Multimedia Tools and Applications (2022) 81:27111–27136
https://doi.org/10.1007/s11042-021-11880-2

1183: MULTIMEDIA PROCESSING TO TACKLE THE DARK SIDE OF SOCIAL LIFE

Application-specific word embeddings for hate and offensive language detection

Claver P. Soto[1,2] · Gustavo M. S. Nunes[2] · José Gabriel R. C. Gomes[2] · Nadia Nedjah[3]

Received: 12 November 2020 / Revised: 18 September 2021 / Accepted: 23 December 2021 / Published online: 25 January 2022

Abstract

For the task of hate speech and offensive language detection, this paper explores the potential advantages of using small datasets to develop efficient word embeddings used in models for deep learning. We investigate the impact of feature vectors generated by four selected word embedding techniques (*word2vec*, *wang2vec*, *fastText*, and *GloVe*) applied to text datasets with size in the order of a billion tokens. After training the classifiers using pre-trained word embeddings, we compare the classification performance with the results from using feature vectors generated from small datasets with size in the order of thousands of tokens. Using numerical examples, we show that the word embeddings with the smallest size yield slightly worse accuracy values but, in combination with smaller training times, such embeddings lead to non-dominated solutions. That fact has an immediate application in significantly reducing training time at a small penalty in classification accuracy. We explore two ways to rank the studied alternatives based on performance factors and on PROMETHEE-II scores. According to both rankings, *GloVe* is the best method for NILC-embedding, and *fastText* is the best method for dataset-specific embedding. It is expected that specific word embedding should yield a better fit to a particular dataset, which should yield shorter training and better accuracy. However, the obtained results indicate that NILC-embeddings would lead to an equally good fit.

✉ Claver P. Soto
claver@pads.ufrj.br

Gustavo M. S. Nunes
gustavo_mn@poli.ufrj.br

José Gabriel R. C. Gomes
gabriel@pads.ufrj.br

Nadia Nedjah
nadia@eng.uerj.br

[1] Department of Computation, Federal Rural University of Rio de Janeiro, Rio de Janeiro, Brazil

[2] Electrical Engineering Program, Federal University of Rio de Janeiro, Rio de Janeiro, Brazil

[3] Department of Electronics Engineering and Telecommunications, Engineering Faculty, State University of Rio de Janeiro, Rio de Janeiro, Brazil

Keywords Word embeddings · Convolutional neural network · Hate and offensive language detection · Multi-criteria analysis

1 Introduction

With the flourishing of social media, more and more people have moved online. Hate speech and offensive tweets have increased worldwide.Hate speech is generally understood as public speech exerting hate towards a group of people cursing their race, religion, sex, or sexual orientation. On the other hand, offensive language is usually defined as public speech that upsets or embarrasses a person because it is rude or insulting. So, individuals or group of people inclined toward racism, misogyny, or homophobia have found the perfect niches to reinforce their views, which often lead to violence. Any group of people working together, such as in societies, companies, or enterprises must confront the trend without compromising free speech and censorship on widely used technological platforms. Therefore, every company should have a hate speech and offensive language detector embedded into its informational system.

In 2018, journalist Fernanda Pugliero [29] published a study about the situation of hate and offensive contents on the Brazilian Internet. Since 2007, nearly 4 million hate crime reports have been received by the National Cyber Crime Reporting Center. Therein, it was also pointed out that the annual average of complaints is decreasing because there are fewer people willing to report a hate crime. Apparently, the feeling of hatred is being trivialized, becoming natural in social networks where users replicate the speeches, often without posting a critical position. In 2021, a broad sociological study was published, highlighting the importance that academia should give to the issue of hate speech and social media [12].

Social media platforms prohibit hateful, offensive and/or harassing comments. However, applying these prohibition rules, and identifying the hate and/or offensive comments on a large scale, is still an open problem. This was confirmed in March 2019, in the SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media competition (OffensEval), as documented in [37].

In the state-of-the-art literature, the aforementioned problem is usually viewed as a supervised classification task, where machine-learning-based systems are trained using a set of data containing posted comments, accompanied by the respective labels that describe the nature of the comment as either offensive or not. The classifiers usually take advantage of Machine Learning (ML) models, such as Support Vector Machine (SVM) [8], Convolutional Neural Network (CNN) [16], Recurrent Neural Network (RNN) [32] and Bidirectional Long-Short Term Memory (BiLSTM) [35], among others. Another critical aspect of the problem regards the language modeling, where Natural Language Processing (NLP) models are used to extract, from the available texts, the syntactic and semantic information necessary to build numerical representations of the texts. Such a representation is primordial for learning systems to present the input patterns to the classifier that is used. It has a huge impact on the effectiveness and efficiency of the system, and as such must be planned or selected carefully to guarantee the proper usage of the proposed learning system.

In this work, we carry on existing research works in the field of sentiment analysis that exploit real data available. In particular, we consider the hate speech and offensive language detection task. Our work is inspired by [17], in which the authors considered four language models for representing words in text as numeric vectors, namely *word2vec* [23], *wang2vec* [22], *fastText* [4], and *GloVe* [25]. Depending on the adopted model, the vector space considers different aspects and word interactions within text, such as syntax analysis and word analogies. In [17], the models are compared in the context of semantic similarity and post-tagging tasks. These models are previously trained in a large multi-genre Portuguese *corpus*, and they are tested in specific text datasets that are related to the two considered tasks. The authors encouraged researchers to extend the performance comparison of these models to different applications, so that a better knowledge of the suitability of each representation for a variety of NLP tasks is gained. In this present work, we are interested in determining whether a universally trained word embedding is necessary to reach good levels of classification accuracy or, instead, an application-specific embedding would be enough to achieve such accuracy levels. Application-specific word representations may require less time to train than universal ones, because application-specific corpora are usually smaller than generic, multi-purpose corpora. Although we perform this investigation on datasets oriented to detect hate speech and/or offensive expressions, we expect the conclusions of this work to be valid for any other NLP task.

In a previous work [24], we compared the performance of *word2vec* and *wang2vec* representations for a detection task of hate speech and/or offensive tweets. In the present work, we extend this comparison by including the *fastText* and *GloVe* representations, and by presenting a processing time analysis regarding the training and inference times demanded by each representation. Without loss of generality, we apply our findings to texts written in the Portuguese language. It is safe to assume that the conclusions of the study are not language-specific and thus, can be generalized to any language. In general, this study allows for the impact evaluation of several internal representations of the input patterns. We compare CNN's performance as a classifier regarding two situations. In the first one the classifier receives word embeddings trained in a general domain. In the second one it receives word embeddings trained in the application domain exploiting the same dataset used for classification.

More recently, embeddings such as USE [6], BERT [10], InferSent [7], and ELMo [26] were introduced. Unlike the word-embedding models considered in this study, these are sentence-embedding models. Instead of codifying separate words, sentence-embedding models take the context of the words into account during the inference of a sentence encoding. In doing so, such embeddings are able to clear out the ambiguity between the meanings of words in different contexts. Of course, this comes at a price. It increases immensely the underlying computational cost. In this study, we limit the evaluation to word embeddings. Sentence-embedding models might be developed in future studies, as computational resources become more accessible.

The main contributions of this work are two-fold: *(i)* To the best of our knowledge, there are no other works that compare, for the Portuguese language, how word representations obtained from word embedding models trained in different types of corpora (*i.e.* generic texts *versus* application-specific ones) impact on the classification performance for hate/offensive speech detection. Moreover, only few datasets are available for hate-speech detection in Portuguese. These datasets do not contain a very large *corpus*, especially when compared with the number of samples from datasets designed for other types of applications. Given this limitation, we investigate which strategy (*i.e.* data quantity or data quality) leads to the best classification performance for the task at hand. The first strategy is to obtain the word representations used to train the classifier from publicly available word embedding models that are pre-trained in very large, but generic corpora. The second one is to obtain such representations from word embedding models that are trained in smaller, but application-specific corpora. We expect that the results from our work will help researchers interested in this task to assess the strengths and weaknesses of each strategy; *(ii)* We propose a systematic way of comparing the classification performance under multiple criteria (*i.e.* classifier accuracy and training time[1]) simultaneously. Most preferable solutions are the ones that provide the best trade-offs between accuracy and training time, rather than the ones that emphasize one criterion in detriment of the other. To do so, we perform a dominance analysis, which takes into account both the classifier accuracy and the training time, which are obtained from each word embedding configuration that is considered. We also apply two ranking metrics. One metric ranks the configurations based on their respective ratio between accuracy and normalized training times. The other metric uses scores from the PROMETHEE-II method [5], which is specifically designed for multi-criteria problems, in order to rank the configurations.

The aforementioned contributions can be summarized in two main points, which are presented as follows:

1. a novel analysis/methodology to decide which training data (application-specific versus generic) lead to the best word representations for hate and/or offensive speech detection in Portuguese language;
2. a systematic comparison framework, using dominance analysis, for assessing classifier performance as a multi-criteria problem.

Rather than generic NLP tasks, the analyses we introduce are limited to hate/offensive speech detection. The present work does not take into account advanced models, such as BERT or ELMo, which would require dedicated hardware. Different training strategies might be selected as the best ones, should one consider other embedding models. Enhancements of the present study could include extending the current analyses for other NLP tasks, and exploring word embedding sizes to determine the most adequate word representation. Other improvement possibilities are considered in Section 6.

The remainder of this paper is organized into six sections. First, in Section 2, we present and comment on recent related works in the literature. Then, in Section 3, we present the proposed methodology, and give a brief explanation of the exploited techniques. After that, in Section 4, we describe the datasets used in the performed experiments regarding classification as well as language representation. Later on, in Section 5, we present, analyze and discuss the obtained performance results. Furthermore, we compare the obtained results to those from related works. Last but not least, in Section 6, we draw some conclusions and point out some promising directions for future work.

2 Related works

Since the 90's, there have been, in the digital world regarding the English language, many works carried out to automatically identify hostile messages. For example, in [34], a system for recognizing hostile messages contained in e-mails is described. In their work, messages are encoded based on the syntax and semantics of each sentence, and then classified using decision trees.

Research works regarding sentiment analysis involve many areas of knowledge, such as natural language processing, computational intelligence and/or computational linguistics.

[1]Training time involves CNN training, exclusively, and to stress that, we often refer to it as *CNN training time* throughout the manuscript. The word-embedding training is assumed to be performed *a priori* and only once: we either use pre-trained word embeddings, or train our own models. In either case, the training time of the word-embedding model is not taken into account in this study.

The development of this research area indicates that both language models and classification models perform better when using computational intelligence and machine learning techniques. This has been proven by the many existing works that explored these research trends.

In [3], the authors propose the training of a neural network for learning the joint probability function of the word sequence in two datasets for the English language. They achieve better results than those at the state of the art, using n-grams language models in terms of intrinsic metrics regarding language modeling.

In [23], the authors use a simple Neural Net Language Model (NNLM) compared to the neural language model proposed by [3]. The proposed neural network has no hidden layers. It specializes in representing words by vectors of continuous values obtained from very large datasets. The authors performed tests with metrics of word syntactics and semantic similarity. Their method is known as *word2vec*, and has two models: Continuous Bag-Of-Words model (CBOW), which predicts the target words, given the contexts; and the Skip-gram model, which predicts contexts, given the target words.

In [19], the author performs several experiments improving the state-of-the-art results for some tasks, including analysis of sentiments. The author uses *word2vec* representations obtained from pre-trained data of about 100 billion words, as provided by Google^{TM}. For the classification, a CNN architecture that is chosen through a fine mesh search is proposed. For sentiment analysis, the author uses a dataset of positive and negative reviews of films, and another dataset regarding reviews that are further pre-classified as very positive, positive, neutral, negative and very negative. Based on the obtained results, the author deduces that the pre-trained vectors contain the so-called universal features that are claimed as good enough to be used in the data classification.

In [22], the authors present modifications to *word2vec* in order to generate word embeddings[2] more suitable for syntax tasks. The authors called the proposed model as structured *word2vec*, also known as *wang2vec*, referring to the first author's name. Their method also explores the CBOW and Skip-gram models. The authors report that training took around 24 hours, using a dataset that contained a number of tokens[3] in the order of a billion.

In [38], the authors present a practical work, analyzing the CNN architecture proposed in [19]. They use the same datasets to analyze sentiments, and compare the results obtained when the language is modeled using *word2vec* to those obtained when using Global Vectors (*GloVe*) for word representation [25].

Furthermore, there are many research works regarding languages other than English. For instance, in [27], the authors show sentiment analysis results for the Italian language. They use the word embeddings *word2vec* and *fastText* [4] obtained from different datasets that are either designed for sentiment analysis or generic applications (*i.e.* applications not related to sentiment analysis). The generic datasets contain a number of tokens in the order of millions, whereas sentiment analysis datasets contain a smaller number of tokens. Based on the obtained results, it was noted that *word2vec*, when applied to the domain-specific datasets (*i.e.* sentiment analysis datasets, in this case), achieves better results than those achieved by applying *word2vec* to generic (and also larger) datasets.

In [30], the authors create the first public repository of word embeddings for the Portuguese language, obtained from Brazil and Portugal. They are trained with 1.7 billion

[2]The term "word embeddings" refers to the representation of words as vectors containing real numbers. These representations should include some knowledge of positional information among words.

[3]Token are atomic units of data used for text analysis. In general, any string delimited by spaces or punctuation marks is considered as a token.

tokens using only *word2vec*. The authors point out that the training of 31 configurations of the model lasted a week and a half, with an average of 8 hours for each.

In [17], the authors continue the work reported in [30], generalizing the application to other datasets. They evaluate *fastText*, *GloVe*, *wang2vec* and *word2vec*, trained using around 1.3 billion tokens, including Brazilian and European Portuguese. This number is lower than that handled in [30] because pre-processing is used to reduce the vocabulary size. With the obtained word embeddings, they performed tests in some NLP applications. The authors encourage other researchers to consider using these pre-trained word embeddings in other applications, making them available in Inter-institutional Nucleus for Computational Linguistics (NILC) repository. In the present work, these four word embeddings are exploited, and they are referred to as NILC-embeddings.

In [9], the authors develop two datasets of sentences, obtained from Brazilian news sites, labeled either as offensive or non-offensive. For the codification of sentence characteristics, they use mixtures of n-grams. Later on, in [21], the authors carried on this research by adding, to the codification of characteristics, meta-attributes that were generated based on the neighborhood of each sentence within the text. Both works use SVM for the sentence classification.

In [14], the authors provide a detailed study of the problem of hate speech automatic detection and describe most of the resources available. In [13, 15], the authors analyze the complexity of dataset annotation for its use in hate speech detection, and they propose a hierarchical annotation scheme that facilitates the identification of different types of hate speech and intersections. The authors present a dataset for the Portuguese language composed of 5,668 comments labeled to analyze hate speech. In [13], the authors use the n-gram model as a language model, and the classification is performed using several models, such as MLP and SVM. The best results were obtained with SVM, with an accuracy of 0.783 and $F1S$ of 0.764. In [15], the authors used pretrained *GloVe* word embeddings of dimension 300 available in [17] and, for classification, they used a LSTM-based model obtaining an $F1S$ of 0.72. In the present work, as seen in Section 4.1, the dataset proposed by these authors is used under the name HdsPt.

In [33], the authors perform the classification of hate sentiments using a CNN model. They use the datasets made available in [9] and [13] with the NILC-embeddings [17], *wang2vec* and *GloVe*. They achieve better results when using *wang2vec*.

In [31], the authors compare classical machine learning methods and some architecture configurations using CNN and LSTM with word-embeddings *Glove*, for the detection of hate speech in the English language. The authors use CNN with a single filter group and some kernel sizes with one and two layers. As the dataset used is unbalanced, the authors propose some modifications to work around this problem.

In [20], the authors present a new dataset for Brazilian Portuguese with 21,000 social media comments manually annotated in seven toxicity categories. Also, they elaborated a binary case of toxic and non-toxic comments from these annotations. For the case of binary classification, the authors compared the performance of the Bag-of-Word (BoW) model together with machine learning techniques and some versions of the BERT model, noting that the monolingual approach of the BERT model presents a better performance. For the case of multi-level classification, the authors pointed out the challenge of dealing with the unbalanced dataset.

In [28], the authors review issus related to the corpus that is used to detect hate speech. They analyze resources, development methodologies, coverage of various languages, and so forth. They review several recent works that dead with problems in NLP and abusive language detection.

In [36], the authors propose to incorporate a lexicon of 1,000 offensive expressions annotated with binary classes: context-dependent and context-independent offensive. This lexicon was extracted from a dataset with 7,000 comments for the Portuguese language built by the authors. These comments were annotated in three different layers with 2, 3 and 10 categories respectively. For comparison in classification, the authors used language models based on mixtures of the BoW model, the lexicon, and context-dependence markers proposed by themselves. As classification models the authors used NB (Baïve Bayes), SVM, MLP and LSTM. They also used the BERT model and *fastText*+LSTM. The best results were obtained using the model proposed by the authors and *fastText*. With the dataset of [15], the authors reported an $F1S$ of 0.86 while with the dataset of [9] an $F1S$ of 0.88. The works in [20, 36] illustrate the growing interest in the hate and offensive language detection topic for the Portuguese language.

In the present work, we carry on the research work reported in [9, 13, 17, 21, 24, 33]. We maintain the *wang2vec* and *GloVe* word embeddings, and include two more word embeddings for comparison: *word2vec* and *fastText*. The contribution of this work is to analyze the performance of a CNN-based classifier for the hate speech detection in text written in the Portuguese language, when trained with different vector representations of the same input texts. Each representation is obtained from a different word embedding that is either pre-trained in datasets from generic applications (NILC-embeddings) or trained in datasets from the application domain. Performance is evaluated in terms of classification accuracy and training speed of the CNN model.

3 Proposed methodology and exploited techniques

The sentiment analysis task uses techniques and models that can be organized into three groups: text pre-processing, language models that represent the language of the target application, and classification models for sentiment analysis.

3.1 Text pre-processing

The sentences in the datasets considered in this work are divided into isolated tokens. Following the recommendations from [17], the tokens are normalized, in order to reduce the vocabulary size. The token normalization procedure consists in: (i) replacing numeric characters with zeros; (ii) representing Uniform Resource Locator (URL) strings with the token defined as “url”; and (iii) representing e-mail addresses by the token defined as “email”. We also remove punctuation marks, and all words are rewritten using lowercase letters. In each dataset, we consider that all sentences have the same length as the longest sentence in the corresponding dataset. Sentences that are smaller than the longest sentence in each dataset are filled with the required number of zeros, in order to achieve the length of the corresponding longest sentence.

3.2 Language models and word embeddings

In this work, we compare four language models: (*i*) the Skip-gram model of *word2vec*; (*ii*) the structured Skip-gram model of *wang2vec*; (*iii*) the Skip-gram model of *fastText*; and (*iv*) the *GloVe* model. The Skip-gram model of *word2vec* is widely used to represent the language in sentiment analysis applications. The structured Skip-gram model of *wang2vec* was originally proposed for applications involving syntax analysis. This model also achieves good results for applications that consider semantics in sentiment analysis, as reported in [33]. The *fastText* model was proposed for tasks related to text classification. According to [18], in sentiment analysis and tag prediction tasks, *fastText* achieves accuracy performance comparable to other word embeddings, and trains significantly faster than the considered word embeddings. The *GloVe* model takes into account some form of meaning between words from a *corpus*, when encoding words into vectors. *GloVe* is especially used for word similarity tasks. For simplicity, in the following sections, we refer to the Skip-gram model of *word2vec* as *word2vec*, and the structured Skip-gram model of *wang2vec* as *wang2vec*, and the Skip-gram model of *fastText* as *fastText*.

3.2.1 word2vec embedding

In [23], the authors present the CBOW and Skip-gram models. The Skip-gram model is commonly chosen to create word embeddings because it emphasizes the relationships between an arbitrary word and its context (*i.e.* neighboring words). The Skip-gram model learns the embeddings by iterating several times through a set of word embeddings. At each iteration, the model adjusts the embedding of each word w, in order to make this embedding more similar to embeddings of words that are close to the word w (*i.e.* words that are located within a defined distance around the word w). At the same time, the model makes the embedding of word w less similar to the embeddings of words whose distance to word w is above a pre-established threshold distance.

3.2.2 wang2vec embedding

In [22], the authors present the “structured Skip-gram model” as an adaptation of *word2vec*. They argue that *word2vec* is insensitive to the order in which the words occur in text, which is improved in *wang2vec*. Because of this sensitivity, *wang2vec* is useful for learning semantic representations, and sub-optimal for tasks that involve syntax analysis. The word ordering in text is considered during the structured Skip-gram model training.

3.2.3 GloVe embedding

In [25], the authors propose the “Global Vectors” (*GloVe*) word embedding that is based on a log-bilinear model. Authors argue that ratios between co-occurring word probabilities capture better a crude relationship of meaning between words than the individual co-occurring word probabilities themselves. The logarithm operator is applied to the ratio of co-occurrence probability values, which converts the ratios into differences. Meaning is then encoded as vector differences in this (log) word vector space. A global word-word co-occurrence matrix is calculated from the target corpus, and the word embedding model is trained considering only the nonzero elements in this matrix.

3.2.4 fastText embedding

In [4], the authors propose the *fastText* word embedding, which is based on a relatively simple linear classifier. The model efficiently computes n-gram features that capture information regarding local word ordering. The features are combined into a “hidden” variable, which is a latent vector representation of the input text. A *softmax* function is applied to the

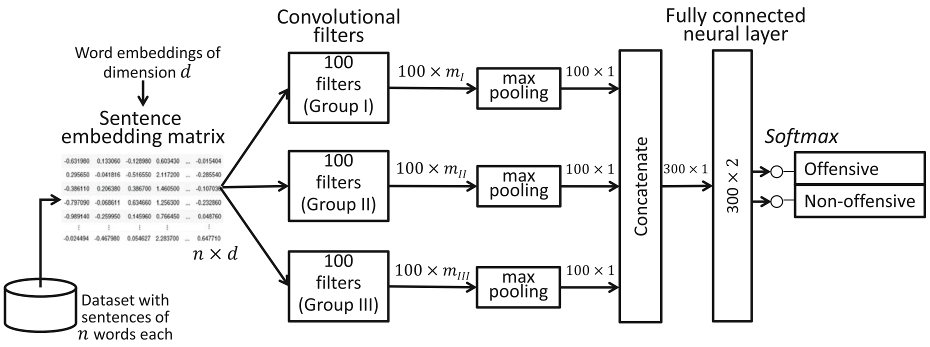

Fig. 1 CNN architecture for classification of sentences written in Portuguese language

values resulting from the application of a weight matrix to this hidden variable. The *softmax* outputs represent the probability of the input text belonging to each considered class.

3.3 Classification model

In this work, we use the CNN architecture for the classification model. Although the CNN architecture is mainly used for image processing tasks, it has recently achieved good performance in NLP tasks, for example, as in [19] and [38]. We adopt the architecture proposed in [19], which is used with sentences in Portuguese language. Figure 1 illustrates this architecture. In this picture, the dataset used for classification contains sentences, and each sentence has n words w_i, where $i = \{1, 2, \ldots, n\}$. For one sentence, a matrix with the word embeddings for each of the words in that sentence is assembled. That is, the sentence embedding matrix of this sentence has size $n \times d$, where d is the dimension of each word embedding. This sentence embedding matrix is processed by 300 2-D convolutional filters. These filters are divided into three groups, namely Group A, B and C. Each group contains the same number of filters (*i.e.* 100 filters). A different kernel size is used for the filters of each group. Table 1 describes the filter kernel size adopted in the three groups. Depending on the kernel size, the convolutional filters generate feature maps with different lengths, m_A, m_B and m_C, also described in Table 1, wherein n is the sentence length. Figure 2 illustrates the sentence embedding matrix, one Group A filter and the formation of the feature map for this filter. Rectified Linear Unit (ReLU) activation functions are applied after each convolution. A max-pooling operation is applied to each feature map, so that one scalar value is obtained for each feature map. Each filter group then yields vectors of dimension 100×1 (100 filters per group $\times$ one scalar representing each feature map). These vectors are concatenated into a single vector of dimension 300×1. This vector serves as input for a fully-connected layer with two outputs. Finally, a *softmax* layer is used after the fully-connected layer, in order to

Table 1 CNN filter kernel size and features map length, wherein d stands for the dimension of the word embedding, and n is the sequence length

Group	Filter kernel size	Feature map length
A	$3 \times d$	$m_A = n - 2$
B	$4 \times d$	$m_B = n - 3$
C	$5 \times d$	$m_C = n - 4$

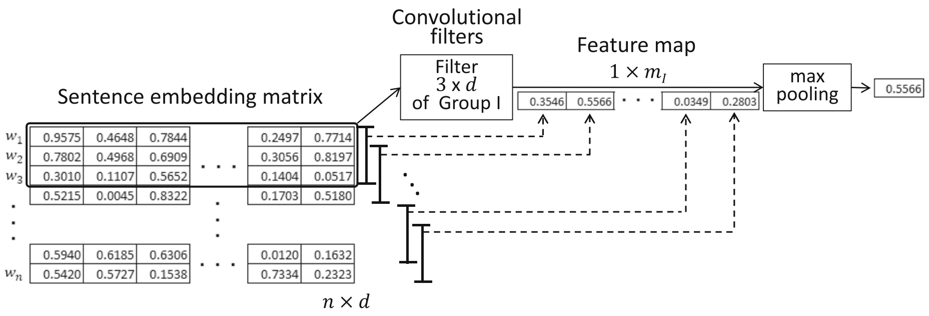

Fig. 2 Illustration of a sentence embedding matrix and CNN kernel filter of Group A

obtain the probability values, associated with the input sentence, of belonging to either of the two classes (*i.e.* offensive or non-offensive).

4 Datasets

In this section, we present the datasets that are used for the classification task, and the ones that are used for the language representation.

4.1 Data for classification

Considering Brazilian Portuguese language, in [9], the authors present offensive and non-offensive comments collected from the Politics and Sports sections of a Brazilian news website. A total of 10,336 comments posted in 115 news from the website were collected. From this number of comments, 1,250 comments were randomly selected and were analyzed separately by three human subjects. Each subject classifies the comment as offensive or non-offensive. The final label for the considered comment corresponds to the classification given by the majority of the three reviewers. In this case, at least two subjects agreed upon the classification. This dataset is called OffComBr2[4], which stands for Offensive Comments from Brazil. From this dataset, the subset of comments, upon which all three human subject classifications agree, is extracted, in order to create another dataset. This dataset is called OffComBr3, and it contains 1,033 comments.

Considering Portuguese language from Portugal, in [13], the author presents a dataset[5] extracted from Twitter. From the original 42,390 extracted tweets, 5668 tweets were labeled as hateful or non-hateful by two human subjects. This dataset is denoted as HsdPt, which stands for Hate speech dataset from Portugal.

Table 2 provides a numerical description of the datasets considered in this work, wherein LSL represents the longest sentence size, and VS the vocabulary size. Dataset HsdPt is the largest one, but its sentences have smaller lengths. The longest sentence has 36 words. In the three datasets, namely, OffComBr2, OffComBr3 and HsdPt, most comments are classified as non-offensive: more than 66% comments in each dataset.

[4]http://inf.ufrgs.br/~rppelle/hatedetector/, last accessed in October 23rd, 2020.
[5]https://rdm.inesctec.pt/id/dataset/cs-2017-008, last accessed in October 23rd, 2020.

Table 2 Description of the datasets used for classification

Dataset	#Comments	Offensive	Non-Offensive	LSL	VS	#Tokens
OffComBr3	1,033	202 (19.5%)	831 (80.5%)	94	412	10,255
OffComBr2	1,250	419 (33.5%)	831 (66.5%)	94	480	12,207
HsdPt	5,668	1,228 (21.5%)	4,440 (78.5%)	36	2051	75,223

In the present work, to perform the training and testing of the model classification, each dataset was randomly divided in two sets at the 90/10 ratio as described in Section 5. Table 3 provides the statistics for training set (90% of the dataset), and test set (10% of the dataset), for each dataset. It can be observed that, in general, the proportionality of the class of comments is maintained in the subsets of each dataset. Only OffComBr2 test set had a 6% increase in the number of non-offensive comments. It is also noted that the size of comments in the subsets of each dataset is kept the same, on average, and that the HsdPt dataset has less variability in relation to the comments size.

4.2 Data for language modeling

The NILC-embeddings[6], proposed in [17], is a repository of word embeddings. These word embeddings were trained for Portuguese language from Brazil and Portugal using texts that come from a variety of sources. The texts represent different genres (*i.e.* encyclopedic articles, informative, prose, and so on). Overall, the text dataset contains more than 1.3 billion of tokens. With this dataset, in [17], the authors trained four word embeddings models: *word2vec*, *fastText*, *wang2vec* and *GloVe*. The time required to train each model was not informed in [17], but similar works reported training times of approximately 24 hours, as in [22], and approximately 8 hours, according to [30]. In the present work, we consider the word embeddings that are available in the NILC repository. These versions are denoted as "NILC-embeddings" throughout the remaining text.

In [22], the authors modified the *word2vec* model, and created the *wang2vec* model to be able to generate word embeddings from text provided by the user (new corpus, our datasets in this work). The authors developed the code implementation for training the models *word2vec* and *wang2vec* and made it publicly available[7]. In this work, we use this code implementation to train the Skip-gram models of *word2vec* and *wang2vec*. In order to train the Skip-gram model of *fastText*, we use the code implementation publicly available in [4][8]. In order to train the *GloVe* model, we use the code publicly available from the NLP group of Stanford University[9]. We investigated the effectiveness of these four models, using the OffComBr2, OffComBr3 and HsdPt datasets for training and validation. We consider three different lengths per word embedding: 50, 100, or 300. On average, 4 seconds are required to train each word embedding. This is because the considered datasets contain a relatively small number of tokens. The HsdPt dataset contains the largest number of tokens: approximately 75,000, as shown in Table 2.

[6]http://nilc.icmc.usp.br/embeddings, last accessed in October 24th, 2020.
[7]https://github.com/wlin12/wang2vec, last accessed in October 24th, 2020.
[8]https://fasttext.cc/docs/en/support.html, last accessed in October 24th, 2020.
[9]https://github.com/stanfordnlp/GloVe, last accessed in October 24th, 2020.

Table 3 Statistics of the datasets for training and testing sets, wherein *Off* and *Non-off* stand for offensive and non-offensive comments, respectively

Dataset	Step	#Comments		Comment length statistics			
		Offensive	Non-offensive	Min	Max	Mean	Std
OffComBr3	Training	182 (19.6%)	748 (80.4%)	1	94	13.50	11.47
	Testing	20 (19.4%)	83 (80.6%)	1	62	13.19	11.24
OffComBr2	Training	384 (35.1%)	741 (65.9%)	1	94	13.04	11.11
	Testing	35 (28.0%)	90 (72.0%)	1	62	12.54	8.90
HsdPt	Training	1118 (22.0%)	3984 (78.0%)	3	36	15.82	6.42
	Testing	110 (20.0%)	456 (80.0%)	3	34	15.98	6.49

5 Performance results

In this section, we present the results obtained using the techniques explained in Section 3, applied to the three datasets described in Section 4. These results form the basis for a solid conclusion, with respect to the impact of a twofold transfer learning approach: use of pre-trained word embeddings, and use of word embeddings obtained through the exploitation of the proper dataset. We report the time required for training the CNN-based classifier considering each word embedding, which is given in seconds. For pre-trained word embeddings (*i.e.* NILC-embeddings), the reported times correspond only to the ones required for training the classifier. For word embeddings trained in the application dataset, the reported times do not comprise the time required for training the word embeddings. This is done using the code offered by the developers of the model.

In the present work, CNN-based classifier is modeled and trained using the open-source TensorFlow library [1]. For network training, the three datasets described in Section 4 are used for each case of considered word embeddings. Table 4 describes the CNN hyper-parameters adopted in all cases. For comparison purposes, the setting values that gave the best results in [33] are adopted. Note that the dropout rate is regarding the fully connected layer.

The training is performed using the Adam optimization algorithm, as it is the case in [9, 13, 21, 24, 33]. These works are used in the comparison reported later in this section. The parameter settings for the Adam algorithm are shown in Table 5. We do not explore other optimizers as one would expect the results to be somehow algorithm-independent.

During the CNN training, for each experiment carried out in this work, the 10-fold cross-validation technique is used. Each dataset for classification is initially divided into two sets: training and testing. The test set, composed by 10% of the data, is not used for training. The training set, composed of the remaining 90% of the data, is divided into ten parts. Nine parts are used to train the model, and one is used to validate it. This training/validation process is repeated ten times. At each time, different parts are used for training and validation. By the

Table 4 Hyper-parameter settings for CNN training

Hyper-parameter	Value
Number of epochs	50
Size of mini-batch	50
Dropout rate	0.5

Table 5 Parameter settings according to the Adam optimizer

Hyper-parameter	Value
Learning rate	0.001
Beta1	0.9
Beta2	0.999
Epsilon	10^{-8}

end of this process, each part is used once to validate the model. We then have ten models trained with the same architecture. Each model is tested using the test set. The performance of the learning process is evaluated using the accuracy and F1-score metrics.

Training performance is mainly based on neural network accuracy in predicting the class of the test data. Let TP be the number of true positive classifications, TN the number of true negative cases, FN that of false negative classifications and FP the number of false positive cases. The accuracy (ACC) metric represents the model correct classification rate with respect to the entire dataset. It is usually expressed as percentage, as it is always the case in this work. It is defined as in (1):

$$ACC = \frac{TP + TN}{TP + FN + TN + FP}. \tag{1}$$

The F-Measure, also known as F1-score ($F1S$), combines precision and sensitivity in order to yield a unique value that indicates the general quality of the trained model. The F1-score represents the harmonic mean between two other metrics, which quantify the precision and sensitivity (or recall) of the trained model. Note that the closer $F1S$ is to 1, the better the model is. It is defined in (2):

$$F1S = \frac{2TP}{2TP + FP + FN}. \tag{2}$$

5.1 Precision, sensitivity and accuracy analysis

In this section, we present the results obtained for the three considered datasets. For each dataset, we show the results of 24 experiments, varying three aspects: *(i)* the nature of the word embeddings, which can be either NILC-based or dataset-based; *(ii)* the word embeddings representation, which can be *wang2vec*, *word2vec*, *fastText* or *GloVe*; *(iii)* the vector dimension used to represent a given word, which can be 50, 100 or 300. That corresponds to a total of 72 experiments, considering the three studied datasets. For each experiment, we provide the F1-score, which is a combined measure of precision and sensitivity, and accuracy metrics, as well as time requirements for training the classification model (given in seconds). In this section, we only focus on the accuracy and F1-score metrics obtained in each experiment, in order to select the best configurations. In Sections 5.2 and 5.3, we provide a dominance analysis, which takes into account both accuracy and CNN-based classifier training time values for such decision-making process.

5.1.1 Results for dataset OffComBr3

Table 6 shows the results obtained in the present study, using the CNN classifier. These results are based on the pre-trained word embeddings *wang2vec*, *word2vec*, *fastText* and *GloVe*, identified as NILC-embeddings (left side), and based on the respective word embeddings trained with data from the OffComBr3 dataset itself (right side). Note that dim refers

Table 6 Results obtained for dataset OffComBr3, wherein the best classification results are shown in red; the second best in blue; the third best in green, and the non-dominated solutions with grey background

	NILC-embedding						OffComBr3-embedding					
Dim	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time
	wang2vec			word2vec			wang2vec			word2vec		
50	0.89	80.10	134.74	0.88	79.61	154.33	0.89	80.39	151.98	0.89	80.10	155.01
100	**0.89**	**81.19**	143.45	0.88	79.71	166.48	0.89	80.39	154.22	0.88	79.83	162.12
300	0.89	80.89	202.92	**0.89**	**81.17**	202.88	0.89	80.84	200.35	**0.89**	**81.05**	201.51
	fastText			GloVe			fastText			GloVe		
50	0.88	78.54	133.78	0.89	80.10	147.42	0.89	80.39	129.64	0.88	79.61	133.32
100	0.88	79.22	154.24	0.89	80.29	188.26	0.88	79.22	145.22	0.88	79.51	141.92
300	**0.89**	**81.17**	185.83	0.88	80.00	197.81	0.89	80.97	160.11	0.89	80.00	168.51

to the word-embedding dimension, denoted by d in Figs. 1 and 2. The three highest accuracy values are shown in colored boldface font. For this dataset, the best classification performance (red) is observed when using the NILC-embeddings *wang2vec* with dimension 100. It is similar to the second best performance (blue), which was obtained using the NILC-embeddings *word2vec* and *fastText*, both with the same dimension 300. Nonetheless, the third best performance (green) is achieved when using embeddings *word2vec* trained in the dataset itself. Note that the F1-scores are compatible in all four cases. The main difference is that the best case scenario (NILC-embedding *wang2vec*) uses 200 fewer entries in the corresponding word embeddings than the second cases (NILC-embeddings *word2vec* and *fastText*) and the third case (OffComBr3-embeddings *word2vec*). Of course, this has an impact on the required training time, which is on average about 20% shorter in the best scenario.

The OffComBr3 dataset has the smallest number of comments, as it can be confirmed in Table 2. Even so, the results presented in Table 6 indicate similar performances, with a slight advantage of the NILC-embedding compared to the OffComBr3-embedding.

5.1.2 Results for dataset OffComBr2

As for the dataset OffComBr3, Table 7 shows the results achieved using the pre-trained word embeddings *wang2vec*, *word2vec*, *fastText* and *GloVe* based on the NILC-embeddings (left side), and using the respective word embeddings trained on the OffComBr2 dataset itself (right side). For this dataset, in terms of accuracy, the two best performances are achieved with NILC-embedding: the first (red) with *word2vec* and the second (blue) with *GloVe*. The third best performance (green) is achieved when no transfer learning is used *i.e.* the word representation is trained in the OffComBr2 dataset itself. It is obtained using *GloVe*. The three cases are highlighted. Note that all three scenarios are based on word vectors of dimension 300, and the F1-scores are similar in all three cases. Nonetheless, the winning (or very best) case with respect to accuracy (NILC-embedding *word2vec*) requires about 20% more time than the other two cases.

For this dataset, note that the achieved accuracy is the lowest one, when compared to those obtained for the other two datasets. One possible explanation for this low performance is the fact that almost 18% of the dataset sentences are equally annotated by only two out

Table 7 Results obtained for dataset OffComBr2, wherein the best classification results are shown in red; the second best in blue; the third best in green, and the non-dominated solutions with grey background

	NILC-embedding						OffComBr2-embedding					
Dim	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time
	wang2vec			word2vec			wang2vec			word2vec		
50	0.84	74.88	164.28	0.82	72.96	169.30	0.84	75.52	159.41	0.84	75.76	186.04
100	0.85	76.72	170.72	0.85	76.56	167.34	0.85	76.20	162.16	0.85	76.40	217.50
300	0.85	76.94	214.71	**0.85**	**77.84**	219.32	0.85	76.08	218.80	0.85	76.12	240.69
			fastText			GloVe			fastText			GloVe
50	0.83	74.40	138.24	0.84	75.92	143.14	0.83	74.24	135.03	0.84	75.44	128.79
100	0.85	76.72	145.67	0.85	76.40	152.06	0.85	76.16	150.82	0.84	75.04	134.00
300	0.85	77.12	181.11	**0.85**	**77.36**	196.82	0.85	76.72	187.33	**0.86**	**77.20**	181.29

of three human subjects, as explained in Section 4.1. This suggests that such sentences may not contain clear hate and/or offensive words, but rather subtle ones, which makes their assessment more subjective. These comments are hence more difficult to be correctly classified by the model.

5.1.3 Results for dataset HsdPt

Table 8 presents the results yielded by the present work using the HsdPt dataset, which is unbalanced as explained in Section 4.1. Considering the three studied datasets, HsdPt is the richest one, having the largest number of comments. The results reveal similar classification performances using NILC-embedding or HsdPt-embedding. However, for this dataset, the best (red) and third (green) performances are achieved by the HsdPt-embeddings *wang2vec* and *word2vec*, respectively. The second performance (blue) is obtained when using NILC-embedding *GloVe*. All three scenarios are based on word embeddings of dimension 300,

Table 8 Results obtained for dataset HsdPt, wherein the best classification results are shown in red; the second best in blue; the third best in green, and the non-dominated solutions with grey background

	NILC-embedding						HsdPt-embedding					
Dim	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time	$F1S$	ACC	Time
	wang2vec			word2vec			wang2vec			word2vec		
50	0.90	84.51	156.72	0.91	85.27	161.64	0.91	85.57	149.57	0.91	85.32	155.00
100	0.92	86.08	176.22	0.91	85.90	175.16	0.91	85.41	170.75	0.91	85.67	180.72
300	0.92	86.33	246.22	0.92	86.36	239.83	**0.92**	**86.70**	234.57	0.92	86.29	241.51
	fastText			GloVe			fastText			GloVe		
50	0.91	85.46	145.22	0.91	85.04	140.16	0.91	84.82	148.84	0.91	84.81	156.17
100	0.91	85.85	172.75	0.92	86.17	157.74	0.91	86.04	163.61	0.92	86.34	188.44
300	0.92	86.52	233.84	**0.92**	**86.68**	224.46	0.92	86.33	221.50	**0.92**	**86.55**	238.50

and so their required training times are similar. Also, note that F1-scores are the same in all three cases.

5.2 Multi-criteria result analysis

Selecting one of the studied scenarios according to their performances regarding the achieved accuracy and required CNN-based classifier training time constitutes a multi-criteria decision making problem [11]. Figure 3 shows the fronts formed by the non-dominated cases among the 72 performed experiments. A solution s is said to be *dominated* by another solution s' if and only if all metrics of s' are better than all respective metrics of s. A solution s is said to be *non-dominated* if and only if no other solution s' in the solution set dominates s. When both accuracy and time are of concern, it is safe to select one of the scenarios that are included on the fronts that are shown. For dataset OffComBr3, there are only two non-dominated cases, with one NILC-based and the other dataset-based. For OffComBr2, there are eight such cases with six NILC-based and two dataset-based, and for HsdPt, there are seven cases, with four NILC-based and three dataset-based. For each dataset, these non-dominated cases are highlighted with a grey background in Table 6, Table 7 and Table 8, respectively.

Table 9 shows the number of cases that are dominated by the solution achieved by each configuration. For each dataset, and for a point given in Fig. 3, the number of dominated cases is obtained by counting the points located to the right of (*i.e.* solutions with longer training times) and below (*i.e.* solutions with lower accuracy values) the given point. The highest number of dominated solutions for each dataset (highlighted in red) occurs for NILC-embedding with dimension 100, with the *fastText*, *wang2vec* and *GloVe* models for the OffComBr2, OffComBr3 and HsdPt datasets, respectively. These configurations belong to the set of non-dominated solutions, as it can be verified in Tables 6, 7 and 8. The second highest (highlighted in blue) and the third highest (highlighted in green) numbers of dominated solutions are distributed between NILC-embedding and dataset-embedding. The absence of high occurrences of dominated solutions using the *word2vec* model is also notorious. This can be explained by the fact that the *word2vec* embedding presents, in general, solutions with the highest training time, as it can be observed in Tables 6, 7 and 8.

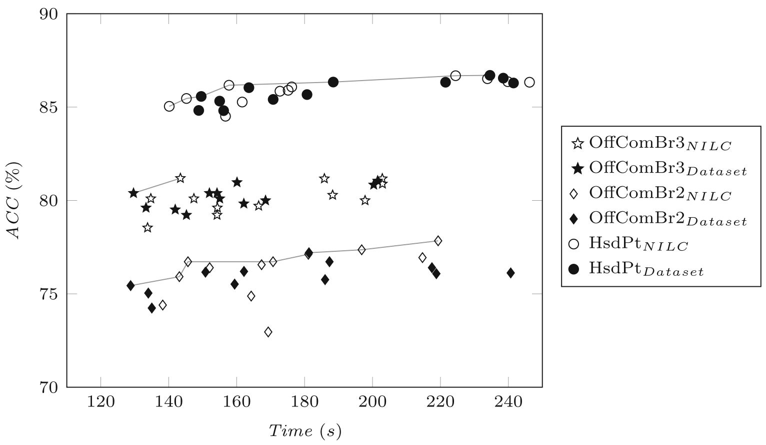

Fig. 3 Fronts formed by the non-dominated solutions within the results obtained from datasets OffComBr2, OffComBr3, and HsdPt

Table 9 Total numbers of solutions that are dominated by each configuration (dimension, embedding method, and embedding dataset), for each dataset: Off2, Off3 and Hsd stand for OffComBr2, OffComBr3 and HsdPt, respectively. In each dataset, the highest number of dominated solutions is shown in red; the second highest in blue; the third highest in green

	NILC-embedding						Dataset-embedding					
Dim	Off2	Off3	Hsd	Off2	Off3	Hsd	Off2	Off3	Hsd	Off2	Off3	Hsd
	wang2vec			word2vec			wang2vec			word2vec		
50	1	**8**	0	0	0	0	2	**8**	**5**	0	4	3
100	4	**18**	1	5	0	1	5	**8**	0	2	1	0
300	3	0	0	1	1	2	0	0	4	0	1	0
	fastText			GloVe			fastText			GloVe		
50	1	0	**6**	4	6	3	1	**14**	2	5	4	1
100	**11**	0	1	**7**	1	**7**	**6**	0	4	4	2	3
300	**6**	5	3	4	0	**5**	3	7	1	**6**	0	3

5.3 Ranking analysis

Furthering up multi-criteria analysis, it would be interesting to reach a kind of ranking of the studied word embeddings regarding achieved accuracy and required CNN-based classifier training time. We obtain the rankings between scenarios using two different methods: (i) calculating the ratio between accuracy and normalized classifier training time in each configuration, which we denote as *performance factor (PF)* metric; and (ii) using the improved version of Preference Ranking Organization METHod for Enrichment of Evaluations, also called PROMETHEE-II metric [5]. First, in Section 5.3.1, we briefly present and explain how this metric is used in this work. Then, in Section 5.3.2, we analyze and discuss the ranking results considering both metrics.

5.3.1 Promethee-II metric

The PROMETHEE methods are a set of procedures designed to aid in selecting, among a set of possible alternatives, the one that is the most efficient to solve a multi-criteria problem. In this case, efficiency is determined by taking into account performance values from the considered multiple criteria simultaneously [5]. In this work, we use the PROMETHEE-II method, which assumes that all possible alternatives are comparable. So, a complete ranking between theses alternatives can be obtained. The rankings are based on the PROMETHEE-II scores, calculated using the function defined in (3):

$$\phi(a) = \frac{1}{\#A - 1} \sum_{c=1}^{C} \sum_{x \in A} \left(P_c(a, x) - P_c(x, a)\right) w_c, \tag{3}$$

where $\phi(a)$ denotes the PROMETHEE-II score from alternative a, $\#A$ is the number of possible alternatives in set A, C is the number of criteria observed during the decision-making process, P_c is the preference function for criterion c, and w_c is the weight given to such criterion in the decision-making process.

In this work, each alternative represents a different configuration, which is set considering three aspects: word-embedding dimension size (50, 100 or 300), word-embedding type (*wang2vec*, *word2vec*, *fastText*, or *GloVe*), and word-embedding training dataset (NILC or the corresponding application-specific dataset). Thus, for each dataset, the possible alternatives are $\#A = 24$ different configurations. In this analysis, the considered criteria are accuracy and CNN-based classifier training time. So, we have $C = 2$. The weights w_c are normalized such that $w_1 + w_2 = 1$. We assign equal weights for the two considered criteria ($w_1 = w_2 = 0.5$). The preference function $P_c(a, b) = F_c\left(d_c(a, b)\right)$ returns a real value representing how much configuration a is preferable over configuration b based on the difference between their respective performance values regarding criterion c. This can be expressed as in (4):

$$P_c(a, b) = F_c\left(d_c(a, b)\right), \tag{4}$$

where $F_c(\cdot)$ is a monotonically-increasing function, and $d_c(a, b) = g_c(a) - g_c(b)$ is the difference between performance values $g_c(a)$ and $g_c(b)$, regarding criterion c, for configurations a and b, respectively. In this work, for each criterion, the difference function is computed between each alternative performance value and the average performance value regarding all alternatives for such criterion. Furthermore, we use the preference function defined in (5) [5]:

$$P_c(d_c) = \begin{cases} 0, & \text{if } d \leq 0 \\ 1 - e^{-\frac{d_c^2}{2s_c^2}}, & \text{if } d > 0 \end{cases} \tag{5}$$

Figure 4 depicts the preference function defined in (5). Parameter s corresponds to the curve inflection point. In this work, parameters $p = \max_{a \in A}\left(g_c(a) - \overline{g_c}\right)$ and $q = \min_{a \in A}\left(g_c(a) - \overline{g_c}\right)$ correspond to the largest and smallest difference values obtained for criterion c for a given dataset, respectively. Note that $\overline{g_c}$ is the average performance value regarding criterion c for a given dataset. We define s as the average value between p and q: $s_c = (p + q)/2$. The PROMETHEE-II scores $\phi(\cdot)$ range from -1 (least preferable alternative) to 1 (most preferable alternative).

Table 10 shows the PROMETHEE-II scores obtained from the considered configurations regarding each dataset. Observe that the configurations with the highest scores, which are marked in red for each dataset, are the same ones that correspond to solutions with the largest number of dominated solutions in the corresponding datasets. These are also marked in red in Table 9. For instance, the *wang2vec* word embedding, with dimension 100 and

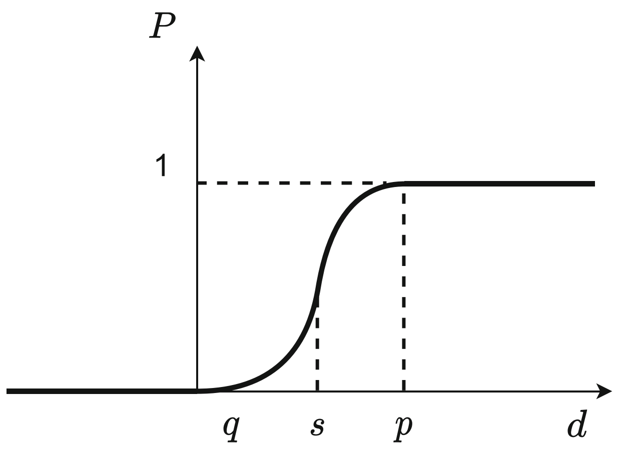

Fig. 4 Preference function used in this work

Table 10 PROMETHEE-II scores for the three datasets, wherein Off2, Off3 and Hsd stand for OffComBr2, OffComBr3 and HsdPt, respectively. The best results (highest scores) in each dataset are shown in red; the second best in blue; the third best in green. The scores range from −1 (least preferable alternative) to 1 (most preferable alternative)

	NILC-embedding						Dataset-embedding					
Dim	Off2	Off3	Hsd	Off2	Off3	Hsd	Off2	Off3	Hsd	Off2	Off3	Hsd
	wang2vec			word2vec			wang2vec			word2vec		
50	-0.260	0.210	-0.258	-0.462	-0.174	-0.085	-0.083	0.198	0.078	-0.167	0.048	-0.041
100	0.196	**0.574**	0.201	0.171	-0.247	0.119	0.098	0.181	-0.080	-0.179	-0.144	-0.031
300	- 0.016	-0.031	-0.136	0.159	0.040	-0.109	-0.274	-0.037	0.019	-0.363	0.020	-0.137
	fastText			GloVe			fastText			GloVe		
50	-0.196	- 0.244	0.052	0.117	0.108	-0.067	-0.196	**0.369**	-0.166	0.051	-0.012	-0.199
100	**0.329**	-0.300	0.108	0.207	-0.216	**0.333**	0.147	-0.230	**0.249**	-0.074	-0.115	**0.219**
300	**0.241**	0.154	-0.037	0.198	-0.412	0.054	0.096	**0.390**	-0.046	**0.260**	-0.126	-0.041

trained in NILC, has the highest score in dataset OffComBr-3 (see Table 10), and this configuration yields the most dominant case (*i.e.* the solution in OffComBr-3 dataset), with 18 dominated solutions (see Table 9). Solutions represented by such configurations are also *non-dominated solutions*. Hence, they are at the Pareto front of the respective datasets, as shown in Fig. 3. This result suggests that the most preferable configurations, according to PROMETHEE-II scores, are in agreement with the configurations that represent the best trade-off solutions between accuracy and classifier training time in the respective datasets.

5.3.2 Ranking results

For ranking purposes, we compute the weighted average values considering all three datasets. Given the volume of data available in each dataset, as presented in Table 2, HsdPt has approximately 6$\times$ the volume of OffComBr2 and OffComBr3. So, we set the weights as 1/8 for those two datasets and 6/8 for HsdPt. These are presented in Table 11, wherein the three best scenarios for NILC and dataset-oriented word embeddings are highlighted: the best case in red, second best in blue and third best in green.

Considering accuracy in Table 11, note that all best cases occur for word embeddings of dimension 300, as it is *often* the case when the datasets are considered separately. For classifier training time, the best cases occur for word embeddings with the smallest considered dimension, which is 50. This is also expected, as smaller input sizes lead to less parameters in the classification model. Thus, training speed is increased, as less parameters need to be optimized. Regarding PROMETHEE-II scores, the best cases are associated with word embeddings with size 100, which is in agreement with what this metric represents. Here, the PROMETHEE-II scores take into account the trade-off between accuracy and training time. Choosing the intermediate word embedding size of 100 reflects a reasonable trade-off between accuracy and training speed, because larger embedding sizes (300) lead to better (*i.e.* higher) accuracy values, but worse (*i.e.* higher) training times. Conversely, smaller embedding sizes lead to worse (*i.e.* lower) accuracy values, but better (*i.e.* lower) training times. Figures 5, 6, and 7 provide a visual comparison of average accuracy values, a visual comparison of average training times, and a visual comparison of the average normalized

Table 11 Weighted average results of accuracy, time and PROMETHEE-II scores regarding the three considered datasets, wherein the best results regarding each metric (accuracy, time and PROMETHEE-II scores) are shown in red; the second best ones in blue; the third best ones in green. The scores range from −1 (least preferable alternative) to 1 (most preferable alternative)

	NILC-embedding						Dataset-embedding					
Dim	*ACC*	Time	Score	*ACC*	Time	Score	*ACC*	Time	Score	*ACC*	Time	Score
	wang2vec			word2vec			wang2vec			word2vec		
50	82.755	154.918	-0.200	83.024	161.684	-0.143	83.666	151.101	0.073	83.473	158.881	-0.045
100	84.299	171.436	**0.247**	83.959	173.098	0.080	83.631	167.610	-0.025	83.781	182.993	-0.064
300	84.476	236.869	-0.108	**84.646**	232.648	-0.057	84.640	228.321	-0.025	84.364	236.408	-0.145
	fastText			GloVe			fastText			GloVe		
50	83.213	**142.918**	-0.016	83.283	**141.440**	-0.022	82.944	**144.714**	-0.103	82.989	149.891	-0.144
100	83.880	167.051	0.084	84.214	160.845	**0.248**	83.953	159.713	**0.176**	84.074	175.820	0.141
300	**84.676**	221.248	0.021	**84.680**	217.674	0.014	84.459	209.555	0.026	84.563	222.600	-0.014

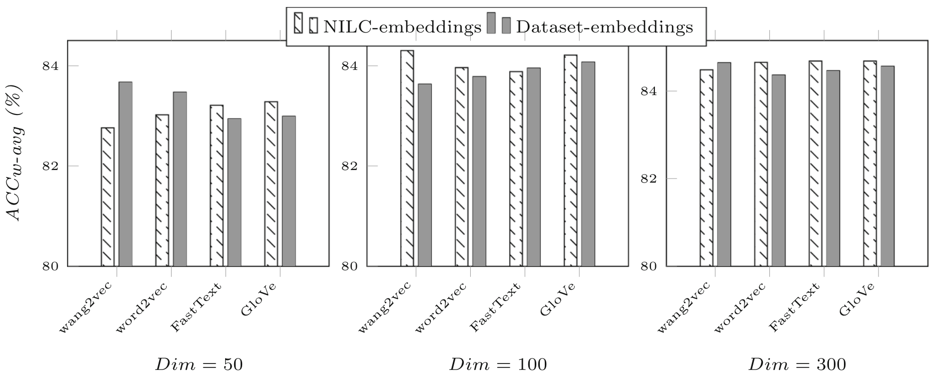

Fig. 5 Average accuracy values, computed with respect to the results from OffComBR3, OffComBR2 and HsdPt datasets

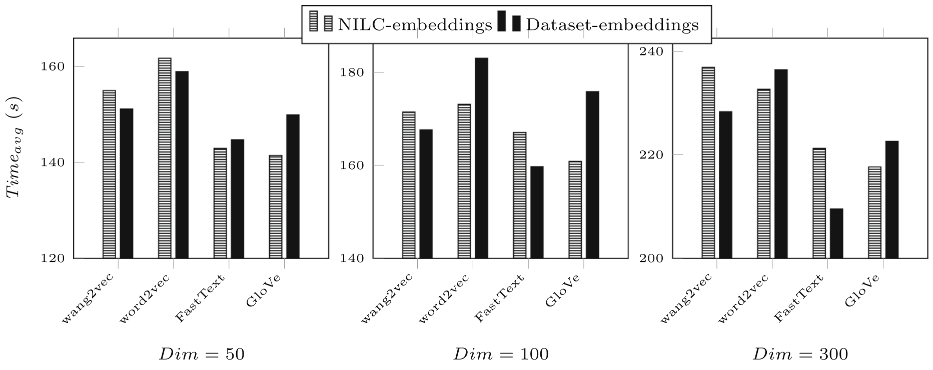

Fig. 6 Average training times, computed with respect to the results from OffComBR3, OffComBR2 and HsdPt datasets

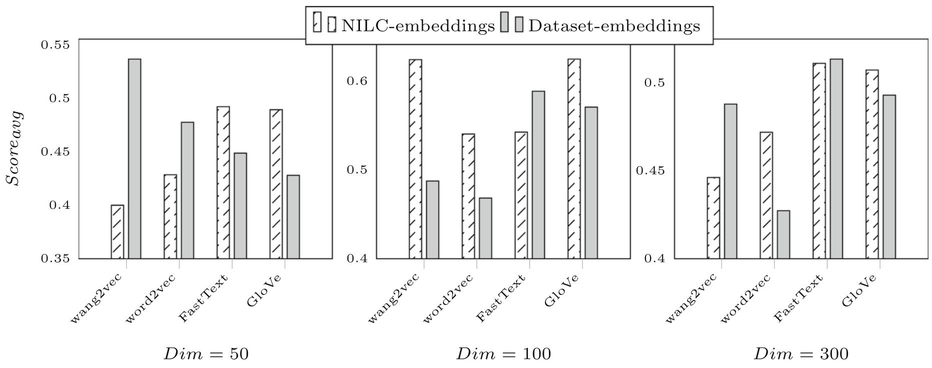

Fig. 7 Average normalized scores, computed with respect to the results from OffComBR3, OffComBR2 and HsdPt datasets

PROMETHEE-II scores, respectively, where the averages are computed with respect to the three datasets under consideration (Table 2).

Table 12 shows the overall performances values and scores, averaged for the studied scenarios (*i.e.*, the three used datasets and the three considered word embedding dimensions). Specifically, it presents the accuracy values, the CNN-based classifier training times, the performance factors PF, and the PROMETHEE-II scores. The performance factor PF is defined as the ratio between the accuracy value and normalized CNN training time. Based on the values of these four metrics, we elaborate four rankings: two mono-criterion ranks: $Rank_{ACC}$, $Rank_{Time}$, and two multi-criteria based ranks: one based on the computed performance factor $Rank_{PF}$ and the other on the PROMETHEE-II score $Rank_{score}$.

Considering the four rankings presented in Table 12, it can be easily noted that NILC-embedding *fastText* and dataset-embedding *GloVe* have maintained the same ranks (third and fifth respectively) independently of the used ranking criterion. This surely shows a robust performance of these scenarios regardless of the dataset used and the word representation dimension.

Another valid observation is that the ranking based on the performance factor and that based on the PROMETHEE-II score provide a similar result, except for the case of NILC-embedding *GloVe* and dataset-embedding *fastText*, which exchanged places in the ranking.

Table 12 Overall average performances of the studied scenarios and their respective ranking

	NILC-embedding				Dataset-embedding			
Metric	wang2vec	word2vec	fastText	GloVe	wang2vec	word2vec	fastText	GloVe
ACC	83.843	83.876	83.923	84.059	83.979	83.873	83.785	83.875
Rank$_{ACC}$	7	4	3	1	2	6	8	5
$Time$	187.741	189.143	177.072	173.320	182.344	192.760	171.327	182.770
Rank$_{Time}$	6	7	3	2	4	8	1	5
PF	0.861	0.855	0.913	0.935	0.888	0.839	0.943	0.885
Rank$_{PF}$	6	7	3	2	4	8	1	5
$Score$	−0.020	−0.040	0.030	0.080	0.008	−0.085	0.033	−0.006
Rank$_{Score}$	6	7	3	1	4	8	2	5

Table 13 Characteristics of related works for the performance comparison

Reference	Representation	Classification
[9]	n-grams	SVM 10-folds
[21]	n-grams+	SVM
[33]	NILC-embeddings	CNN 10-folds
[13]	n-grams	SVM Linear 3-folds
Proposed$_{NILC}$	NILC-embeddings	CNN 10-folds
Proposed$_{Dataset}$	Dataset-embeddings	CNN 10-folds

Nonetheless, it is safe to conclude that the former is best when NILC-embedding is used and the latter is best when dataset-embeddings are preferred. The word embedding *word2vec* provides the worst performance in both scenarios: NILC and dataset based. Moreover, word embedding *wang2vec* seems to perform better when word representation training is done within the dataset than in the case of NILC pre-trained representations.

5.4 Performance comparison

The architecture proposed in [19], described in Section 3, is designed to perform sentiment analysis for texts written in the English language. Later on, in [33], the same architecture is used to classify texts written in the Portuguese language. In the research work reported in [33], the authors based their findings on the three datasets described in Section 4. They perform the pre-processing step, which consists of dividing the texts into tokens, rewriting all words in lowercase and removing the punctuation marks from the analyzed text. Then, they compare the performance of CNN model when trained with word vector representations obtained from some word embeddings, such as *wang2vec* and *GloVe* provided by NILC. They considered vectors of size 50, 100 and 300. Here, we only report the cases with best performance. The authors make available the source code used to obtain their results[10], on which we rely to conduct the experiments carried out in the present work. We run the code using the pre-processing described in Section 3.1, and the aforementioned cross-validation process.

In [13], the authors used the HsdPt dataset, extracting the feature vectors using n-grams. As HsdPt presents an imbalance between positive and negative comments, the author limited the number of sentences to 1,228 sentences for each class in order to obtain a balanced set. Hence, the SVM-based classifier was trained with a balanced set of 2,056 sentences only, using a 3-fold cross validation. Testing is performed using a set of 400 sentences that is also balanced. In [33], a CNN-based classifier without compensating for the imbalanced nature of the dataset is exploited.

The characteristics of the compared works are summarized in Table 13. In [21], the word embeddings used are n-grams augmented by meta-attributes yielded from information about the neighboring words in each classified text.

Table 14 presents a comparison of the results obtained in this work with the best ones reported in four related studies, as identified in the first column, for the detection of hate speech and offensive language. The results are reported considering the OffComBr3, OffComBr2, and HsdPt datasets. In this case, the best scenarios for the proposed work are

[10]https://github.com/samcaetano/hatespeech_detector, last accessed in July 25th, 2019.

Table 14 Comparison of the obtained results with existing related works

Set	Reference	Embedding	Dim	$F1S$	ACC	Time
OffComBr3	[9]	–	–	0.82	–	–
	[21]	–	–	0.85	–	–
	[33]	*wang2vec*	300	0.96	92.82	–
	Proposed$_{NILC}$	*wang2vec*	100	0.89	81.19	143.45
	Proposed$_{Dataset}$	*word2vec*	300	0.89	81.05	201.51
OffComBr2	[9]	–	–	0.77	–	–
	[21]	–	–	0.72	–	–
	[33]	*wang2vec*	300	0.89	82.64	–
	Proposed$_{NILC}$	*word2vec*	300	0.85	77.84	219.32
	Proposed$_{Dataset}$	*GloVe*	300	0.86	77.20	181.29
HsdPt	[13]	–	–	0.76	78.30	–
	[33]	*wang2vec*	100	0.96	92.74	–
	Proposed$_{NILC}$	*GloVe*	300	0.92	86.68	224.46
	Proposed$_{Dataset}$	*wang2vec*	300	0.92	86.70	234.57

elected based on accuracy alone. In [13], the HsdPt dataset is augmented so as to have a balanced distribution regarding both classes. Such data balancing is performed neither in the case of [33] nor in the proposed work.

Nonetheless, in the case of the work reported in [33], samples within each dataset are randomly replicated and shuffled, before dividing these samples into the training and test subsets. This procedure may cause some samples to be included both in the training and test subsets that are used by the CNN classifier. This might explain the higher accuracy values reported by the authors, when compared to our proposed solution results. Our results are obtained by training and testing the exact model used in their work, but without performing the sample replication/shuffling procedure.

In view of the availability of metric values for all compared works, we compare, in Fig. 8, the F1-score computed for all the compared works. As it can be observed, in Table 14, the F1-scores obtained in the proposed work are lower that those reported in [33]. This too follows from the shuffle operation with random replications that was performed by the authors before the division of the data into training and test sets, as explained previously.

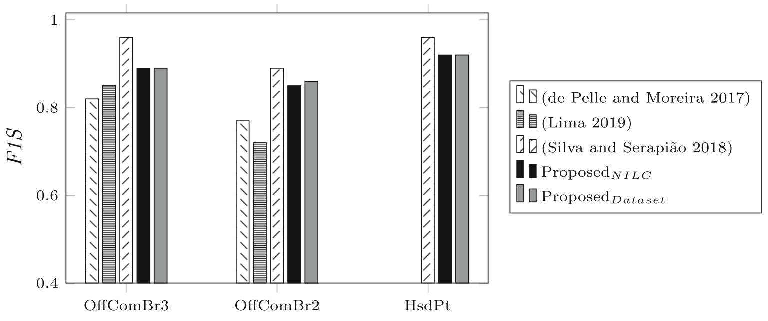

Fig. 8 Comparison among F1-scores achieved in the present work and four related works, considering datasets OffComBr3, OffComBr2, and HsdPt

F1-scores from the proposed solutions are significantly better than those obtained in [9] and [21], both of which are based on SVM classification models, and slightly lower than those obtained in [33], that is also based on a CNN classification model.

6 Conclusions

This paper continues previous works on Portuguese language related to binary classification of texts using CNNs. It experimentally investigates the performance of four embedding methods, namely *wang2vec*, *word2vec*, *fastText*, and *GloVe*. The accuracy and F1-score results seem fairly similar at first glance. A closer analysis, involving trade-offs between accuracy and training time cast on non-dominance fronts, average results on the datasets as well as performance ranks, indicates that using word embeddings that are trained within the application domain may be advantageous in some cases, even though these "local dataset" embeddings are much smaller in terms of the token number. Taking into account the rankings based on performance factors and on PROMETHEE-II scores, *GloVe* is the best method for NILC-embedding, and *fastText* is the best method for dataset-specific embedding. One might expect specific word embedding to yield a better fit to a particular dataset, translating into shorter training and better classifier accuracy. However, directly-applied NILC-embeddings seem to be equally fit.

Tables 6 to 8 present results obtained from word embeddings with size 50, 100 and 300. The word embeddings with the smallest size (50) yield slightly worse accuracy values but, in combination with smaller training times, such embeddings lead to non-dominated solutions which are shown in Fig. 3 and highlighted with a gray background in Tables 6 to 8. Particularly in Table 6, the accuracy of the best NILC-embedding (*wang2vec* with 100 dimensions) is similar to the accuracy of the second-best NILC-embeddings (*word2vec* or *fastText* with 300 dimensions), but the embedding size difference (200) is related to a significant complexity reduction or, in other words, a significant reduction in training time. Another example is observed in Table 8: non-dominated solutions are obtained from word embeddings with size 50 (NILC-embedding or local dataset embedding) and 100 (NILC-embedding). It is possible that, by trying other word embedding sizes, new non-dominated solutions arise.

We point out that previous work [2] investigates pre-processing methods applied to sentiment analysis, comparing the effectiveness of each method in yielding accurate classification. Research focusing on the influence of pre-processing techniques in sentiment analysis for OffComBr3, OffComBr2 and HsdPt data is a relevant future work topic as well.

In order to improve the accuracy results in application-specific word embeddings, one avenue that should be explored consists of using the totality of the available datasets to yield the word vectors. This would improve the quality of the produced word-embedding, which should lead, in turn, to classification process accuracy improvement. For instance, all datasets (without labels) – OffComBr2, and HsdPt – would be merged and used, during training, to develop a single word embedding to be used in hate speech detection within text written in Portuguese language. This implies that the corpus used to generate the word embedding representations would be larger, but the size of these word vectors would still have to be defined based on the tests. As another suggestion for future work, one might use average-sized training sets, with a number of tokens in the order of 10^5, for the language model, thus gathering more sentences within the application domain. Training itself is fast, so a grid search for embedding sizes and embedding methods is feasible.

Acknowledgment This work was supported by *Conselho Nacional de Desenvolvimento Científico e Tecnológico* (CNPq – Brazil), research grants 432997/2018-0, 310841/2019-4 and 440074/2020-7, and by *Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro* (FAPERJ– Brazil), research grant 210.364/2018 and 203.111/2018.

References

1. Abadi M et al (2016) Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv:1603.04467
2. Angiani G, Ferrari L, Fontanini T, Fornacciari P, Iotti E, Magliani F, Manicardi S (2016) A comparison between preprocessing techniques for sentiment analysis in Twitter. In: 2nd international workshop on knowledge discovery on the WEB. Cagliari, Italy
3. Bengio Y, Ducharme R, Vincent P, Jauvin C (2003) A neural probabilistic language model. J Mach Learn Res 3:1137–1155
4. Bojanowski P, Grave E, Joulin A, Mikolov T (2017) Enriching word vectors with subword information. Transactions of the Association for Computational Linguistics 5:135–146
5. Brans J-P, Mareschal B (2005) Promethee Methods. Springer, New York, pp 163–186
6. Cer D, Yang Y, Kong S-Y, Hua N, Limtiaco N, John RS, Constant N, Guajardo-Cespedes M, Yuan S, Tar C, Sung Y-H, Strope B, Kurzweil R (2018) Universal sentence encoder. arXiv:1803.11175
7. Conneau A, Kiela D, Schwenk H, Barrault L, Bordes A (2017) Supervised learning of universal sentence representations from natural language inference data. In: Palmer M, Hwa R, Riedel S (eds) Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP Copenhagen, Denmark, September 9-11, 2017. Association for Computational Linguistics, p 2017
8. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20:273–297
9. de Pelle R, Moreira V (2017) Offensive comments in the brazilian web: a dataset and baseline results, SP, Brazil
10. Devlin J, Chang M.-W., Lee K, Toutanova K (2019) BERT: Pre-training Of deep bidirectional transformers for language understanding. In: 4171–4186. Association for Computational Linguistics
11. Dhiman H, Deb D (2020) Multi-criteria decision-making: An overview Decision and Control, vol 253. Springer, Singapore
12. Ezeibe C (2021) Hate speech and election violence in Nigeria. Journal of Asian and African Studies 56(4):919–935
13. Fortuna P (2017) Automatic detection of hatespeech in text: An overview of the topic and dataset annotation with hierarchical classes. Master's thesis. https://hdl.handle.net/10216/106028, Faculty of Engineering, University of Porto. Porto, Portugal
14. Fortuna P, Nunes S (2018) A survey on automatic detection of hate speech in text. ACM Comput Surv 51(4):1–30
15. Fortuna P, Rocha da Silva J, Soler-Company J, Wanner L, Nunes S (2019) A hierarchically-labeled Portuguese hate speech dataset. In: Proceedings of the Third Workshop on Abusive Language Online. Association for Computational Linguistics, Italy, pp 94–104
16. Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. http://www.deeplearningbook.org MIT Press
17. Hartmann N, Fonseca E, Shulby C, Treviso M, Rodrigues J, Aluísio S. (2017) Portuguese word embeddings: Evaluating on word analogies and natural language tasks. In: Anais do XI simpósio brasileiro de tecnologia da informação e da linguagem humana, Porto Alegre, RS, Brasil, pp 122–131. SBC
18. Joulin A, Grave E, Bojanowski P, Mikolov T (2017) Bag of tricks for efficient text classification. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. Association for Computational Linguistics, Spain, pp 427–431
19. Kim Y (2014) Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Qatar, pp 1746–1751
20. Leite J, Silva D, Bontcheva K, Scarton C (2020) Toxic language detection in social media for Brazilian Portuguese: New dataset and multilingual analysis. In: Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing. Association for Computational Linguistics, China, pp 914–924

21. Lima C, Dal Bianco G (2019) Extração de característica para identificação de discurso de ódio em documentos. In: Anais da XV escola regional de banco de dados, Porto Alegre, RS, Brasil, pp 61–70. SBC
22. Ling W, Dyer C, Black A, Trancoso I (2015) Two/Too simple adaptations of word2Vec for syntax problems. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. Association for Computational Linguistics, Colorado, pp 1299–1304
23. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representation in vector space. arXiv:1301.3781v3
24. Pari C, Nunes G, Gomes J (2019) Avaliação de técnicas de word embedding na tarefa de detecção de discurso de ódio. In: Anais do XVI encontro nacional de inteligência artificial e computacional, porto alegre, RS, Brasil, pp 1020–1031. SBC
25. Pennington J, Socher R, Manning C (2014) GloVe: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics, Qatar, pp 1532–1543
26. Peters M, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1. Association for Computational Linguistics, Louisiana, pp 2227–2237
27. Petrolito R, Dell'Orletta F (2018) Word embeddings in sentiment analysis, Turin, Italy
28. Poletto F, Basile V, Sanguinetti M, Bosco C, Patti V (2021) Resources and benchmark corpora for hate speech detection: a systematic review. Lang Resour Eval 55(2):477–523
29. Pugliero F (2018) Como ódio viralizou no Brasil Available at: https://www.dw.com/pt-br/como-o-odio-viralizou-no-brasil/a-45097506 Accessed: October 23rd
30. Rodrigues J, Branco A, Neale S, Silva J (2016) LX-DSEmVectors: Distributional semantics models for Portuguese language. In: 12Th international conference on computational processing of the portuguese, PROPOR. Tomar, Portugal
31. Roy P, Tripathy A, Das T, Gao X-Z (2020) A framework for hate speech detection using deep convolutional neural network. IEEE Access 8:204951–204962
32. Sherstinsky A (2020) Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Physica D: Nonlinear Phenomena 404:132306
33. Silva S, Serapião A (2018) Detecção de discurso de ódio em português usando CNN combinada a vetores de palavras. In: Symposium on knowledge discovery, mining and learning, KDMILE. São Paulo, Brazil, p 2018
34. Spertus E (1997) Smokey: Automatic recognition of hostile messages. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, AAAI'97/IAAI'97. AAAI Press, pp 1058–1065
35. Thireou T, Reczko M (2007) Bidirectional long short-term memory networks for predicting the subcellular localization of eukaryotic proteins. IEEE/ACM Trans Comput Biol Bioinformatics 4(3):441–446
36. Vargas F, de Góes F, Carvalho I, Benevenuto F, Pardo T (2021) Contextual lexicon-based approach for hate speech and offensive language detection. arXiv:2104.12265
37. Zampieri M, Malmasi S, Nakov P, Rosenthal S, Farra N, Kumar R (2019) Semeval-2019 task 6: Identifying and categorizing offensive language in social media (offenseval). arXiv:1903.08983
38. Zhang Y, Wallace B (2015) A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv:1510.03820