally, conclusion over the whole work are outlined (section 5).

## 2 Task definition

The COSET shared task aim was to classify Spanish written tweets talking about the 2015 Spanish General Election, where each tweet had to be classified into one of five different categories: (i) political issues, related to the most abstract electoral confrontation; (ii) policy issues, about sectorial policies; (iii) personal issues, on the life and activities of

the candidates; (iv) campaign issues, related with the evolution of the campaign; (v) and other issues.
Participants had access to a labelled corpus composed of training set (2242 tweets) and development set (250 tweets) for system benchmarking, we analysed it and find the following statistical information that guided us in the baselines building:

- Average train sequence length:
  - 135 chars
  - 24 words
- Max train sequence length:
  - 140 chars
  - 49 words

## 3 Systems description

In this section we are going to describe the tweet classification systems built, from a module perspective we can describe our systems as composed of three main blocks: text pre-preprocessing (section 3.2), text representation (section 3.3) and classification model (section 3.4).

### 3.1 Initial investigation

To address the tweets classification problem we began our investigation analysing some of the most widely used text representations and classifiers. In the analysing for possible text representations we began focusing our attention on lexical features based: *Bag Of Words* [6],*Bag Of N-Grams* (bigrams and trigrams), both with and without *term frequency-inverse document frequency* normalization (i.e. TF-IDF norm). In relation to classification model that can exploit above representations, we analysed *Random Forest, Decision Trees, Support Vector Machines* and *Multi Layer Perceptron*, but since the results obtained with the combination of those *model + representation* were outperformed by neural network based models, we decided not to report their analysis in this paper, but rather focus the module description in relation to neural models.

### 3.2 Text pre-processing

Regarding the text pre-preprocessing, has to be mentioned that the corpus under observation can not be treated as proper written language, because computer-mediated communication (CMC) is highly informal, affecting diamesic[1] variation with creation of new items supposed to pertain lexicon and graphematic domains [7,8]. Therefore, in addition to well know pre-processing approach, as stemming (i.e. ST), stopwords (i.e. SW) and punctuation removal (i.e. PR), specific tweets pre-processing techniques has to be taken in consideration.
From previous consideration, we define a set of specific tweet pre-processing approach that take into consideration the following items: (i) mentions (i.e. MT), (ii) smiley (i.e. SM), (iii) emoji (i.e. EM), (iv) hashtags (i.e. HT), (v) numbers (i.e. NUM), (vi) URL (i.e. URL) (vii) and Tweeter reserve-word as RT and FAV (i.e. RW).
For each of these items we leave the possibility to be removed or substituted by constant string (e.g. (i) *Pre-processing of @Ambros and #atoppe :)* $\xrightarrow{substitution}$ *Pre-processing of $MENTION and $HASHTAG $SMILEY*, (ii) *Pre-processing of @Ambros and #atoppe :)* $\xrightarrow{removing}$ *Pre-processing of and* ).
To implement above pre-processing tecnique we take advantage of the following tools: (i) NLTK [3] and (ii) Preprocessor [4].

[1] The variation in a language across medium of communication (e.g. Spanish over the phone versus Spanish over email)

### 3.3 Text representation

The use of neural model suggest us to exploit recent trend over text representation, in particular we decided to use embedding vectors as representation following the approach described by [5], where tweet elements like *words* and *word n-grams* are represented as vectors of real number with fixed dimension $|v|$. In this way a whole sentence $s$, with length $|s|$ its number of word, is represented as a *sentence matrix* $M$ of dimension $|M| = |s| \times |v|$. $|M|$ has to be fixed a priori, therefore $|s|$ and $|v|$ have to be estimated. $|v|$ was fixed to 300 following [5]. $|s|$ was left as a system parameter that after optimization was fixed to $|s| = 30$, with this choice input sentences longer than $|s|$ are truncated, while shorter ones are padded with null vectors (i.e. a vector of all zeros). Depending of chosen tweets elements a different embedding function has to be estimated (i.e. learnt), in the continuation we are going to analyse the possible choices.

**Word embedding.** Choosing words as element to be mapped by the embedding function, raise some challenge over the function estimation related to data availability, in our case the available corpus is very small and estimated embeddings could lead to low performance. To solve this problem, we decided to used a pre-trained embeddings estimated over Wikipedia data base using a particular approach called *fastText* [5].
Using this approach, after the sentence matrix embedding is calculated, its weights can be *static* or *non-static*, in the latter case backward propagation will be able to adjust their values otherwise they will stay fixed as initially calculated by the embedding function.
In this way four possible combination of sentence matrix embedding can be formulated: (i) the use of a pre-trained embedding function (i.e. fastText from Wikipedia) and (ii) static or non-static weights. From this combination the one composed of static weight without pre-trained embeddings won't be take in consideration for obvious reason, while we decided to use two pre-trained function from Spanish (i.e. ES) and Catalan (i.e. CA) to see how the use of pre-trained embeddings of a similar language will perform in relation to static/non-static weights. Meaning that the case in consideration will be five: (i) ES static, (ii) CA static, (iii) ES non-static, (iv) CA non-static, (v) (no pre-trained embeddings) non-static.

**N-gram embedding.** Choosing n-gram as element to be mapped by the embedding function, raise more challenges respect simple words, because no pre-trained embeddings are available and in this case the corpus has to be significantly big, otherwise n-gram frequencies will be really low and the estimation algorithm is not able to learn a valid embedding. Our insight was empirically validated by very low performance, nevertheless, as explained in the following, this embedding will be used in a particular model that won't rely its performance just over n-gram embeddings.

### 3.4 Classification models

In the following, we describe the neural models used for the classification module, where for each of them the input layer uses text representations described in section 3.3 (i.e. sentence matrix).

**Fast text.** This model was introduced in [2], where its main difference from our neural model is the use of a particular input layer, in details, rather than use only words or only n-gram as element for the embedding, both elements are embedded with the aim of capture partial information about words order. The complete architecture is illustrated in figure 1, here the input layer is directly fed into a *Global Average Pooling* layer, that transforms the sentence matrix in a single vector, who is projected into two dense layers. Regarding the architectural references in [2], they used a number of hidden layers fixed to ten, but we measured better performance using just two layers, moreover we integrate both dropout, gaussian noise and batch normalization.

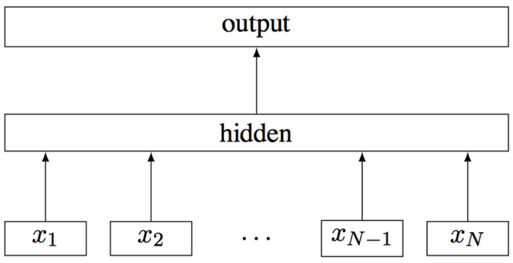

Fig. 1: [2] Model architecture of *fastText* for a sentence with $N$ n-gram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

**Convolutional Neural Network.** Convolutional Neural Networks (i.e. CNN) are considered state of the art in many text classification problem, therefore we decide to use them in a simple architecture composed by a convolutional layer, followed by a *Global Max Pooling* layer and two dense layers.

**KIM.** This model was introduced in [1], it can be seen as a particular CNN where the convolutional layer has multiple filter widths and feature maps. The complete architecture is illustrated in figure 2, here the input layer (i.e. sentence matrix) is processed in a convolutional layer of multiple filters with different width, each of these results are fed into *Max Pooling* layers and finally the concatenation of them (previously flatten to be dimensional coherent) is projected into a dense layer. The intuition behind this model is that smaller filter should be able to capture short sentence patterns similar to n-grams, while bigger ones should capture sentence level features. Regarding the architectural references in [1], the number filter $|f|$ and their size was optimized leading to the following results: $|f| = 4, f_1 = 2 \times 2, f_2 = 3 \times 3, f_3 = 5 \times 5, f_4 = 7 \times 7$.

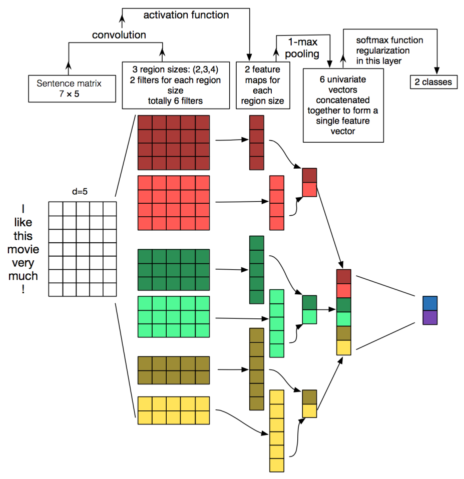
activation function
convolution
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

Fig. 2: [9] Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification

**Long short-term memory.** LSTM is a type of Recurrent Neural Network that is relatively insensitive to gap length, thanks to this behaviour, they are considered state of the art in some NLP problems (e.g. machine translation). Our architecture was made of an embedded input layer followed by an LSTM layer of 128 units, terminated by a dense layer, moreover to avoid overfitting we used dropout and recurrent dropout.

**Bidirectional LSTM.** Similar to previous model, bidirectional LSTM is a variation of LSTM where the two RNN receive different inputs, the original and its reverse order, and their results are connected through the recurrent layers. Our architecture follow the previous one with LSTM layer of 128 units terminating with a dense layer, where all the layer used dropout and recurrent dropout.

## 4 Evaluation

In this section we are going to illustrate results from the comparative study elaborated during the system development, first we illustrate the metric used to evaluate the system (section 4.1) and then we report results produced by a 10-fold cross validation over the given data set (section 4.2).

### 4.1 Metrics

System evaluation metric were given by organizers and reported here in the following equations (1) to (4), their choice was to use an $F_{1-macro}$ measure due to class unbalance in the corpus.

$$F_{1-macro} = \frac{1}{|L|}\sum_{l\in L} F_1(y_l, \hat{y}_l) \tag{1}$$

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2}$$

$$precision = \frac{1}{|L|}\sum_{l\in L} Pr(y_l, \hat{y}_l) \tag{3}$$

$$recall = \frac{1}{|L|}\sum_{l\in L} R(y_l, \hat{y}_l) \tag{4}$$

### 4.2 Results

In the continuation we present a comparative study over possible combination of pre-processing (table 1) and text representation (table 2), in both case results are calculated from averaging three runs of a 10-fold cross validation over the complete data set. Notations used in table 1 refers to the one introduced in section 3.2, where the listing of a notation means its use for the reported result, to simplify notation SW and PR were merged as SR. Regarding the tweet specific pre-processing all the items have been substituted, exception for the URL and RW that have been removed (in the table reported as RM). From all the possible combinations (i.e. $2^3 \cdot 3^7 + 1$) we report only the most relevant ones.
From the analysis of table 1 no absolute conclusion can drawn, meaning that it wasn't possible to find a combination of pre-processing that gives the best performance for all the model, meaning that each model is highly sensible to the performed combination. Nevertheless, some relative observation can be made:

– The RM preprocessing (i.e. removing of the URL and RW) leads to performance improvement to all the model respect to no pre-processing at all,
– Kim model is the only one having a significant decrease in the performance when stemming is applied alone or with other preprocessing.

Analysing results in table 2, here the used notation refers to the one introduced in section 3.3, where the listing of a notation means its use as embedded input layer for the reported result. From its analysis following interpretation can be drawn:

Table 1: Pre-processing study comparing 10-fold cross validation results over the development set in terms of percentuage of $F_{1-macro}$ score.

| Preprocessing | Models | | | | |
|---|---|---|---|---|---|
| | CNN | LSTM | B-LSTM | FAST-TEXT | KIM |
| Nothing | 50,5 | 55,6 | 51,1 | 53,0 | **55,8** |
| ST | 49,6 | 49,9 | 47,5 | **53,1** | **53,1** |
| ST+SR | 47,6 | **55,3** | 47,6 | 52,9 | 51,1 |
| ST+SR+RM | 51,9 | **56,8** | 52,2 | 56,0 | 50,8 |
| ST+SR+RM+MT | 54,6 | **56,1** | 47,7 | 54,6 | 53,6 |
| ST+SR+RM+MT+NUM | 53,2 | **55,5** | 51,0 | 53,4 | 51,5 |
| ST+SR+RM+MT+NUM+EM+SM | 52,7 | **56,4** | 53,7 | 54,2 | 51,8 |
| ST+SR+RM+MT+NUM+EM+SM+HT | 55,1 | 54,0 | 52,9 | **56,7** | 51,1 |
| SR+RM+MT+NUM+EM+SM | 54,5 | 54,8 | 53,9 | **57,0** | 54,8 |
| RM | 55,9 | 43,4 | 48,5 | 56,5 | **57,7** |
| RM+EM+SM | 57,1 | 52,1 | 49,9 | 54,8 | **58,9** |
| RM+MT+NUM+EM+SM | 54,3 | 54,6 | 55,5 | **56,8** | 54,8 |

- Setting as *static* the sentence matrix weights has the worst performance (independently of the used language)
- As opposite to the previous point, the set to *non-static* lead to better performance, where this insight can be deduced by corpus characteristic (i.e. a good example of Computer Mediated Communication)
- The use of pre-trained embedding is useful in combination with *non-static* weights (i.e. best performances with ES non-static)
- Even if is not available a pre-trained embedding for the task language the use of a similar language with non-static weight (i.e. CA non-static) can increase the performance respect only non-static, this can be interpreted as a case of transfer learning.

Table 2: Text representation study comparing 10-fold cross validation results over the development set in terms of percentuage of $F_{1-macro}$ score. The pre-processing setting was fixed at RM+EM.

| Embedding | Models | | | | |
|---|---|---|---|---|---|
| | CNN | LSTM | B-LSTM | FAST-TEXT | KIM |
| ES static | **48,1** | 36,1 | 38,9 | 36,4 | 43,6 |
| CA static | **45,1** | 30,6 | 38,5 | 30,1 | 39,6 |
| ES non-static | 57,1 | 52,1 | 49,9 | 54,8 | **58,9** |
| CA non-static | 53,5 | 53,6 | 46,3 | 54,1 | **56,2** |
| non-static | 52,6 | 47,3 | 51,8 | 53,3 | **54,7** |

## 5 Conclusions

In this paper we have presented our participation in the IberEval2017 Classification Of Spanish Election Tweets (COSET) shared task. Five different neural models were explored, in combination with 11 types of preprocessing.
No preprocessing emerged to be the best with every kind of model, indicating that the preprocessing pipeline optimization has a big impact on results. **NO BRO fermo un**

**secondo, qui c' qualcosa che non va, non capisco il senso se no prepro  il best, volevi dire il worst?**
preprocessing strongly influence the performance, no general rule (one pre rule them all, to note that some specific pre could have a huge gap between them
We also explored static vs non-static word embeddings and non-static vectors initialized with pre-trained vectors on a bigger corpus is the best performing combination.
In this task word order is not important Per il TOP no one pre-prepr Every model need a different preprocessing (global vs local) -¿ ST+SW wors performance -¿ CL URL+RW improve Using pre-trained word vectors is useful Learn vectors during training is usefull (non-static)

## References

1. Kim, Yoon. "Convolutional neural networks for sentence classification." arXiv preprint arXiv:1408.5882 (2014).
2. Joulin, Armand, et al. "Bag of tricks for efficient text classification." arXiv preprint arXiv:1607.01759 (2016).
3. Edward Loper and Steven Bird. 2002. NLTK: the Natural Language Toolkit. In Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1 (ETMTNLP '02), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 63-70. DOI=http://dx.doi.org/10.3115/1118108.1118117
4. Preprocessor is a preprocessing library for tweet data written in Python, https://github.com/s/preprocessor
5. Bojanowski, Piotr and Grave, Edouard and Joulin, Armand and Mikolov, Tomas. "Enriching Word Vectors with Subword Information" arXiv preprint arXiv:1607.04606 (2016).
6. Harris, Zellig S. "Distributional structure." Word 10.2-3 (1954): 146-162.
7. Bazzanella, Carla. "Oscillazioni di informalit e formalit: scritto, parlato e rete." Formale e informale. La variazione di registro nella comunicazione elettronica. Roma: Carocci (2011): 68-83.
8. Cerruti, Massimo, and Cristina Onesti. "Netspeak: a language variety? Some remarks from an Italian sociolinguistic perspective." Languages go web: Standard and non-standard languages on the Internet (2013): 23-39.
9. Zhang, Ye, and Byron Wallace. "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification." arXiv preprint arXiv:1510.03820 (2015).