ALC

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 (put the cite of other work that use social network). 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 (find some cite from google scholar), 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 development test 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 training set (2242 tweets) and development set (250 tweets) of labelled data to benchmark their system, 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,**?**]. 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

As a tweet representation used as first layer
The use of neural model suggest us to exploit recent trend over text representation, in particular we decided to use embedding vector 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 matrix of dimension $|s| \times |v|$.
We left the sentence length as a parameter and the best results were obtained with length=30, that's reasonable since the average tweet length in words in 24. If the sentence dimension is bigger than the selected window we used the first length words, if the sentence is smaller than the windwos we used 0-right padding.
@LUCA: sorry bro for me it's not so clear XD if i got it: if bigger than windows, just cut; if smaller padding it from the last word till the 30th (aka 0-right padding... is it the formal terminology?)
Transform a list of num_samples sequences (lists of scalars) into a 2D Numpy array of shape (num_samples, num_timesteps). num_timesteps is either the maxlen argument if provided, or the length of the longest sequence otherwise. Sequences that are shorter than num_timesteps are padded with value at the end. Sequences longer than num_timesteps are truncated so that it fits the desired length. Position where padding or truncation happens is determined by padding or truncating, respectively.
@LUCA: Do you know any TOP/primary reference over word embeddings to cite it properly and then refrase the paper to explain it formally?

Words embedding Since the number of tweets is considered small to learn vector representation of words, we tried to initialize the embedding matrix with pretrained word vectors. In particular we used vectors trained on wikipedia using fastText [5]. **@LUCA: "We tried static pretrained vectors, learning them during training starting from a random matrix or from the pretrained embeddings." what do you mean? the non-static case? or that out of vocabolary are randomly initialized? Sorry bro I don't get it.**
matrix initialization (if not inside fast text the vector is all zero) static (3) vs non-static (1, 2)

N-gram embedding We also tried to learn an embedding for n-grams (bigrams and trigrams), but, since the corpus is small, n-grams frequencies are very low and the algorithm is not able to learn a valid embedding. Also there are no pre-trained n-grams embeddings available. Bigrams brought a small improvement, bigger n-grams result in performance decrease.
Bigrams brought a small improvement, in which case? IMHO Here we should just explain it and then in the evaluation explain the use or not. because otherwise doesn't make so much sense to talk about it here and also saying that has poor results. This same comment is related to the B-LSTM comment in the following. Feedback?
scriuvi che questo viene utilizzato solo da fast-text nella parte di modello

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. matrix sentence representation).

Fast text This model is based on this paper [2]. Differently to the other models where just words are used, here Bag of n-grams are added to the input as additional features to capture some partial information about the local word order. It this model the **embedding output (should it be input layer, that in our case is a word embedding vector?)** is directly fed into a *Global Average Pooling* layer, that transforms the whole sentence in a single vector computed as the average of the word vectors. This vector is then projected into 2 fully connected layers. In the paper the number of hidden layers is fixed to 10, but we measured better performance using just 2 layers. We explored both dropout and gaussian noise + batch normalization and best results were given by the latter, introducing a 0.2 noise after the embedding layer and after the first dense layer.

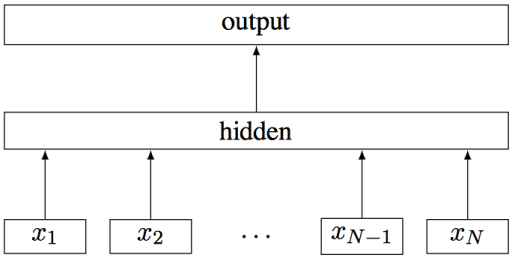

Fig. 1: [2] Model architecture of *fastText* for a sentence with N n-gram features $x_1, \ldots, x_N$. The features are embedded and averaged to form the hidden variable.

Convolutional Neural Network Convolutional Neural Networks are considered state of the art in many text classification problem. This model is composed by a convolutional layer, followed by a global max pooling layer and two fully connected layers.

KIM. Cita direttamente questo paper e scaricalo add it to refs **Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners Guide to) Convolutional Neural Networks for Sentence Classification. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/**

This model is based on this paper [1]. It is based on different filter region size convolutions, followed by maxpooling layers. All those outputs are concatenated to build sentence representation that is finally projected into a dense layer for the classification. The intuition behind this model is that smaller filter region size should be able to capture short sentence patterns (similar to ngrams), while bigger sizes should capture sentence level features. We reached the best performance using [2, 3, 5, 7] as filter sizes.

Long short-term memory LSTM is a type of Recurrent Neural Network (i.e. RNN) that is relatively insensitive to gap length, compared to others RNN, they are considered state of the art in machine translation. In this model the classical embedding layer is followed by an LSTM layer with 128 units, terminated by a fully connected layer. We also tried Bidirectional LSTM, To avoid overfitting we used droput and recurrent dropout, best results were obtained with a droput value of 0.4 and recurrent dropout value of 0.3.

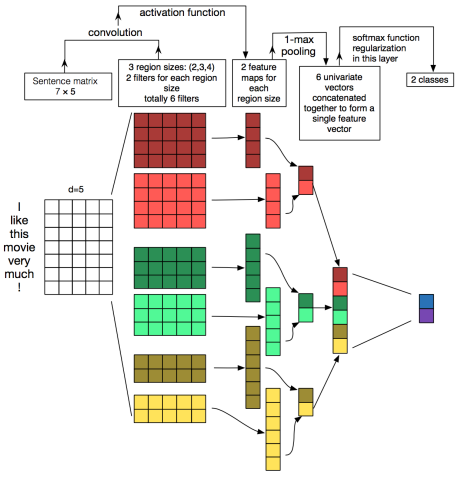

Fig. 2: Illustration of a Convolutional Neural Network (CNN) architecture for sentence classification

4 Evaluation

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

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

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (2)$$

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

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

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 development set (**LUCA put in your word the explanation of training/development set evaluation**). Notations used in table 1 refers to the one introduced in section 3.2 (the listing of a notation means its use for the reported result, where 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.
conclusion over pre-processing

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

– CL serve a tutti (URL e RW rumore)
– recurrent model perform better with ST+SW, while other peggiorano
– fast text just cl improve performance a lot

Intro su text representation

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
ES non-static	57,1	52,1	49,9	54,8	**58,9**
CA static	**45,1**	30,6	38,5	30,1	39,6
CA non-static	53,5	53,6	46,3	54,1	**56,2**
Online	52,6	47,3	51,8	53,3	**54,7**

conclusion over text representation, introduction over the use of catal pre-trained vector

– static always worse, CMC learnt over wikipedia
– no-static always improve
– if no-static + pretrained improve respect just no-static, even if used a pre-trained of a similar language
– best result when non-static es

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.

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.
   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.