Detection of hate speech spreaders using convolutional neural networks

Notebook for PAN at CLEF 2021

Marco Siino[1], Elisa Di Nuovo[2], Ilenia Tinnirello[1] and Marco La Cascia[1]

[1]*Università degli Studi di Palermo, Dipartimento di Ingegneria, Palermo, 90128, Italy*

[2]*Università degli Studi di Torino, Dipartimento di Lingue e Letterature Straniere e Culture Moderne, Torino, 10124, Italy*

Abstract

In this paper we describe a deep learning model based on a Convolutional Neural Network (CNN). The model was developed for the Profiling Hate Speech Spreaders (HSSs) task proposed by PAN 2021 organizers and hosted at the 2021 CLEF Conference. Our approach to the task of classifying an author as HSS or not (nHSS) takes advantage of a CNN based on a single convolutional layer. In this binary classification task, on the tests performed using a 5-fold cross validation, the proposed model reaches a maximum accuracy of 0.80 on the multilingual (i.e., English and Spanish) training set, and a minimum loss value of 0.51 on the same set. As announced by the task organizers, the trained model presented is able to reach an overall accuracy of 0.79 on the full test set. This overall accuracy is obtained averaging the accuracy achieved by the model on both languages. In particular, with regard to the Spanish test set, the organizers announced that our model achieves an accuracy of 0.85, while on the English test set the same model achieved - as announced by the organizers too - an accuracy of 0.73. Thanks to the model presented in this paper, our team won the 2021 PAN competition on profiling HSSs.

Keywords

Author Profiling, Hate Speech, Twitter, Spanish, English

1. Introduction

The aim of the PAN 2021 Profiling Hate Speech Spreaders (HSSs) on Twitter task [1, 2] was to investigate whether the author of a given Twitter thread is likely to spread tweets containing hate speech or not. The multilingual dataset, namely English and Spanish datasets provided by the organizers of the task, consisted of 120,000 tweets: 200 tweets per author, 200 authors per each language training set and 100 authors per each language test set [3]. The model we used to compete for the task consists of a shallow Convolutional Neural Network (CNN). Broadly speaking, our network preprocess each sample in the dataset to build a dictionary[1] where the

CLEF 2021 – Conference and Labs of the Evaluation Forum, September 21–24, 2021, Bucharest, Romania

marco.siino@unipa.it (M. Siino); elisa.dinuovo@unito.it (E. Di Nuovo); ilenia.tinnirello@unipa.it (I. Tinnirello); marco.lacascia@unipa.it (M. La Cascia)

0000-0002-4453-5352 (M. Siino); 0000-0002-4814-982X (E. Di Nuovo); 0000-0002-1305-0248 (I. Tinnirello); 0000-0002-8766-6395 (M. La Cascia)

[1]In this paper we use the well known computer science concept of a dictionary. A dictionary (or associative array) is a data type composed by a collection of (key,value) pairs. However, in the TextVectorization class definition used in our model is used the word *vocabulary* referring to a list of tokens (e.g., n-grams returned after preprocessing

key is an integer number and the value is an n-gram resulting from our custom preprocessing function. Each integer value (e.g. a key in the dictionary) is then mapped to a single point into a 100-dimensional word embedding space. Then, a 1-dimensional convolution is applied. The output of the convolution layer is then fed into an average pooling and then into a global average pooling layer fully connected to a single dense layer output.

The remainder of this paper is organized as follows. In Section 2 we present some related works about the usage of deep learning methods for similar text classification tasks. In Section 3 we describe in detail our approach, explaining our choices and the configuration of each layer of the model. In Section 4 we report the results obtained in the final 5-fold cross validation and on the test set. In Section 5 we discuss some interesting future works and in Section 6 we conclude our paper.

2. Related works

To address the problem of identifying HSSs within the Twitter microblogging platform, we started from the analysis and study of state-of-the-art techniques for text classification [4, 5, 6]. Our choice to use a deep learning-based approach was dictated by the notable performances reached in various text classification tasks where deep AI methods outperformed classical techniques used in natural language processing (e.g. Bayes, Decision Tree, K-Neearest Neighbour, Support Vector Machine) as reported in [7, 8].

Also hybrid approaches are present in the literature, as in the case of a CNN used to extract textual features and a SVM to carry out the classification and prediction [9]. However, similar results to that obtained by a CNN can be achieved.

As a starting point to develop our model we decided to consider the work conducted in [10] in which, for the first time, a CNN was used to address a text classification task. The CNN obtained promising performances compared to the state of the art. To model text in a vector format, a common representation in most deep learning models is based on word embeddings [11, 12]. And it is thanks to word embedding-based methods that since 2015 [13] deep and non-deep models have performed very well on several text classification tasks.

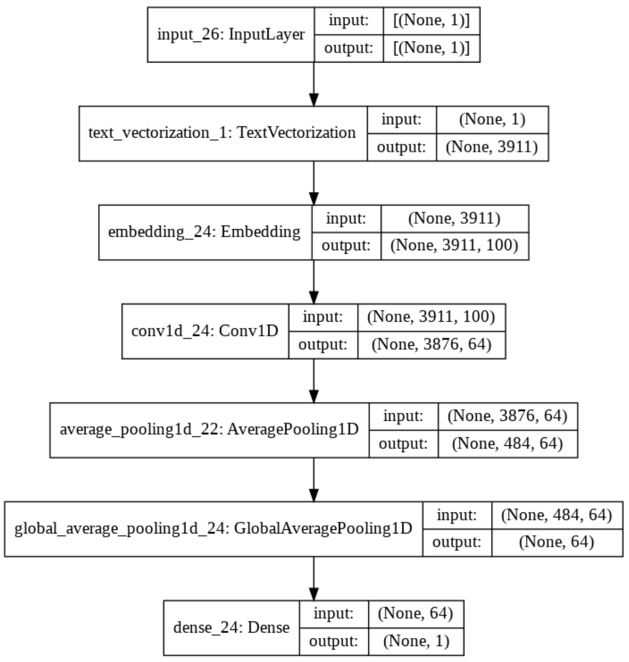
As already mentioned, our work concerning the task of identifying users prone to spread hate speech on Twitter by analysing their last 200 tweets was experimentally tested on the dataset provided by the PAN 2021 competition organizers.

A similar author profiling task was organized last year [14], in which the participants had to identify authors prone to spread fake news based on their last 100 tweets. The winners were [15] and [16]. Their models obtained an overall accuracy of 0.77 on the provided test set. The approaches used by the winners were based on a SVM and n-grams and on an ensemble of different machine learning models. As reported in [14] the only approach based on CNNs was the one presented in [17], achieving an overall accuracy of 0.72.

some input text) in which keys are token indices for such a list and values are the tokens.

3. Proposed model

The architecture of the model is presented in Figure 1, in which the dimensions of inputs and outputs of each model layer are highlighted.



input_26: InputLayer | input: [(None, 1)] | output: [(None, 1)]
text_vectorization_1: TextVectorization | input: (None, 1) | output: (None, 3911)
embedding_24: Embedding | input: (None, 3911) | output: (None, 3911, 100)
conv1d_24: Conv1D | input: (None, 3911, 100) | output: (None, 3876, 64)
average_pooling1d_22: AveragePooling1D | input: (None, 3876, 64) | output: (None, 484, 64)
global_average_pooling1d_24: GlobalAveragePooling1D | input: (None, 484, 64) | output: (None, 64)
dense_24: Dense | input: (None, 64) | output: (None, 1)

Figure 1: Model architecture. Numbers in brackets indicate tensor dimensions; $None$ stands for the batch dimension not yet known before running the model. Layers as depicted on our Google Colab Notebook.

Each layer of the network is chosen taking into account the works presented in Section 2 as well as those cited below and the hints gained from our extensive tests conducted over the training set provided. Indeed, many experiments were attempted to determine the appropriate hyperparameter values and network architecture. In what follows we present each layer of the network and the choosen hyperparameter values. Before discussing the network architecture, it is important to bear in mind that each set of the dataset (training and test per language) is made of XML files—each XML is related to a single author—containing 200 tweets of the author. In addition, a ground truth file containing the labels 0 and 1 matched to each XML file is provided for the training set. For handling these files and before training, our system organizes these XML files into two folders (i.e., 0 and 1) while reading the ground truth file. Then each sample (i.e., a single XML file) is read by the model for training or test, depending on the number of fold validation. These function for reading samples is accomplished by the first layer (namely, *InputLayer*).

3.1. Text vectorization

The first layer of our model reads the text in the XML files and apply our custom preprocessing function to split n-grams. In what follows, we refer to an n-gram as a sequence of characters. This sequence of characters is determined looking at the space before and after the sequence considered (i.e., the n-grams are splitted from the input text in correspondence of spaces). Then we build a dictionary where the keys are integer numbers and the values are the n-grams from the training set. While applying this tokenization based on spaces to the English dataset, we likely obtain n-grams that correspond to traditional tokens, or syntactic words. It is not the case when applied to the Spanish language. Hence, being n-gram a broader term as defined above, we prefer saying n-grams instead of tokens. Since the classification of HSSs is approached as an author profiling task, we decided to keep punctuation and capitalization to maintain a certain amount of stylistic information in our dictionary entries. Specifically and as an example, when splitting text, we associate two different dictionary entries to the word *Hello* and to *hello* or to *Hello!*. The hyperparameters characterizing this layer are described below.

- *Standardize.* It is the preprocessing function applied to the text before proceeding with its vectorization. In our case, this function, in addition to removing tabulations and newline characters, substitutes the occurrences of the CDATA tag with a space followed by a *minus than* sign, adds a space between the closing of one tag and the next, and then split each n-gram at each space;
- *Max tokens.* This parameter refers to the dictionary size. To get this value we simply count the numbers of different n-grams resulting from our preprocessing step. It is worth noting that our dictionary size is developed scanning both the Spanish and the English training sets;
- *Output mode.* This parameter is the type of token index returned by the vectorization function. We used the INT type, so that every word is mapped to a positive integer number;
- *Sequence length.* Although each XML document contains 200 tweets, the size in terms of produced n-grams is different for each sample because of the different length of each tweet. For this reason, we decided to consider the longest sample of the training set as size value, padding the shorter documents. As shown in Figures 1 and 2, this size is 3,911. As mentioned, padding is used for documents with a resulting number of token indices less than 3,911. Eventually, longer documents in the test set would be cut at this value.

The resulting output of this layer is a sequence of 3,911 positive integers corresponding to the dictionary keys of the n-grams of the XML document considered. Some random examples of *value* $\rightarrow$ *key* pairs in our dictionary are shown below.

...

rock $\rightarrow 210$

...

Hi! $\rightarrow 2315$

...

pregunta $\rightarrow 1508$

...

3.2. Embedding

This layer takes as input a tensor of 3,911 integer numbers generated as described in the previous subsection. Each integer value of this tensor is mapped to a 100-dimension word embedding tensor. In this way, each integer from the previous layer is mapped to a single tensor consisting of 100 floating point values. A notable difference with the previous layer is that the 100 coordinate values of each tensor is updated at each optimization step while training the model. More precisely, we trained and tested multiple models as the word embedding space varies from 2 to 800 dimensions, as also discussed in a similar Twitter text classification problem [18]. The best performances over different tests on a 5-fold cross validation were obtained with a 100-dimension embedding space.

3.3. Convolution

In our model a single 1D-convolution layer is implemented. This layer consists of 64 filters of size 36. The layer then performs convolution on 36-ngram windows with stride value of 1 (i.e., after each convolution, the convolutional filter is shifted of one word embedding tensor). For this layer no padding is added and ReLu [19, 20] is used as activation function on the output values. Number of filters and filters size (i.e., the two main parameters of this layer) are of paramount importance for the global performance of the model. Indeed, the filter size determines the size of the windows over the text of the input sample provided. In this way, we observed that a filter of size 36 generally gets n-grams from 3–4 different tweets each time. Similarly, the number of filters used (i.e., 64) determines the number of different feature maps relevant for the classification task. Both parameters are determined after extensive experiments conducted over the training set on many 5-fold cross validation runs. To fine-tune these two hyperparameters, we performed a binary search [21, 22] for both, looking in the range values 1–1,024. We discovered that a number of filters greater than 256 increases the overfitting of the model while a filter size greater than 1,024 does not allow the model to reach an accuracy of 1.0 not even on the train fold considered.

3.4. Average and global average pooling

The average pooling layer [23] downsamples the input representation by taking the average value over the window defined by a pool size parameter. The window is shifted by strides. As an example, consider a single dimension array X = [1.0, 2.0, 3.0, 4.0, 5.0]. Defining a 1D-average pooling layer having pool size of 2 and stride of 1 and providing X as input to such a layer, the array Y=[1.5, 2.5, 3.5, 4.5] is returned. In this case too, in the attempt of finding the best value for the hyperparameters of this layer, we performed a binary search and found an optimimum value of 8 for the pool size and 1 for the stride. The pool size of 8 represents the number of averaged values outputted from the convolution layer at each step. We suppose that the optimum of 1 as stride value might be maybe due to our tokenization choices.

A final 1D-Global Average Pooling layer is similar to the previously described average pooling one. In this case, it is not the average value over a window of the pool size defined that is returned as output but, instead, a global average along the first dimension from the previous layer outputs. Looking at the Figure 1, the output of AveragePooling1D layer is made of 484x64 elements. The global average value is calculated along the first dimension of size 484, in fact reducing by one the dimension of the input tensor. As an example consider the following matrix X.

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix}$$

Providing X as input to a 1D-Global Average Pooling layer returns as output the following array Y, where each y_i is calculated averaging all the values along the i-th column of the matrix X.

$$Y = \begin{bmatrix} y_1 & y_2 & y_3 & \dots & y_n \end{bmatrix}$$

3.5. Dense

The Global Average Pooling 1D layer is fully-connected to the last layer which is a single dense unit output. The layer is followed by a simple linear activation (e.g., $a(x) = x$). The final output is a single float value. Positive values are considered as HSSs and negative ones as nHSSs. A threshold of 0.0 is set to determine the accuracy of the model in predicting the label of the sample provided.

3.6. Model training

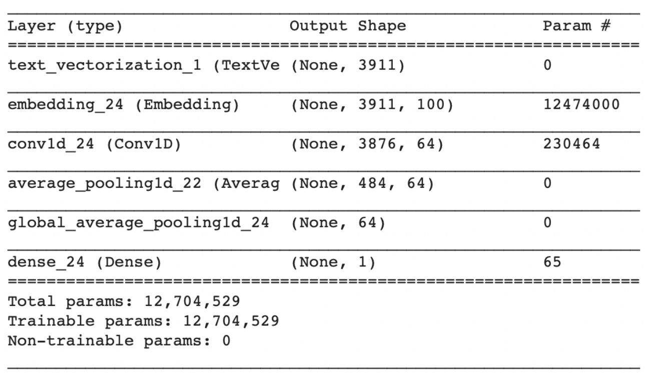
The values assigned to the various hyperparameters were originally set taking into account many of the decisions adopted in the studies conducted in [24, 25] and subsequently fine-tuned to improve the accuracy achieved by our model. To initialize the weights of the model we used a Glorot uniform initializer [26]. The model is compiled with a binary cross entropy loss function; this function calculates loss with respect to two classes (i.e., 0 and 1) as defined in 1.

$$Loss_{BCE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n \times \log\left(h_\theta(x_n)\right) + (1 - y_n) \times \log\left(1 - h_\theta(x_n)\right)] \tag{1}$$

where:

- N is the number of training examples;
- y_n is the target label for the training sample n;
- x_n is the input sample n;
- h_θ is the neural network model with weights θ.

Optimization is performed with an Adamic optimizer [27] after giving each batch of data as input. We performed a binary search for finding the optimal batch size. The model achieved the best overall accuracy with a batch size of 2. The model architecture is depicted in Figure 2, where the number of the various network hyperparameters are provided.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
text_vectorization_1 (TextVe (None, 3911)              0
_________________________________________________________________
embedding_24 (Embedding)     (None, 3911, 100)         12474000
_________________________________________________________________
conv1d_24 (Conv1D)           (None, 3876, 64)          230464
_________________________________________________________________
average_pooling1d_22 (Averag (None, 484, 64)           0
_________________________________________________________________
global_average_pooling1d_24  (None, 64)                0
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 65
=================================================================
Total params: 12,704,529
Trainable params: 12,704,529
Non-trainable params: 0
_________________________________________________________________
```

Figure 2: Model representation showing the number of parameters involved at each layer. Such a small number of parameters allows low computational load for training and testing. Figure as depicted on our Google Colab notebook.

	Training Set	**Test Set**
English | class 0	100	50
English | class 1	100	50
Spanish | class 0	100	50
Spanish | class 1	100	50

Table 1
Dataset summary showing the number of samples (i.e., authors) for each set, language and class.

4. Experimental evaluation and results

In this section we report the results obtained by our model during evaluation on the 5-fold cross validation on the training set. Then, we report the results of the trained model on the test set.

4.1. Experiments

Table 1 reports the number of samples within each set. Each sample in all the sets is an XML file whose name corresponds to the author id. Each XML file contains 200 tweets of the considered author. Considering that we have 200 tweets per author (the number of authors is shown in Table 1), the whole dataset consists of 120,000 tweets.

Task organizers invited participants to deploy their models on TIRA[28]. As communicated by email by the task organizers, TIRA has been experiencing technical issues. Therefore, our model was developed and tested as a Jupyter Notebook in Google Colab using TensorFlow. The complete source code is publicly available and reusable.[2]

To validate our model and fine-tune its hyperparameters, we ran 5-fold cross validations at each test performed. We considered the full training set made by the union of both language sets, then we shuffled this multilanguage training set and used it for the cross validations. Then we split 80–20 for each of the fold. Specifically, the first fold was made using the first 80% of the samples for training and the remaining 20% for validation. The remaining folds were made as

[2]Model notebook: https://colab.research.google.com/drive/1hUwn_uk0YPC6Tpo3MK1gDVGPQxmzPh_E.

1-Fold	**2-Fold**	**3-Fold**	**4-Fold**	**5-Fold**
80%T - 20%V	60%T - 20%V - 20%T	40%T - 20%V - 40%T	20%T - 20%V - 60%T	20%V - 80%T

Table 2
5-fold splitting applied on the complete multilingual training set. *T* indicates that this percentage is used for training and *V* for validation on this fold.

	Fold Nr.						
	1	2	3	4	5	Avg.	Dev.
Accuracy	0.6625	0.7000	0.6750	0.8000	0.6875	0.7050	0.0491
Loss	0.6097	0.7070	0.7771	0.5074	0.6234	0.6449	0.0916

Table 3
Results achieved by the model on a 5-fold cross validation on the complete multilingual training set (i.e., Spanish and English data). Both loss and accuracy are computed for the validation set used at the fold indicated on the upper row. In the last two columns we report the values of the arithmetic mean and the standard deviation over the 5 folds.

reported in Table 2 with the order of the percentage of samples taken for both sets (train and validation) from the complete training set.

4.2. Results

In Table 3 the results obtained adopting a 5-fold cross validation on the complete multilingual training dataset are reported. The 5-folds were made as explained in the previous subsection. Table 3 reports accuracy and loss values achieved on the validation set used at each fold, together with the arithmetic mean and standard deviation. For each fold we trained the model for 15 epochs, then we reported the higher accuracy and the related loss over the 15 epochs of training with respect to the validation set used at the fold indicated in the upper row. As can be noted, some splits achieved a better performance and this could be due to a higher level of similarity between the considered train and validation sets.

Finally, as reported in the PAN website, our model achieved an accuracy of 0.73 on the English test set and of 0.85 on the Spanish test set.[3] Considering these results, the overall accuracy (i.e., the arithmetic mean of the accuracy achieved per language) is 0.79.

5. Future works

In future developments, it would be interesting to analyze the behaviour of our model and the output of each layer. Furthermore, another point deserving to be investigated concerns the reasons behind the fact that the tested models are better at identifying HSSs in Spanish than in English. In fact, in each performed test, our models performed better on the Spanish set, even using a non-deep model (i.e., a Naïve Bayesian model using the same preprocessing function). Perhaps, conducting a qualitative analysis of the dataset tweets could be beneficial for understanding why profiling HSSs in Spanish achieves a higher accuracy than in English.

[3]Pan 2021 task results: https://pan.webis.de/clef21/pan21-web/author-profiling.html#results.

This investigation could help us shed some light on this accuracy difference in the two datasets–which concerned also the last year PAN author profiling task—but also on how our model works looking at both correct and wrong predictions. Another direction to improve accuracy in profiling HSSs could be to add more complexity to the model, maybe using some additional layers. Given the dimension of the dataset provided some techniques of data augmentation could be also applied. Finally, some investigation on the content of each tweet could guide us in applying some techniques to remove noise (i.e., not relevant features) from the input samples.

6. Conclusion

In this paper, we described the submitted model for the Profiling HSSs on Twitter task at PAN 2021. It consists of a CNN based on a single convolutional layer. To get a more accurate evaluation of the model performance, we run several 5-fold cross validation for each different hyperparameter configuration. In fact, several binary searches are conducted to fine tuning the hyperparameters involved during the training of our proposed model. After finding the model achieving the highest accuracy during our cross validation tests, we trained such a model on the entire training set to submit our predictions on the test set. The model proved to maintain the good accuracy achieved on the cross-validation process also when tested on the test set. Overall, as announced by the organizers, our software—achieving an average accuracy of 0.79 (0.73 on the English test set and 0.85 on the Spanish test set)—ranked first in the PAN 2021 HSSs profiling task. Our model, developed in TensorFlow, is publicly available as a Jupyter Notebook on Google Colab.

Acknowledgments

We would like to thank anonymous reviewers for their comments and suggestions that have helped to improve the presentation of the paper.

CRediT Authorship Contribution Statement

Marco Siino: Conceptualization, Formal analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing - Original draft, Writing - review & editing. **Elisa Di Nuovo:** Formal analysis, Investigation, Writing - Original draft, Writing - review & editing. **Ilenia Tinnirello:** Supervision, Writing - review & editing. **Marco La Cascia:** Supervision, Writing - review & editing.

References

[1] J. Bevendorff, B. Chulvi, G. L. D. L. P. Sarracén, M. Kestemont, E. Manjavacas, I. Markov, M. Mayerl, M. Potthast, F. Rangel, P. Rosso, E. Stamatatos, B. Stein, M. Wiegmann, M. Wolska, , E. Zangerle, Overview of PAN 2021: Authorship Verification,Profiling Hate Speech

Spreaders on Twitter,and Style Change Detection, in: 12th International Conference of the CLEF Association (CLEF 2021), Springer, 2021, p. 1.

[2] F. Rangel, G. L. D. L. P. Sarracén, B. Chulvi, E. Fersini, P. Rosso, Profiling Hate Speech Spreaders on Twitter Task at PAN 2021, in: CLEF 2021 Labs and Workshops, Notebook Papers, CEUR-WS.org, 2021, p. 1.

[3] F. Rangel, M. A. Chulvi, G. L. De La Pena, E. Fersini, P. Rosso, Profiling Hate Speech Spreaders on Twitter [Data set], https://zenodo.org/record/4603578, 2021.

[4] M. Thangaraj, M. Sivakami, Text classification techniques: A literature review., Interdisciplinary Journal of Information, Knowledge & Management 13 (2018).

[5] B. Altınel, M. C. Ganiz, Semantic text classification: A survey of past and recent advances, Information Processing & Management 54 (2018) 1129–1153.

[6] R. Oshikawa, J. Qian, W. Y. Wang, A survey on natural language processing for fake news detection, arXiv preprint arXiv:1811.00770 (2018).

[7] H. Wu, Y. Liu, J. Wang, Review of text classification methods on deep learning, CMC-COMPUTERS MATERIALS & CONTINUA 63 (2020) 1309–1321.

[8] S. Hashida, K. Tamura, T. Sakai, Classifying tweets using convolutional neural networks with multi-channel distributed representation, IAENG International Journal of Computer Science 46 (2019) 68–75.

[9] Z. Wang, Z. Qu, Research on web text classification algorithm based on improved cnn and svm, in: 2017 IEEE 17th International Conference on Communication Technology (ICCT), IEEE, 2017, pp. 1958–1961.

[10] Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint arXiv:1408.5882 (2014).

[11] G. E. Hinton, et al., Learning distributed representations of concepts, in: Proceedings of the eighth annual conference of the cognitive science society, volume 1, Amherst, MA, 1986, p. 12.

[12] S. Wang, W. Zhou, C. Jiang, A survey of word embeddings based on deep learning, Computing 102 (2020) 717–740.

[13] A. Severyn, A. Moschitti, Twitter sentiment analysis with deep convolutional neural networks, in: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2015, pp. 959–962.

[14] F. Rangel, A. Giachanou, B. Ghanem, P. Rosso, Overview of the 8th author profiling task at pan 2020: Profiling fake news spreaders on twitter, in: CLEF, 2020, p. 1.

[15] J. Pizarro, Using n-grams to detect fake news spreaders on twitter, in: CLEF, 2020, p. 1.

[16] J. Buda, F. Bolonyai, An ensemble model using n-grams and statistical features to identify fake news spreaders on twitter, in: CLEF, 2020, p. 1.

[17] M. P. Chilet L., Profiling fake news spreaders on twitter, in: CLEF, 2020, p. 0.

[18] X. Yang, C. Macdonald, I. Ounis, Using word embeddings in twitter election classification, Information Retrieval Journal 21 (2018) 183–207.

[19] K. Fukushima, Visual feature extraction by a multilayered network of analog threshold elements, IEEE Transactions on Systems Science and Cybernetics 5 (1969) 322–333.

[20] K. Fukushima, S. Miyake, Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition, in: Competition and cooperation in neural nets, Springer, 1982, pp. 267–285.

[21] L. F. Williams Jr, A modification to the half-interval search (binary search) method, in: Proceedings of the 14th annual Southeast regional conference, 1976, pp. 95–101.
[22] D. Knuth, The Art Of Computer Programming, vol. 3: Sorting And Searching, Addison-Wesley, 1973.
[23] TensorFlow, AveragePooling1D layer, https://keras.io/api/layers/pooling_layers/average_pooling1d/, 2021.
[24] Y. Zhang, B. Wallace, A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification, arXiv preprint arXiv:1510.03820 (2015).
[25] A. Jacovi, O. S. Shalom, Y. Goldberg, Understanding convolutional neural networks for text classification, arXiv preprint arXiv:1809.08037 (2018).
[26] Keras, Layer weight initializers, https://keras.io/api/layers/initializers/, 2021.
[27] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017. `arXiv:1412.6980`.
[28] M. Potthast, T. Gollub, M. Wiegmann, B. Stein, TIRA Integrated Research Architecture, in: N. Ferro, C. Peters (Eds.), Information Retrieval Evaluation in a Changing World, The Information Retrieval Series, Springer, Berlin Heidelberg New York, 2019, p. 1. doi:`10.1007/978-3-030-22948-1\_5`.

A. Online Resources

The source code of our model is available via

- GitHub,
- Google Colab.