IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 5, 2021, accepted November 30, 2021, date of publication December 3, 2021,
date of current version December 15, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3132502

Context-Aware Misinformation Detection: A Benchmark of Deep Learning Architectures Using Word Embeddings

VLAD-IULIAN ILIE[1], CIPRIAN-OCTAVIAN TRUICĂ[1], ELENA-SIMONA APOSTOL[1], AND ADRIAN PASCHKE[2]

[1]Computer Science and Engineering Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, 060042 Bucharest, Romania
[2]Fraunhofer Institute for Open Communication Systems (FOKUS), 10589 Berlin, Germany

Corresponding authors: Ciprian-Octavian Truică (ciprian.truica@upb.ro) and Elena-Simona Apostol (elena.apostol@upb.ro)

This work was supported in part by the German Academic Exchange Service (DAAD) through the Project "AWAKEN: content-Aware and netWork-Aware faKE News mitigation" under Grant 91809005 and Project "Deep-Learning Anomaly Detection for Human and Automated Users Behavior" under Grant 91809358, in part by the German Federal Ministry of Education and Research (BMBF) Project "PANQURA-a technology platform for more information transparency in times of crisis" under Grant 03COV03F, and in part by the European Union Project "FAST-LISA-Fighting hAte Speech Through a Legal, ICT and Sociolinguistic Approach" under Grant 101049342.

ABSTRACT New mass media paradigms for information distribution have emerged with the digital age. With new digital-enabled mass media, the communication process is centered around the user, while multimedia content is the new identity of news. Thus, the media landscape has shifted from mass media to personalized social media. While this progress brings advantages, it also carries the risk of being detrimental to society through the emergence of misinformation (false or inaccurate information) and disinformation (intentionally spreading misinformation) in the form of fake news. Fake news is a tool used to manipulate public opinion on particular topics, distort public perceptions, and generate social unrest while lacking the rigor of traditional journalism. Driven by this current and real-world problem, in this paper, we train multiple Deep Learning architectures for multi-class classification and compare their performance in detecting the veracity of the news articles. To achieve accurate models in detecting misinformation, we employ a large dataset containing 100 000 news articles labeled with ten classes (one with real news and the rest with different types of fake news). We use two preprocessing techniques, i.e., one simple and another very aggressive, to clean the dataset. We also employ three word embeddings that preserve the word context, i.e., Word2Vec, FastText, and GloVe, pre-trained and trained on our dataset to vectorize the preprocessed dataset. For the misinformation task, we train a Logistic Regression as a baseline and compare its results with the performance of ten Deep Learning architectures. We obtain the best results using a Recurrent Convolutional Neural Network based architecture. The experimental results show that the models are highly dependable on text preprocessing and the word embedding employed.

INDEX TERMS Misinformation detection, deep learning, multi-class text classification, word embeddings, text preprocessing, benchmarking.

I. INTRODUCTION

The digital age has seen the emergence of new mass media paradigms for information distribution, substantially different from classical mass media. With these changes in mass media, the users become the center of communication, their interests being the core topic of news feeds, their views being targeted through news articles. Thus, the media landscape has shifted from mass media to personalized social media. Along with the advantages this progress brings, it aggravates the risk of misinformation [42] with potentially detrimental consequences to society [40], by facilitating the spread of misinformation (false and inaccurate information) and disinformation (intentionally spreading misinformation) in the form of fake news (which, for example, appears to have influenced the Brexit referendum [3] and the 2016 US presidential

The associate editor coordinating the review of this manuscript and approving it for publication was Md. Moinul Hossain.

election [5]). However, current trends and the accessibility of technology render this proneness even more potentially damaging than it has been historically, thus having dire consequences to the community [50] and being a major challenge to democratic values (e.g., public polarization regarding elections, vaccination, etc.).

Fake news consists of news stories that are intentionally and verifiable false, and which could mislead readers [44] by presenting alleged, imaginary facts about social, health, economic, and political subjects of interest [46]. Another perspective is to directly treat false news as fake news, which includes fabrications, hoaxes, and satires [41], [43].

Untruthful news stories have been around for a long time, and every technological advancement, from the telegraph to the social media, has provided new ways to create deception. Historically, the mass media industry proved to use manipulation and follow their own agenda by adding bias to important events, e.g., articles published in the New York Journal in 1897 that played a prominent role in the start of the war between the USA and Spain [14]. Currently, it is widely believed that fake news plays a significant role in shaping public opinion [14] and even the perception on current health treads, e.g., SAR2-COVID19 [6] misinformation. International and national organization[1] and governments[2] have spoken against fake news, the most current being the speech given by the president of the European Commission, Ursula von der Leyen.[3] Thus, decisions based on facts and evidence are compromised by fake news and the consequences sometimes are irreversible [21].

Fake news detection presents many challenges, as the information presented in the news articles has the explicit purpose of deceiving the reader. There is interdisciplinary research on this subject, which tries to analyze the matter from different perspectives. In forensic psychology, the focus is the human vulnerability to fake news and the deceptive styles used by disinformation propagators [52]. In computer science research, the main objective is to find out methods of detecting fake news from multiple features like the knowledge bases, style, how it propagates, and its credibility [52]. There is a strong accent in modern times on the speed and volume of information. This has affected even the way that the news is produced and consumed, by increasing the amount of news articles as to keep up to the demand. However, this comes with a drawback. The veracity of the information is no longer a requirement when it is produced, as we live in an age of ''post-truth'' in which objective facts are less influential in shaping public opinion than emotion and personal beliefs [20]. The consumer is now fed news that are not necessarily based on true events but mostly aims to manipulate the public opinion, by making the public accept biased or false beliefs. Current research found that having a pre-existing knowledge base against which to compare news is not sufficient, because fake news may cite true facts to support a non-factual claim. This mostly invalidates the assumption that if the news refers to true news and is similar to true news then it is true news as well [9].

In traditional news media, there is a psychological foundation of fake news, meaning that deceptive articles are targeting the vulnerabilities of the individual [43]. Consumers not only tend to believe that their perception of reality is the only accurate one, while those who disagree are irrational, but they also prefer information that supports their already established ideas and opinions. Furthermore, people tend to agree to information so that they will be socially accepted and will increase their self-esteem, even if that information is not based on actual facts [51]. Moreover, publishers tend to maximize their short-term utility, in which they maximize their profits, while the consumers tend to maximize the satisfaction they receive by supporting their prior opinions and social image.

Recently, it was addressed and highlighted the need to have an automatic tool of detecting deceptive news and the challenges that the development of the tool presents. As the web data is unstructured, it demands flexible methods to verify its truthfulness, unlike structured datasets or well-defined topics that require a certain domain of the language [9]. However, most tools for automatic detection of fake news consisted of classical Machine Learning algorithms, like Support Vector Machine (SVM) and Naive Bayesian models. These models have high accuracy for not very large, balanced datasets, but they have problems in building accurate models when the size of the dataset increases exponentially. Recent research points towards the importance of using Deep Learning classifiers in fake news detection. Also, by adding linguistic features, e.g., different types of embedding algorithms, the performance of the prediction task is increasing [16]. Thus, a benchmark of Deep Learning Architectures using different word embeddings and an in-depth analysis of the models' results for the task of Misinformation Detection is required.

In this paper, we propose such a Deep Learning based benchmark for Misinformation Detection that considers different text feature representations, such as general and specific word embeddings. The main contributions of this article are the following. Firstly, we perform an extensive benchmarking study on ten different Deep Learning architectures and a Logistic Regression architecture as a baseline. Secondly, we provide a comprehensive evaluation and analysis of the most popular context-aware word embeddings models, i.e., Word2Vec, FastText, and GloVe, both with pre-trained embeddings and trained on our corpus. Such a comparison of the effects of different embedding models is lacking in the current research. Although there are a few fake news detection models that use dynamic word embeddings methods, i.e., Bidirectional Encoder Representations from Transformers (BERT) [39] [30], they use such methods as pretrained language model, i.e., without a detailed evaluation and discussion of the effects of the training on the results. That is, BERT would not fit in our generic versus specific

[1] World Economic Forum: Digital Wildfires in a Hyperconnected World
[2] UK Parliament: Online Information and Fake News
[3] European Commision: Fighting disinformation

embeddings comparison and is out of the scope of this contribution, as it is a pre-trained model. Thirdly, we provide a detailed analysis of the impact of different text preprocessing techniques on various Deep Learning models. Our experimental results show that the models are highly dependable on text preprocessing and the word embedding employed. The experiments were conducted on a large real-world news dataset consisting of 100 000 news articles. Fourthly, we consider a multi-class fake news classification consisting of ten classes, e.g., Fake News, Satire, Extreme Bias, Hate News, etc. In comparison, other current solutions use either a binary approach, i.e., the news articles' veracity is either true or false, or levels of veracity. Finally, we propose a preprocessing and classification pipeline based on our findings.

In this paper, we identified some shortcomings in the current literature, and we try to address them by answering the following research questions:

(Q1) How do Machine and Deep Learning models perform on the task of misinformation detection?
(Q2) What is the influence of the dataset size in the task of misinformation detection?
(Q3) Do specific trained word embedding improve the performance of Machine Learning models than pretrained word embeddings on the task of misinformation?
(Q4) Do different preprocessing techniques improve the performance of Machine Learning models on the task of misinformation?

The main contributions of our work, which try to address the shortcoming in the current literature, are:

(1) change the scope from binary or levels of veracity to multi-class classification;
(2) analyzing and benchmarking multiple Deep Learning architectures;
(3) presenting an in-depth analysis regarding:
    (a) the effects of the size of the dataset on the misinformation detection task;
    (b) how simple and aggressive text preprocessing techniques influence the accuracy of Machine and Deep Learning algorithms on the misinformation detection task;
    (c) the use of specific and generic word embeddings for the multi-class classification task;
    (d) the efficiency of Machine and Deep Learning algorithms on the task of misinformation detection.

The paper is structured as follows. In Section II, we review the current state of the art in the field of fake news detection. We discuss the models and the algorithms used in our solution in Section III. We present the architecture of our solution and discuss the implementation decisions in Section IV. In Section V, we analyze and interpret the experimental results. In Section VI, we discuss our findings from the perspective of the dataset, preprocessing, word embeddings, and algorithms. Finally, in Section VII we summarize our findings and hint at future research directions.

II. RELATED WORK

In this section, we present some of the current work in fake news detection. There are many articles that research this subject, mainly centered around two perspectives: *the language perspective*, which refers to the actual content of the news (headline, author, and news content), and *the data mining perspective*, which refers to the social context of the news article and the stylometric features [43].

In the linguistic approach to deception detection, it was observed that although most liars carefully use their language to avoid being caught, language cues that are hard to monitor occur, such as frequencies and patterns of pronoun, conjunction, and negative emotion word usage [9]. When trying to learn these aspects, using the simplest method of word representation, bag-of-words, many aspects of context are lost with n-grams. One way to improve the bag-of-words method is to take into account the syntactic structure of the document. Semantic analysis can also provide additional information in deciding if facts are deceptive or not, which can be done by comparing a personal experience with true and objective facts. The idea behind this is that deceptive news articles will most likely contain some contradictions or omission of facts that are present in other articles. However useful this analysis is, it has been restricted by the amount of available data to compare the articles against.

Considering the data mining approach, researchers have recently focused on applying Machine Learning techniques to fake news detection. Also, another important research direction is centered on adding linguistic cues to this process.

In [16] is investigated the impact of using content-based features and Machine Learning algorithms for fake news detection. The authors propose a benchmarking study with different feature selection techniques, e.g., Mutual information(, Minimum Redundancy Maximum Relevance), and several classical Machine Learning classifiers, i.e., Naive Bayes, SVMs, Decision Trees, k-NNs, and two ensemble methods, i.e., AdaBoost, Bagging. To accurately detect deception, they combined the resulting feature sets and enhanced them with Word2Vec features, obtained using a pre-trained model. The authors claim that they also experimented with GloVe, but without noticing relevant differences in performance. We argue that this is not the case, as we demonstrate in Section V, where we consider both pre-trained and specific word embedding models. Another important observation from this study is that linguistic features could provide better performance to the fake news detection task than term-frequency features. Although this study offers interesting results on the impact of Machine Learning and content-based features on fake news detecting, it lacks the ability to deliver high-quality classification results on large datasets using Deep Learning architectures.

The impact of Hierarchical Attention Networks (HAN) on fake news detection is tackled in [34] via SADHAN model that introduces various latent aspect embeddings over the HAN (Self-Attention based Hierarchical Attention Network

BiLSTM) architecture. Thus, SADHAN is using the Attention mechanism to learn embeddings for different latent aspects of news articles, i.e., subject, author, and domain embeddings. The solution applies pre-trained GloVe embeddings that, unlike Word2Vec, relies on both local and global word statistics. To assess the trustworthiness of a claim from a news article, the authors also propose a technique to extract evidence snippets, using Attention weights, that supports or refutes the analyzed claim. Considering the presented results, the proposed model outperforms all the analyzed baseline architectures, i.e., CNN, BiLSTM, and two other Attention-based architectures, emphasizing the importance of aspect embeddings for the prediction of false news. An important discussion that is completely missing from this article is the utilized preprocessing mechanism and its impact on the overall performance. From our experiments, we deducted that the preprocessing step can have a big influence on the accuracy of the results. HAN-based models are also tested in [23]. These models are compared with CNN, LSTM, BiLSTM, and Convolutional-LSTM, on three datasets, i.e., a small ∼6K dataset, a medium ∼12K dataset, and a large ∼75K dataset. The models are initialized with pre-trained GloVe embeddings. HAN has the best performance on the small dataset, while Convolutional-HAN on the medium dataset. Both models are outperformed by BiLSTM and Convolutional-LSTM on the large dataset.

Convolutional Networks architecture is a popular Deep Learning-based model used in classifying news articles. Although a CNN network can be prone to over-fitting, a deeper CNN resolves this problem [22]. The work presented in [22] consists of a GloVe-enabled deep convolutional-based model, i.e., FNDNet. Their solution was compared with classical Machine Learning algorithms and several Deep Learning architectures, i.e., classical CNN with GloVe and LSTM with GloVe. Though the FNDNet results are promising in comparison with the other evaluated solutions, we have some concerns about the experimental setup for the tested networks, especially on the small epoch numbers for each neural network.

Another solution based on CNN architecture is presented in [27]. The authors proposed MCNN-TFW, a multiple-level convolutional neural network-based fake news detection system. The architecture is actually a CNN architecture combined with semantics features that are determined using a pre-trained Word2Vec model. To measure the word's sensitivity coefficient, the authors propose the weight of sensitive words (TFW) that is based on the formulas used in TF-IDF. Although some of the ideas presented in this paper are interesting, some improvements can be done, especially regarding the experiments. First of all, the used datasets are rather small, i.e., 4180 items and 6728 items, respectively. The use of deep networks makes sense in the context of large datasets, where traditional Machine Learning techniques are losing their efficiency. Secondly, in our opinion, it would have been interesting if the authors also compared their architecture with other word embeddings enabled deep-learning solutions. Instead, they compare it with LIWC (linguistic inquiry and word count) features, a CNN architecture without embeddings, and a join SVM classifier with RST (Rough Set Theory). It is proven that most of the lexicon-based approaches, LIWC in particular, have lower performance compared with Machine Learning algorithms [18].

Several current fake news detection approaches use advanced pre-trained Transformer models. BERT (Bidirectional Encoder Representations from Transformers) [10] is the most used Transformer to classify and detect Fake News. BERT is part of a new class of word embeddings methods, more specifically the dynamic one, and it is said that in many Natural Language Processing tasks, it outperforms the static embeddings methods. It uses an Attention-based mechanism, i.e., the Transformer encoder. Although some works employ large datasets and properly analyze the textual content [38], the majority of the Fake News solutions use BERT on relatively small datasets, and without a clear explanation of the results [24], [30].

One model for fake news detection build on BERT is described in [30]. The authors propose a two-stage model, both stages having similar BERT-based architectures. In the first stage, all the statements are classified, using two labels: false or true, resulting in a temporary label for each statement. The second stage uses some extra information, such as the speaker's name, job, etc., that are modeled as features. This results in a number of fine-grained labels. For the final label, the system considers both coarse-grained and fine-grained labels. The experiments are done using the LIAR dataset from PolitiFact,[4] which consists of ∼12K very short news statements. Being a relatively small dataset with little information, the proposed model and the compared ones have trouble learning correctly. This is why the maximum accuracy obtained in [30] is around 40%. Whereas in [24], the BERT-based architectures use a relatively big balanced dataset, i.e., ∼45K items, and obtain very good results. The solution presented in [24] has a hybrid architecture connecting BERT word embeddings with RNN. The RNN network is used to obtain document embeddings. The major issue with this solution is that the results are not explained or discussed, and there is no comparison with other types of word embedding and deep networks. In [38], BERT is used as a baseline on a large corpus consisting of ∼103K documents. This manuscript is focusing on the style of writing, i.e., the form of text rather than its meaning. The authors proposed two new classifiers, i.e., a BiLSTM based neural network and a stylometric classifier. These classifiers are compared with two baseline models, i.e., bag of words and BERT.

Recently other transformers were used for the task of Fake News detection. RoBERTa (A Robustly Optimized BERT pre-training Approach) [31] is a robustly optimized method that improves BERT's language masking strategy. A weak social supervision RoBERTa-based model for Fake News detection from limited annotated data is proposed

[4] LIAR dataset from PolitiFact

in [45]. Another Transformer used for Fake News detection is XLNet [48], an extension of BERT. In [13], the XLNet model is combined with topic information and applied on a relatively small annotated dataset (∼10K).

An in-depth benchmark on the subject of Fake News detection is done in [23]. The authors consider only binary classification on three datasets, i.e., two smaller corpora, Liar (∼12K items) and Fake or Real (∼6K items), and a larger corpus (Combined Corpus) consisting of ∼75K items. For testing, the article presents traditional Machine Learning, Deep Learning, and five transformer-based models. The neural network Deep Learning models are initialized with pre-trained GloVe embeddings. The overall highest accuracy values are obtained on the largest dataset. The conclusion is that Naıve Bayes (with bi-gram) performance, the best of the classical Machine Learning solutions tested, is on average 0.2 lower than the best tested Deep Learning networks, i.e., 0.95 for Bi-LSTM and C-LSTM, that combines a convolutional layer with LSTM. The best performance was obtained by RoBERTa pre-trained model, i.e., 0.96 on Combined Corpus, but it is not made clear if they used or not fine-tuning.

Taking into consideration the presented related work, we further exploited the use of different word embedding techniques and Deep Learning architectures to detect news with harmful content. Finally, we can stipulate that most of the current solutions do not clearly present the pre-processing step, although it is an important step that can alter the fake news detection accuracy. As far as we know, none explore different variants of pre-processing techniques and their impact on the fake news detection task. Likewise, all the studied current solutions use pre-trained general embeddings and not specific word embeddings done on the news corpus.

III. METHODOLOGY

In this section, we discuss the algorithms, models, and architectures used by our solution from a theoretical perspective. First, we start with a discussion of text prepossessing and cleaning. Second, we present the mathematical formulation of all the Machine and Deep Learning classification algorithms employed for Misinformation Detection. Finally, we discuss the Evaluation metrics to correctly do an assessment of the quality of the classification algorithms.

A. TEXT PREPROCESSING

Data preprocessing is an important step in Machine Learning algorithms because it can greatly influence its performance. It refers to filtering out the noise or unwanted features and improving the features both in quantity and in quality. Text preprocessing is used to remove known common words or stopwords. Stopwords do not bring any information gain or add any meaning from the perspective of weighting the terms or improving the textual context. Thus, these words acts as noise and need to be removed. Moreover, as in the case of stopwords, punctuation bring no information gain or improve the news articles context and is usually removed.

Another text preprocessing technique is lemmatization, i.e., to the process of extracting the words lemma. The lemma is the root of the word, i.e., the base of the inflections of that word. Lemmatization manages to minimize the vocabulary and capture the similar meaning of some terms, even if their inflection differs.

B. WORD EMBEDDING

1) Word2Vec

Word2Vec [32] is a shallow neural network model that is used to produce word embeddings. The output vectors are positioned in the vector space such that common contexts are in close proximity to each other. This allows the understanding of the meaning of a word by looking at its context. Basically, it tries to maximize the probability of the target word based on the input word (Equation (1)), where w_j is the target word, w_I is the input word, v'_{w_j} is the word embedding of the target word, v_{w_I} is the word representation of the input word.

$$p(w_j|w_I) = \frac{exp(v'^T_{w_j} v_{w_I})}{\sum_{j'=1}^{V} exp(v'^T_{w_{j'}} v_{w_I})} \tag{1}$$

Word2Vec provides two models used in word embeddings: *the continuous bag-of-words* (CBOW) and *the Skip-Gram* models. The idea behind both models is based on the Distributional Hypothesis which states that different words that appear in similar contexts have similar meaning.

The CBOW model is based upon unigrams which is an n-gram (continuous sequence of n terms) comprised of one term. The sequence captures the context of the word by averaging the context of each word in the selection window. This allows Word2Vec to predict the target word based on the input context sequence and better results than Skip-Gram for the frequent words in the dataset [32].

The Skip-Gram model uses skip-grams in defining word context. This allows the capture of the context of more distanced words in the sequence, and it is used to predict the context based on an input word, opposite to what the CBOW model does. This model should have better results than CBOW for rare words in the dataset [32].

2) FastText

The FastText [4] model is a set of improvements on top of the Word2Vec model [33], using both CBOW and Skip-Gram. The first improvement refers to the drawbacks brought on by ignoring the position of the words in the context window. The proposed solution learns the representations of word positions and afterwards uses them to weight the context words. Equation (2) shows how the context vector transforms, considering P is the set of positions $[-c, \ldots, -1, 1, \ldots, c]$, d_p is the position weight vector, u_{t+p} is the context word at position p and $\odot$ is the pointwise multiplication operation.

$$v_C = \sum_{p \epsilon P} (d_p \odot u_{t+p}) \tag{2}$$

The second improvement refers to adding support for n-grams in the CBOW model, which initially used only unigrams. This also improves the information brought by word order, and removes cluttering and noise caused by unwanted words by creating the n-gram tokens based on merging only words with high mutual information [33].

The last improvement takes into account the information brought by the word internal structure. Instead of simply taking into account just the word representation, FastText also extracts information from subwords, or character n-grams. This allows representing words that are out of the training dataset. Equation (3) presents the context vector, where N is the set of character n-grams, $|N|$ is the number of character n-grams, v_w is the word context using the above mentioned methods and x_n is the vector representation of the character n-gram.

$$v_C = v_w + \frac{1}{|N|}\sum_{n\epsilon N} x_n \tag{3}$$

3) GloVe

GloVe [36] is an unsupervised Machine Learning algorithm that generates word embeddings. It does so by first constructing a term co-occurence matrix, that contains the frequency with which each word appears in context with another word. The number of contexts is combinatorial so the dimensions of this matrix are very large. Equation (4) presents the co-occurence ratio probabilities employed by this embedding, where v_{w_i}, v_{w_j}, v_{w_k} are the representations of words w_i, w_j, w_k, and $P_{w_iw_k}$, $P_{w_jw_k}$ are the probabilities that word w_i appears in context with word w_k and word w_j appears in context with word w_k, respectively.

$$F(v_{w_i}, v_{w_j}, \tilde{v}_{w_k}) = \frac{P_{w_iw_k}}{P_{w_jw_k}} \tag{4}$$

As F should encode information from the vector space of the words, and that vectors spaces are linear, for keeping said linearity, Equation (4) becomes Equation (5).

$$F((v_{w_i} - v_{w_j})^T \tilde{v}_{w_k}) = \frac{P_{w_iw_k}}{P_{w_jw_k}} \tag{5}$$

In word co-occurence matrices the context word can be exchanged with the actual word, seeing as there is no exact distinction between them. This generated the requirement that F should be a homomorphism between groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$ (Equation (6)).

$$F((v_{w_i} - v_{w_j})^T \tilde{v}_{w_k}) = \frac{F(v_{w_i}^T \tilde{v}_{w_k})}{F(v_{w_j}^T \tilde{v}_{w_k})} \tag{6}$$

Equation (7) is obtained by replacing Equation (5) in Equation (6), where $X_{w_iw_k}$ is the frequency word w_i appears with word w_k and X_{w_i} is the number of times the word w_i appears.

$$F(v_{w_i}^T \tilde{v}_{w_k}) = P_{w_iw_k} = \frac{X_{w_iw_k}}{X_{w_i}} \tag{7}$$

If F is an exponential function, we can extract the logarithm (Equation 7). As $log(X_{w_i})$ is independent of w_k, it can transform in a bias b_{w_i} for v_{w_i} and adding a bias $\tilde{b}_{w_k}$ for $\tilde{v_{w_k}}$ restores the symmetry (Equation (9)).

$$v_{w_i}^T \tilde{v}_{w_k} = log(P_{w_iw_k}) = log(X_{w_iw_k}) - log(X_{w_i}) \tag{8}$$

$$v_{w_i}^T \tilde{v}_{w_k} + b_{w_i} + \tilde{b}_{w_k} = log(X_{w_iw_k}) \tag{9}$$

This form is ill-defined because the *log* diverges for zero arguments. Also, this model assigns equal weights to all co-occurrences, which can bring noise. That is why GloVe proposes a weighted least squares regression model with a loss function. Equation (10) presents the loss function, where V is the size of the vocabulary and f is a weighting function defined by:

- $f(0) = 0$ so that the log converges with x converging to 0
- $f(x)$ non-decreasing so rare co-occurences are not overweighted
- $f(x)$ small for large values of x so frequent co-occurences are not overweighted

$$J = \sum_{i,j=1}^{V} f(X_{w_iw_j})(v_{w_i}^T \tilde{v}_{w_j} + b_{w_i} + \tilde{b}_{w_j} - log(X_{w_iw_j}))^2 \tag{10}$$

Thus, GloVe works by extracting information from the *log* of the co-occurence probability ratio, meaning the difference of the *log* of the co-occurence probabilities. Because of the symmetry, it actually extracts information from the difference of word vectors. That is why the model shows good results for word analogies or finding the same meaning for synonyms in the same contexts.

C. MACHINE AND DEEP LEARNING ARCHITECTURES

1) LOGISTIC REGRESSION

Logistic Regression models a relationship between predictor variables and a categorical response variable. The model estimates the probability of an independent variable to fall into a certain level of the categorical response given a set of predictors. Given a set of classes $Y = \{y_k | \kappa = \overline{1,k}\}$ and a document dataset $X = \{x_i | i = \overline{1,n}\}$, the model computes the probability of a class y_κ given a document d_i as $p(y = y_\kappa | x = x_i)$. The probability is the sigmoid function $(\sigma_s(z) = \frac{1}{1+e^{-z}})$ that maps documents to classes in order to determine the parameters of the weight vector $\boldsymbol{w}$ and the bias b that fit the regression line. Equation (11) presents the Logistic Regression probability.

$$p(y = y_\kappa | x = x_i) = \sigma_s(\boldsymbol{w}x_i + b) = \frac{1}{1 + e^{-(\boldsymbol{w}x_i + b)}} \tag{11}$$

To estimate the classes $\hat{y}_i = argmax_{y_\kappa \in C} p(y = y_\kappa | x = x_i)$ and to determine the parameters $\boldsymbol{w}$ that give the best results, the algorithm maximises the log likelihood using gradient descent [49]. The log likelihood function guarantees that the gradient descent algorithm can converge to the

global minimum. Equation (12) presents the log likelihood function $l(\boldsymbol{w})$, where y_i is the real class of document x_i.

$$l(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i \log(\sigma_s(\boldsymbol{w}x_i)) - (1 - y_i) \log(1 - \sigma_s(\boldsymbol{w}x_i))) \tag{12}$$

2) RECURRENT NEURAL NETWORKS

Recurrent neural networks (RNN) are a class of neural networks that are used in sequence processing, because of their ability to cache the previous outputs and add their information to the current inputs. This means that the model can process sequences of any length, while the model size does not change. This presents some unique architectural opportunities regarding the number of inputs and outputs:

- one to one - equivalent to a regular neural network
- one to many - used in image captioning
- many to one - used in sentiment analysis
- many to many - used in Machine Translation

The input x_t at each time step t serves as input to the RNN alongside the activation from the previous time step h_{t-1}. Equations (13) presents the hidden and the output functions, where

i) x_t is the input at time step t;
ii) h_t is the output, or next hidden state;
iii) h_{t-1} is the previous hidden state;
iv) W_h and W_y are the weights corresponding to the input x_t;
v) U_h is the weight corresponding to the previous hidden state h_{t-1};
vi) b_h, b_y, and b_h are the bias vector.
vii) σ is the activation function, i.e., sigmoid, hyperbolic tangent, etc.;

$$\begin{aligned} h_t &= \sigma(W_h x_t + U_h h_{t-1} + b_h) \\ y_t &= \sigma(W_y x_t + b_y) \end{aligned} \tag{13}$$

3) GATED RECURRENT UNITS

The Gated recurrent unit is an improved RNN model that uses two gates to decide the information that will be passed to the output, i.e., the *update gate* (u_t) and the *reset gate* (r_t). The update gate (u_t) has the responsibility of determining how much of the information from previous time steps will be passed to the next time steps. The reset gate (r_t) decides how much of the information from previous time steps will be forgotten. The candidate activation vector ($\hat{h}_t$) calculates the current memory state. The $\hat{h}_t$ vector is computed as a function of the current input and the previous hidden state filtered using the reset gate. Equation (14) presents the update and reset gates, where

i) x_t is the input at time step t;
ii) h_t is the output, or next hidden state;
iii) h_{t-1} is the previous hidden state;
iv) $\hat{h}_t$ is the candidate activation vector;
v) W_u, W_r, and W_h are the weights corresponding to the input x_t;
vi) U_u, U_r, and U_h are the weights corresponding to the previous hidden state h_{t-1};
vii) b_u, b_r, and b_h are the bias vector.
viii) σ_s is the sigmoid activation function;
viii) σ_h is the hyperbolic tangent activation function;
ix) $\odot$ is the element-wise multiplication function.

$$\begin{aligned} u_t &= \sigma_s(W_u x_t + U_u h_{t-1} + b_u) \\ r_t &= \sigma_s(W_r x_t + U_r h_{t-1} + b_r) \\ \hat{h}_t &= \sigma_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - u_t) \odot h_{t-1} + (1 - u_t) \odot \hat{h}_t \end{aligned} \tag{14}$$

4) LONG SHORT-TERM MEMORY NETWORKS

When handling long-term dependencies, the vanilla RNN encounters a phenomenon that affects the gradient, the vanishing or the exploding gradient. This phenomenon causes the actual limitations in how far away in time are the captured dependencies. The exploding gradient can be fixed using the gradient clipping technique, which, as its name suggests, refers to limiting the gradient at a certain threshold. To deal with the vanishing gradient, some functions called gates are used.

The long short-term memory (LSTM) model is an example of model that uses these gates and in addition to the loops in the RNN, they also have extra information called *memory*. There are 3 gates in a LSTM model: *input gate*, *output gate*, and *forget gate*. The inputs to the LSTM at each time step are x_t, the current input, h_{t-1}, the previous hidden state, and c_{t-1} the previous memory state. The outputs at every time step are h_t, the current hidden state, and c_t, the current memory state. Equation (15) shows the expressions of each gate (i.e., f_t, i_t, o_t), the current hidden state (h_t) and current memory state (c_t), where:

i) x_t is the input at time step t;
ii) h_t is the output, or next hidden state;
iii) h_{t-1} is the previous hidden state;
iv) $\tilde{c}_t$ is the cell input activation vector;
v) c_t is the current memory state;
vi) c_{t-1} is the previous memory state;
vii) W_f, W_c, W_i, and W_o are the weights for each gate's current input;
viii) U_f, U_c, U_i, and U_o are the weights for each gate's previous hidden state;
ix) b_f, b_c, b_i, and b_o are the bias vectors;
x) σ_s is the sigmoid activation function;
xi) σ_h is the hyperbolic tangent activation function;
xii) $\odot$ operator is the Hadamard product, i.e., the element-wise multiplication function.

$$\begin{aligned} f_t &= \sigma_s(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma_s(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma_s(W_o x_t + U_o h_{t-1} + b_t) \\ \tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t * \tilde{c}_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{15}$$

As LSTM is good at processing sequences, meaning it is able to capture *past* information, it seems natural to have something that is able to capture *future* information. This is done by the **bidirectional LSTM**, which is an LSTM with two hidden states, one that processes the input in its forward manner, while the second hidden state processes the input backwards.

5) CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks (CNN) are a class of deep neural networks that are specialized in detecting patterns in space, unlike RNNs which detected patterns in time. CNNs are usually employed in computer vision, but they showed great results in text classification as well. Basically, instead of inputting a matrix of pixels from an image, in text classification the input is a 1-dimensional array, to which the CNN applies a window of convolution that can extract n-grams.

Let's consider a sequence of words $w_1, w_2, \ldots, w_n$ as example, each represented by a vector of dimension d. When applying a 1-dimensional convolution with a CNN, this filters the input using a window of size k that slides over the sequence. Equation (16) presents a simple convolution unit, where $x_i = [w_i, w_{i+1}, \ldots, w_{i+k}]$ is the vector of filtered words, $\boldsymbol{w}$ is a weight vector and σ is a non-linear activation function.

$$r_i = \sigma(\boldsymbol{w} x_i) \tag{16}$$

In practice there are usually more filters applied to the same input, so the weight vector $\boldsymbol{w}$ becomes a weight matrix W and a bias b is introduces (Equation (17)).

$$r_i = \sigma(W x_i + b) \tag{17}$$

Another feature of the CNN is the support for multiple input channels. In image classification this is mainly used to separate the RGB color components, but in text processing this allows the inclusion of part-of-speech tags, or the semantic dependency tree.

The convolutional layer in CNN is usually followed by a MaxPooling layer, which extracts the most relevant information provided by the convolution operation, and then outputs to a fully-connected linear layer that acts as the actual classifier.

6) RECURRENT CONVOLUTIONAL NEURAL NETWORKS

A CNN uses a window of a specific size to capture the word context. This comes with the drawback that if the window size is too small, the whole context will not be captured, and if the window size is too large, data sparsity appears.

A recurrent convolutional neural network (RCNN) solves this CNN limitation by employing a recurrent layer as a convolutional layer [26]. This way, instead of sliding a window of fixed size and capturing a limited size context, the bidirectional recurrent structure will capture both the left context and the right context of the word in their entirety. Equation (18) presents the left and right context vectors, where

i) σ is a non-linear activation function;
ii) W_l and W_r are weights applied to the previous word left context and next word right context;
iii) $c_l(w_{i-1})$ and $c_r(w_{i+1})$ are the left context of the previous word and right context of the next word;
iv) W_{sl} and W_{sr} are the weights of the embedding of the previous word and of the next word, respectively;
v) $e(w_{i-1})$ and $e(w_{i+1})$ are the embeddings for the previous word and for the next word, respectively.

$$\begin{aligned} c_l(w_i) &= \sigma(W_l c_l(w_{i-1}) + W_{sl} e(w_{i-1})) \\ c_r(w_i) &= \sigma(W_r c_l(w_{i+1}) + W_{sr} e(w_{i+1})) \end{aligned} \tag{18}$$

The final vector representation of the word becomes the embedding of the word concatenated to the left and right context vectors. Equation (19) presents the output of the first layer.

$$x_i = [c_l(w_i); e(w_i); c_r(w_i)] \tag{19}$$

Equation (20) presents the output to the second layer, where $W^{(2)}, b^{(2)}$ are the weights and biases for the second layer.

$$y_i^{(2)} = \sigma_h(W^{(2)} x_i + b^{(2)}) \tag{20}$$

Equation (21) presents the third layer, i.e., the MaxPooling layer that extract the most important semantic factors from the whole document.

$$y^{(3)} = max_{i=1}^{n} y_i^{(2)} \tag{21}$$

After MaxPooling, a fully connected linear layer that handles the actual classification. Equation (22) presents this layer, where $W^{(4)}, b^{(4)}$ are the weights and bias of the output layer.

$$y_i^{(4)} = W^{(4)} y^{(3)} + b^{(4)} \tag{22}$$

The probability of a document x_i to be in class y_κ can be determined by applying a softmax function (Equation (23)).

$$p(y = y_\kappa | x = x_i) = \frac{e^{y_\kappa^{(4)}}}{\sum_{j=1}^{k} e^{y_j^{(4)}}} \tag{23}$$

7) ATTENTION LAYER

RNNs can be used to predict a sequence based on an input word, or to predict a word based on an input sequence. Moreover, if two RNNs are used one after the other, a sequence can be predicted from an input sequence, an use case mainly encountered in Machine Translation. This type of architecture is called an encoder-decoder network.

Now, seeing that the encoder is a sequence to word model and the decoder is a word to sequence model, the responsibility of the encoder is to capture in one vector the meaning of the whole input sequence, which can prove difficult for large input sequences. Here the Attention mechanism comes in handy. This model assigns importance to hidden states of the encoder and the final context vector that will be the input

to the decoder is actually the original vector with assigned importance weights for each time step.

Equation (24) presents the mathematical formulation of the Attention mechanism, where

i) $e(w_t)$ is the embedding of the word w_t;
ii) M is the length of the sequence;
iii) W and b are the weights matrix and the bias, respectively, of the linear layer of the Attention mechanism;
iv) v are the weights that emphasize the most important dimensions in the space created by the linear layer;
v) α_{w_t} is the importance weight of the word w_t;
vi s is the final value of the context vector;
vii) σ_h is the sigmoid activation function;
viii) σ_{sm} is the softmax activation function.

$$\begin{aligned} u_{w_t} &= \sigma_h(We(w_t) + b) \\ \alpha_{w_t} &= \sigma_{sm}(v^T u_{w_t}) \\ s &= \sum_{t=1}^{M} \alpha_{w_t} e(w_t) \end{aligned} \tag{24}$$

The values of α are defined as the result of a softmax function, meaning that the sum of every α_{w_t} equals to 1, so the Attention mechanism can be interpreted probabilistically. This shows that each value of α_{w_t} is actually the probability that the corresponding word vector $e(w_t)$ is important to the current context.

The described mechanism applies only to the encoder network, meaning that this Attention model can be used anywhere where a context word with importance weights for each time step state is necessary.

8) REINFORCEMENT LEARNING-BASED ADVERSARIAL NETWORKS

Generative adversarial networks (GAN) represent a framework of models, where two neural networks contest in a zero-sum game, meaning that each network gain or loss is balanced by the loss or gain of the other network. One network is called the discriminator and it is trained to categorize if the input data is real or not. The other network is called the generator and it generates fake data for the discriminator to classify.

GANs showed good results in computer vision and image classification, but seeing that the generator generates continuous data, it cannot be used for discrete data like text. To avoid this limitation, the Reinforcement Learning based Adversarial Networks for Semi-supervised learning (RLANS) framework combines reinforcement learning with adversarial networks [28].

In the RLANS framework the discriminator, also known as predictor, is trained with labeled and unlabeled data using policy gradient, and the reward is decided by the judge that identifies predicted and real labels.

For text classification the **predictor** is based on LSTMs. Let $x = \{w_1, \dots w_T | w_t \in \{0, 1\}^k\}$ be a sequence of T one-hot encoded vectors and $y \in \{0, 1\}^c$ the one-hot encoded classification label, where k is the number of words in the vocabulary and c is the number of classes. At the final step, there is a fully connected layer with a ReLU activation function, which gives $h_c \in \mathbb{R}^{1\times d}$, followed by a softmax function. Equation (25) presents the probability of the predicted label, where $B \in \mathbb{R}^{d\times c}$ and $b_p \in \mathbb{R}^{1\times c}$ are the weight matrix and the bias vector, respectively, of the final layer.

$$\begin{aligned} P_\theta(\hat{y}_i = 1|x) &= P_\theta(\hat{y}_i = 1|w_1, \dots w_T) \\ &= \frac{exp(h_c \cdot B_{(:,i)} + b_p^{(i)})}{\sum_{j=1}^{c} exp(h_c \cdot B_{(:,j)} + b_p^{(j)})} \end{aligned} \tag{25}$$

The predictor tries to maximize the reward decided by the judge. Equation (26) presents the reward maximization function, where Y is the set of classes, $P_\theta(\hat{y}|x)$ is the output of the predictor and the action-value $V(\hat{y}, x)$. Equation (27) presents the action-value function, where D_L and D_U the set of labeled and unlabeled examples and $J_\phi(x, \hat{y})$ the output of the judge.

$$R(\theta) = \mathbb{E}[R|X, \theta] = \sum_{\hat{y}\in Y} P_\theta(\hat{y}|x) V(\hat{y}, x) \tag{26}$$

$$V(\hat{y}, x) = \begin{cases} 1 & if\ x \in D_L\ \&\ \hat{y} = y, \\ 0 & if\ x \in D_L\ \&\ \hat{y} \neq y, \\ J_\phi(x, \hat{y}) & if\ x \in D_U. \end{cases} \tag{27}$$

The **judge** network is also based on LSTMs. At each time step t, a sub-network with the same structure as the output layer of the predictor estimates the label $o_t \in \mathbb{R}^{1\times c}$ which is concatenated with the true label y for labeled examples or estimated label $\hat{y}$ for unlabeled examples. The output part of the judge gets as input a weighted combination of the concatenation $[o, y]_\beta = \sum_{t=1}^{T} \beta_t [o_t, y]$, where $\beta \in \mathbb{R}^T$ is a weight vector. The weight vector is represented by two layers: a fully connected one with a ReLU activation function and a sigmoid function. The ReLU function is presented in Equation (28), where $W_{o^{(1)}} \in \mathbb{R}^{2c\times 2c}$ is the weight matrix and $b_{o^{(1)}} \in \mathbb{R}^{1\times 2c}$ is the bias vector. Equation 29 presents the sigmoid function, where $W_{o^{(2)}} \in \mathbb{R}^{2c\times 1}$ is a vector and b_J a bias scalar.

$$o^{(1)} = ReLU([o, y]_\beta \cdot W_{o^{(1)}} + b_{o^{(1)}}) \tag{28}$$

$$J(y; w_1, \dots w_T) = \frac{1}{1 + e^{-(o^{(1)} \cdot W_{o^{(2)}} + b_J}} \tag{29}$$

The judge tries to minimize the cross-entropy (Equation (30)).

$$\begin{aligned} H(x, y) = \min_{\phi} &-\mathbb{E}_{(x,y)\in D_L}[log J_\phi(x, y)] \\ &- \mathbb{E}_{x\in D_U, \hat{y}\sim P_\phi}[log(1 - J_\phi(x, \hat{y}))] \end{aligned} \tag{30}$$

D. EVALUATION

To evaluate the quality of the classification method and correctly interpret the algorithms output for the fake news detection task, we use three metrics, i.e., *accuracy*, *precision*, and *recall*.

Before computing each measure, we need to construct a confusion matrix that holds the following information:

- TP_i (True Positive) is the number of observations that belong to the y_i class and are classified correctly, i.e., $\hat{y}_i = y_i$;
- FN_i (False Negative) is the number of observations that belong to the y_i class and are classified as $y_j (i \neq j)$, i.e., $\hat{y}_i \neq y_i$;
- FP_i (False Positive) is the number of observations that belong to the $y_j (i \neq j)$ class and are classified as y_i, i.e., $\hat{y}_j = y_i$;
- TN_i (True Negative) is the number of observations that belong to the $y_j (i \neq j)$ class and are classified correctly, i.e., $\hat{y}_j = y_j$.

The average accuracy refers to the ratio between correct predictions and total predictions. Thus, average precision measures the per-class effectiveness of a classifier. Equation (31) presents the average accuracy.

$$Accuracy = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i + TN_i}{TP_i + FP_i + TN_i + FN_i} \tag{31}$$

The macro precision averages the precision for each class, i.e., the proportion of correctly predicted positive outputs out of all positive predictions. Thus, macro precision measures the average per-class agreement of the data class labels with those of the classifiers. Equation (32) presents the macro precision.

$$Precision = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i}{TP_i + FP_i} \tag{32}$$

The macro recall refers to the average of the recall for each class, i.e., the proportion of correctly predicted positive outputs out of all actual positives. Thus, macro recall measures the average per-class effectiveness of the classifier to identify labels. Equation (33) presents the macro recall.

$$Recall = \frac{1}{n} \sum_{i=1}^{n} \frac{TP_i}{TP_i + FN_i} \tag{33}$$

IV. PROPOSED SOLUTION

The architecture of our misinformation detection solution is represented by a series of specialized modules with different roles that contribute in classifying the news article in its category, as we can see in figure 1. The first module is the Text Preprocessing Module, which takes the news article as input and outputs the clean text version. The Word Embedding Module outputs a vector representation of the clean text news articles' terms. The Misinformation Detection Module employs the word embeddings and supervised Machine and Deep Learning algorithms to label each news articles and determine their veracity. The quality of classification is measured by the Evaluation Module.

The Text Preprocessing Module has the responsibility of transforming the input news article into a more simplified version of it by eliminating noise and unwanted words or inflections of words. We employ two pipelines, a simple one that lemmatizes the text, and a more aggressive one that removes a lot of terms that bring no information gain.

The Word Embedding Module maps the terms resulted from the Text Preprocessing Module to vector representations. This module reduces the dimensionality of documents considerable as well as the memory requirements but manages to capture the contextual, semantic, and syntactic information.

The Misinformation Detection Module uses different classifier algorithms to analyzes the vector representations of the news articles and outputs their category.

The last module, the Evaluation module, is responsible for measuring the performance of the above classifier, using known metrics such as accuracy, precision and recall.

The source code for our architecture and the used dataset are freely available online on GitHub .[5]

A. TEXT PREPROCESSING MODULE

For text preprocessing, we implemented two pipelines Lemma Text Preprocessing and Aggressive Text Preprocessing. Lemma Text Preprocessing uses lemmatization and replaces pronouns with the *-PRON-* token. The Aggressive Text Preprocessing transforms the text to UTF-8, removes stop words and punctuation, extracts lemmas, and transforms the text to lowercase.

Table 1 presents an example that compares the original article text with the two proposed processing pipelines.

The Lemma Text Preprocessing keeps named entities case, but changes the rest of the text to lowercase. The word inflections are replaced with their lemmas, i.e., *is* becomes *be*. The pronouns are replaced with the token *-PRON-* which helps in lowering the vocabulary size and allows finding connections between all pronouns, regardless of gender or number.

Furthermore, we remove pronouns, not to de-balance classes. Although it is proven that a large number of pronouns is a feature for fake news detection [17], the models can become biased. Thus, the networks can give more importance to this dimension and consider it a representative feature when discriminating between the different classes. The punctuation and stopwords are kept.

The Aggressive Text Preprocessing removes punctuation and stopwords. As pronouns are considered stopwords, they are also removed. The text is also lemmatized, and only the root of the words are stored. Furthermore, named entities are transformed to lowercase. Using this pipeline, the vocabulary size is further reduced.

In some tasks such as Information Retrieval and Text Mining, both stop words and punctuation are seen as noise, and by removing them, the accuracy of the model might improve. The curse of dimensionality [2] is also removed by minimizing the vocabulary size. We note that there are specific situations when the stopwords and punctuation are relevant. Thus, we consider these two approaches.

[5]Source Code https://github.com/ilievladiulian/misinformation-detection

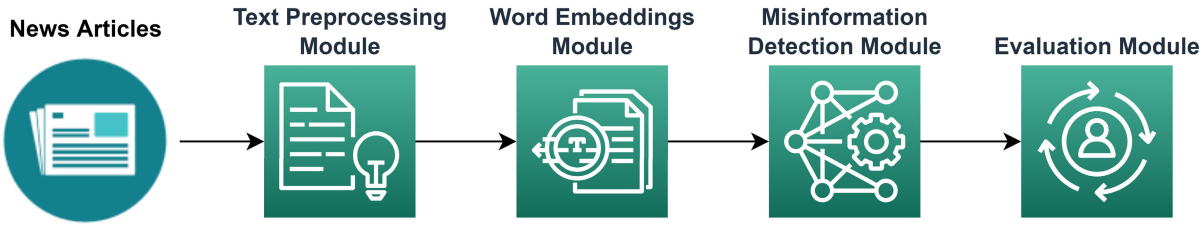

FIGURE 1. Misinformation Detection Architecture Pipeline.

For this module we used the *SpaCy*[6] and the *NLTK*[7] Python libraries. These libraries provide useful modules for Text Analysis and Natural Language Processing, e.g., text and sentence tokenization, part-of-speech tagging, dependency parsing, stop words dictionaries, word stemming and lemmatization, etc..

B. WORD EMBEDDING MODULE

We used three context-aware word embedding models: Word2Vec Skip-Gram model, FastText Skip-Gram model, and GloVe. We chose the Skip-Gram model over the CBOW model because it preserves the word context, allowing the model to capture the context of more distant words in the sequence. GloVe also manages to preserve the local and global context by encoding the words co-occurrence matrix within the embedding.

For each of the word three word embedding described in the Section III, we used two different sets of 300-dimensional word vectors:

i) Generic Word Embeddings containing existing pre-trained word embeddings;
ii) Specific Word Embeddings containing trained word embeddings on our dataset.

These two approaches are employed in order to determine how much the embeddings influence the classification task.

We obtained the Specific Word Embeddings by training the Word2Vec and FastText models available in the *GenSim*[8] library, and the GloVe model provided by the *glove_python* package.[9]

After we applied hyperparameter tuning using the *questions-words* dataset [29] that contains pairs of analogies from different domains, we ended up with the following parameters for training the Specific Word Embeddings. The best models are obtained when using a window size of 10, meaning that the maximum distance between the current word and the predicted word is 10. The minimum count is set to 2, i.e., the minimum frequency for a word to be taken into account. The number of workers is set to 10 and the number of interactions to 10.

For the Generic Word Embeddings, we use the GloVe vectors trained on the Wikipedia 2014 and Gigaword 5 dataset, the Word2Vec vectors trained on the Google News dataset, and the FastText vectors trained on the Common Crawl dataset.

C. MISINFORMATION DETECTION MODULE

The Misinformation Detection Module contains Machine and Deep Learning algorithms (Table 2) implemented in *Pytorch*[10] using GPUs.

We chose Logistic Regression to be the baseline for our classification as it is a classical supervised Machine Learning algorithm similar to the perceptron which is proven to be robust for correlated features. Thus, when there are many correlated features as it happens with textual data, Logistic Regression assigns a more accurate probability to the correct label. CNN architectures are frequently used for classification in fake news detection. In the literature, there are also many small variations of this architecture (e.g., [27]). One of the main advantages of using CNN is that it is computationally efficient. RNN based architectures are also been used to detect misinformation, although there are a couple of known issues, i.e., gradient vanishing and exploding problems. There are several solutions if this problem occurs. Firstly, RNN can be used in hybrid architectures. In general, they are used with convolutional layers (e.g., [1]). Secondly, a specific variant of RNN can be used, LSTM, which can efficiently capture long-range dependencies in the text. There are many LSTM based approaches for the detection of fake news (e.g., [47]). Thirdly, GRUs are an improvement from the basic RNN architecture, as they can capture long-term dependencies to learn the linguistic features from the news articles (e.g., [35]). Attention mechanisms can be added to the neural networks, which can lead to better performance in detecting the fake news. These mechanisms are used to indicate important parts of a news article, by looking over all the information and then generating the proper word according to the context of the current word it works on. The RLANS model combines reinforcement learning with adversarial networks (GAN), using a judge to reword the quality of unsupervised classification done by a predictor. Although it was used in text classification, this model was never used in fake news detection.

[6]https://spacy.io/
[7]https://www.nltk.org/
[8]https://radimrehurek.com/gensim/
[9]https://github.com/maciejkula/glove-python
[10]https://pytorch.org/

TABLE 1. Preprocessing techniques example (Note: The text was truncated and the example is for showcase purposes only).

Method	Text
Original Text	When Sebastian Thrun started working on self-driving cars at Google in 2007, few people outside of the company took him seriously. "I can tell you very senior CEOs of major American car companies would shake my hand and turn away because I wasn't worth talking to, " said Thrun, in an interview with Recode earlier this week.
Lemma Text Preprocessing	when Sebastian Thrun start work on self - drive car at Google in 2007 , few people outside of the company take -PRON- seriously . " -PRON- can tell -PRON- very senior ceo of major american car company would shake -PRON- hand and turn away because -PRON- be not worth talk to , " say Thrun , in an interview with Recode early this week .
Aggressive Text Preprocessing	sebastian thrun start work drive car google 2007 people company senior ceo major american car company shake hand turn worth talk thrun interview recode early week

TABLE 2. Machine and deep learning algorithms.

Notation	Algorithm
LogReg	Logistic Regression
RNN	Recurrent Neural Network
BiRNN	Bidirectional Recurrent Neural Network
Attn + RNN	Recurrent Neural Network with Attention
GRU	Gated Recurrent Unit
Attn + GRU	Gated Recurrent Unit with Attention
LSTM	Long Short Term Memory
Attn + LSTM	Long Short Term Memory with Attention
CNN	Convolutional Neural Network
RCNN	Recurrent Convolutional Neural Network
RLANS	Reinforcement Learning based Adversarial Network for Semi-supervised learning

D. HYPERPARAMETER TUNING

After applying hyperparameter tuning, we have reached the following configurations. The adversarial network is trained over 100 epochs with early stopping with a batch size of 16. Similarly, the other models are trained for 100 epochs with early stopping and a batch size of 4.

There are some setup parameters common to each classifier, like the input size of 300, given by the size of the word vectors and an output size of 10, given by the number of labels. Each model used the same optimizer Adam. Adam is configured with a learning rate of 0.0001 and a weight decay of 0.0005, to minimize the cross-entropy loss function.

The Logistic Regression model, the RNN, the LSTM, the GRU, the RCNN and the Attn + LSTM are initialized with a 256-dimensional hidden layer. Moreover, the recurrent models have their hidden state reset after each batch so that they do not introduce information from the previous word sequence in the next word sequence which will act as noise.

The CNN uses 1 input channel corresponding to the news content, 16 output channels which are passed to MaxPooling layer, and a stride of length 1 for the kernel. We use cross-validation to determine that the best the kernel size is 3. We also test the CNN with kernel sizes of 5 and 7.

E. EVALUATION MODULE

We implemented the three evaluation metrics presented in Section III using Python. For evaluation, we train the model on the training dataset and obtain the predictions $\hat{y}_i$ for each document x_i. Then we compute the confusion matrix for the test dataset. Finally, we obtain the average accuracy, macro precision, and macro recall using the values in the confusion matrix.

V. EXPERIMENTAL RESULTS

In this section, we first present the dataset, its size, and the labels used. Then, we analyze the classification results of our benchmark w.r.t. the preprocessing pipeline, the word embeddings, and the architecture.

A. DATA SET

The dataset used for experiments is the open source Fake News Corpus dataset available on Github.[11] The dataset has been also used for determining the veracity of news articles in [25] by using sampling to create a fake and reliable dataset. In this work, we take a multi-class approach. It is comprised of more than 9 million news articles that contain the following information: the domain of the news provider where the article was found, the type of news that acts as its label, the actual content of the news, its title, its authors and others. For these experiments, we use only the news content and their label.

An example of an entry in the dataset is ''Obama's The Greatest Criminal In History Say Trump And Joe Arpaio Because He Forged His Birth Certificate (Video)'' labeled as *fake*. The experiments were run in two sets, each of them with a different preprocessing pipeline.

The first preprocessing applies a SpaCy lemmatization, which transforms the phrase into ''Obama ' be greatest criminal in history say Trump and Joe Arpaio because -PRON- forge -PRON- birth certificate''. As we can see, the words suffered a lowercase transformation, the pronouns are replaced with the token *-PRON-*, and each word was replaced with its lemma.

The second preprocessing pipeline consists of a more aggressive approach, which removed all known English stopwords present in the SpaCy and NLTK packages, removed all punctuation, and only after that a SpaCy lemmatization was applied. This changes the dataset entry to ''obama greatest criminal history say trump joe arpaio forge birth certificate''.

For our experiments we used 10 of the classes available in the dataset, each class represented by 10 000 documents. Table 3 presents the available classes.

[11] https://github.com/several27/FakeNewsCorpus

After preprocessing, the raw text, the lemma text preprocessing, the aggressive preprocessing and class for each article is stored in a single instance MongoDB v4.2.15. Thus, by utilizing a NoSQL document-oriented database, we manage to *1)* improve the article retrieval process for the word embedding models and classification algorithms and *2)* remove the need to reapply the preprocessing pipeline each time the algorithms are use.

The dataset is split as follows: 20% is used for the testing dataset and 80% for the training dataset. We keep the distribution of labels equal for both training and testing datasets, i.e., 10% of records for each data set are labeled with the same class. Moreover, for the adversarial network the train set was split even further into 50% labeled and 50% unlabeled data. Again, the distribution of labels is kept to achieve balanced datasets.

TABLE 3. News articles classes.

Class	Description
Fake News	Fabricated or distorted information
Satire	Humorous or ironic information
Extreme Bias	Propaganda articles
Conspiracy Theory	Promote conspiracy theories
Junk Science	Scientifically dubious claims
Hate News	Promote discrimination
Clickbait	Credible content, but misleading headline
Proceed With Caution	May be reliable or not
Political	Promote political orientations
Credible	Reliable information

B. RESULTS

We tested all the 11 classification models using each of the three word embeddings with each of the processing pipelines described above. We split the dataset using an 80%-20% training-testing ratio with random seeding. For each split, we maintain the stratification of the labels. For all our models, we employ an early stopping mechanism. Each algorithm was executed 10 times before computing an average for each metric. We use an NVIDIA® DGX Station™ containing 4 NVIDIA® V100 Tensor Core GPUs for our experiments. Table 4 presents the results.

1) LEMMA TEXT PROCESSING RESULTS

Table 5 presents the results for Lemma text preprocessing. We notice that the accuracy of the Logistic Regression model is well below the accuracy of the Deep Learning models, excepting the RNN, the Attention RNN, and the RLANS regardless of embedding, i.e., Generic or Specific Word Embeddings. Also, it looks like Logistic Regression is the only model that consistently performs worse when using Specific Word Embeddings vectors instead of Generic Word Embeddings ones. This suggests that the vectors resulted from training word embedding algorithms on our own dataset are more dense and the linear layer in the Logistic Regression model cannot find a hyperplane that separates the classes with a high enough accuracy.

The results for the RNN and the Attention RNN are better for the Specific Word Embedding but worse for Generic Word Embeddings when compared to the Logistic Regression. Moreover, it seems that for Generic Word Embeddings, the Attention mechanism provides an increase in performance, of about 10% accuracy for Word2Vec and FastText. However, the Attention mechanism does not seem to improve the results for Specific Word Embedding. In the case of the RNN, the Word2Vec embedding shows an increase of 15% in accuracy when trained on our dataset.

The Bidirectional RNN model shows an increase in accuracy of about 20% over the Logistic Regression model for the Generic Word Embeddings and about 40% for the Specific Word Embedding vectors. The difference of accuracy between these embedding types is of 1% for GloVe and FastText, and a 7% increase for Word2Vec. This shows that less noisy word vectors relative the misinformation detection task can influence the overall performance of the model, especially for Word2Vec vectors.

As expected, the GRU model shows even better results than the Bidirectional RNN, with about 5% increase in accuracy for both Generic and Specific Word Embeddings. This suggests that there is useful information in long dependencies in the input sequences, dependencies that the RNN could profit very well from. Moreover, this confirms the hypothesis that Specific Word Embeddings for the task of misinformation detection improve the model results, with an increase of 10% accuracy for Word2Vec and 4% for FastText. However, when using GloVe, the performance drops with 2%. As in the case of the unidirectional RNN and the Bidirectional RNN, it seems Word2Vec embeddings find the greatest improvement when trained specifically on this dataset.

The Attention GRU model improves the performance of the simple GRU. In the case of the Generic Word Embeddings, there is an increase in accuracy of 3, 9 and 6 % for GloVe, Word2Vec and FastText, reaching a steady accuracy of 82% for all three embedding algorithms. The behavior is similar for the Specific Word Embeddings, showing an increase of 4 and 2% for GloVe and FastText over the simple GRU. This is not the case for Word2Vec, but seeing that for custom trained vectors, Word2Vec already increased the performance, it is not surprising to not see an obvious improvement. It seems that for custom trained vectors, a steady accuracy of 82% was reached as well.

The LSTM performance is worse than GRU and Bidirectional RNN, with a drop in accuracy of almost 20%. This suggests that the parameters of the model could be more fine-tuned for this task. The custom embeddings improved the results with an increase of 1% in accuracy when using GloVe, a 6% increase for FastText. As expected after analyzing the previous models, the Word2Vec embedding profits most from the custom training by showing and increase of 23% in accuracy.

When adding the Attention mechanism to the LSTM, we find consistency in the results of the model across all embeddings, seeing an accuracy of 81% when using GloVe

TABLE 4. Experimental results. Note 1: The italic bold numbers present the best results for each metric given a combination of algorithm, embedding model, and text processing steps. Note 2: The overall best results are marked with bold underlined.

Algorithm	Word Embedding	Lemma Text Preprocessing						Aggressive Text Preprocessing					
		Generic Word Embedding			Specific Word Embedding			Generic Word Embedding			Specific Word Embedding		
		Accuracy	Recall	Precision	Accuracy	Recall	Precision	Accuracy	Recall	Precision	Accuracy	Recall	Precision
LogReg	GloVe	***54.19***	***54.19***	***60.52***	30.58	30.58	37.00	53.97	53.96	58.68	49.62	49.63	60.71
	Word2Vec	***53.41***	***53.41***	***61.71***	38.58	38.57	43.61	52.92	52.92	58.92	44.92	44.92	55.20
	FastText	***56.37***	***56.36***	***63.00***	22.42	22.42	48.14	55.08	55.07	61.91	37.60	37.60	47.89
RNN	GloVe	51.04	51.04	56.30	43.56	43.56	55.23	51.69	51.69	54.06	***55.54***	***55.54***	***58.89***
	Word2Vec	37.72	37.72	42.96	52.37	52.37	54.87	***53.04***	***53.04***	***57.03***	51.33	51.33	52.56
	FastText	46.81	46.81	54.61	52.55	52.54	55.46	***54.22***	***54.21***	***56.09***	52.56	52.56	56.02
BiRNN	GloVe	71.66	71.66	73.99	72.55	72.54	74.86	70.30	70.29	73.59	***73.69***	***73.70***	***76.22***
	Word2Vec	69.84	69.84	73.63	***76.23***	***76.23***	***76.59***	68.28	68.29	71.23	74.02	74.01	75.49
	FastText	71.22	71.22	72.45	***74.24***	***74.24***	***74.61***	71.41	71.40	75.50	73.95	73.95	76.24
Attn + RNN	GloVe	51.81	51.81	59.85	48.99	48.99	57.34	***61.16***	***61.17***	***68.92***	60.55	50.56	63.76
	Word2Vec	49.20	49.19	56.60	45.03	45.03	48.80	51.59	51.60	48.35	***61.48***	***61.47***	***67.62***
	FastText	58.03	58.04	***63.57***	47.49	47.50	56.54	***59.03***	***59.03***	60.17	52.30	52.30	53.53
GRU	GloVe	79.28	79.28	80.90	77.51	77.51	81.35	***80.81***	***80.80***	***81.77***	79.87	79.87	81.60
	Word2Vec	73.66	73.66	76.92	83.84	83.84	84.72	75.66	75.66	77.50	***84.09***	***84.09***	***85.02***
	FastText	76.70	76.70	78.71	80.44	80.44	81.19	81.04	81.04	82.30	***82.32***	***82.32***	***83.33***
Attn + GRU	GloVe	***82.93***	***82.93***	***84.22***	81.91	81.91	83.16	***82.93***	***82.93***	83.69	81.87	81.87	83.27
	Word2Vec	82.81	82.80	***84.66***	81.97	81.96	83.63	***83.28***	***83.28***	84.59	82.19	82.19	83.49
	FastText	82.11	82.12	83.40	82.19	82.20	83.33	82.06	82.06	***83.92***	***82.34***	***82.35***	83.04
LSTM	GloVe	60.16	60.16	67.09	61.90	61.89	67.38	***75.43***	***75.43***	***77.08***	60.69	60.69	65.00
	Word2Vec	56.78	56.78	61.56	79.95	79.96	81.06	58.38	58.38	61.18	***83.06***	***83.05***	***83.80***
	FastText	60.79	60.79	62.94	66.95	66.95	68.59	63.70	63.70	67.16	***81.50***	***81.50***	***82.16***
Attn +LSTM	GloVe	81.14	81.14	83.25	81.18	81.18	***83.68***	***81.39***	***81.39***	82.27	78.62	78.62	80.12
	Word2Vec	79.58	79.58	***83.95***	81.00	81.00	83.10	***81.25***	***81.26***	82.55	81.15	81.15	82.67
	FastText	***81.72***	***81.72***	***83.62***	81.08	81.07	82.48	78.89	79.89	82.37	81.09	81.10	83.64
CNN	GloVe	80.06	80.06	81.08	***82.07***	***82.07***	***83.28***	78.53	78.52	80.32	79.78	79.77	80.90
	Word2Vec	78.46	78.46	79.10	***81.00***	***81.00***	***81.40***	76.73	76.73	77.58	79.69	79.70	81.25
	FastText	***80.91***	***80.91***	***82.02***	74.67	74.68	75.85	80.24	80.24	81.27	74.71	74.71	76.97
RCNN	GloVe	***86.86***	***86.86***	***86.95***	86.20	86.20	86.78	85.35	85.35	85.73	84.83	84.83	86.16
	Word2Vec	84.72	84.26	85.21	**87.56**	**87.55**	**88.11**	83.03	83.03	83.82	85.69	85.69	86.12
	FastText	***86.14***	***86.13***	***87.17***	85.68	85.68	86.55	85.54	85.54	86.04	84.85	84.85	85.63
RLANS	GloVe	***71.86***	***71.85***	***75.45***	65.47	65.47	70.52	59.07	59.07	62.49	57.05	57.05	60.08
	Word2Vec	52.45	52.44	57.98	***80.76***	***80.76***	***81.87***	54.45	54.45	59.03	72.08	72.80	74.59
	FastText	59.44	59.44	65.49	68.17	68.16	68.77	59.72	59.71	64.67	***79.19***	***79.19***	***81.20***

TABLE 5. Lemma text preprocessing results.

Algorithm	Word Embedding	Generic Raw Text			Specific Raw Text		
		Accuracy	Recall	Precision	Accuracy	Recall	Precision
LogReg	GloVe	54.19	54.19	60.52	30.58	30.58	37.00
	Word2Vec	53.41	53.41	61.71	38.58	38.57	43.61
	FastText	56.37	56.36	63.00	22.42	22.42	48.14
RNN	GloVe	51.04	51.04	56.30	43.56	43.56	55.23
	Word2Vec	37.72	37.72	42.96	52.37	52.37	54.87
	FastText	46.81	46.81	54.61	52.55	52.54	55.46
BiRNN	GloVe	71.66	71.66	73.99	72.55	72.54	74.86
	Word2Vec	69.84	69.84	73.63	76.23	76.23	76.59
	FastText	71.22	71.22	72.45	74.24	74.24	74.61
Attr + RNN	GloVe	51.81	51.81	59.85	48.99	48.99	57.34
	Word2Vec	49.20	49.19	56.60	45.03	45.03	48.80
	FastText	58.03	58.04	63.57	47.49	47.50	56.54
GRU	GloVe	79.28	79.28	80.90	77.51	77.51	81.35
	Word2Vec	73.66	73.66	76.92	83.84	83.84	84.72
	FastText	76.70	76.70	78.71	80.44	80.44	81.19
Attr + GRU	GloVe	82.93	82.93	84.22	81.91	81.91	83.16
	Word2Vec	82.81	82.80	84.66	81.97	81.96	83.63
	FastText	82.11	82.12	83.40	82.19	82.20	83.33
LSTM	GloVe	60.16	60.16	67.09	61.90	61.89	67.38
	Word2Vec	56.78	56.78	61.56	79.95	79.96	81.06
	FastText	60.79	60.79	62.94	66.95	66.95	68.59
Attr + LSTM	GloVe	81.14	81.14	83.25	81.18	81.18	83.68
	Word2Vec	79.58	79.58	83.95	81.00	81.00	83.10
	FastText	81.72	81.72	83.62	81.08	81.07	82.48
CNN	GloVe	80.06	80.06	81.08	82.07	82.07	83.28
	Word2Vec	78.46	78.46	79.10	81.00	81.00	81.40
	FastText	80.91	80.91	82.02	74.67	74.68	75.85
RCNN	GloVe	86.86	86.86	86.95	86.20	86.20	86.78
	Word2Vec	84.72	84.26	85.21	**87.56**	**87.55**	**88.11**
	FastText	86.14	86.13	87.17	85.68	85.68	86.55
RLAN	GloVe	71.86	71.85	75.45	65.47	65.47	70.52
	Word2Vec	52.45	52.44	57.98	80.76	80.76	81.87
	FastText	59.44	59.44	65.49	68.17	68.16	68.77

and FastText, regardless if the vectors were custom trained or not. Here, Word2Vec improves the results again, increasing the 79% accuracy when using the Generic Word Embeddings to an 81% accuracy when using custom vectors, performance similar to the other embeddings.

For CNN, the overall performance of this model is similar to the performance of the LSTM with Attention, so it performs better than the recurrent models. This suggests that useful information is encoded in the context of the words, something that the CNN is able to capture using its kernel. Here we also see an increase of 2% in accuracy when using GloVe and Word2Vec. However, we see a drop in accuracy for FastText of 5% when trained on our dataset.

The RCNN shows the most promising performance from all the tested models. Its accuracy is 5-6% higher than both CNN and LSTM with Attention, regardless of the word embedding. While for GloVe and FastText there are not noticeable differences, the Word2Vec embedding sees an increase of 3% in accuracy for vectors trained on our dataset. The results are influenced by the way the network learns to detect misinformation. The first major difference between this architecture and the other architectures is that the network uses a CNN window of a specific size to capture the word context. The second difference is related to the enhanced word embedding created by concatenating the left and right context which are learned separately.

The RLANS model has a lower accuracy than the LSTM for Word2Vec (4%) and FastText (1%) when using the Specific Word Embeddings, but a higher accuracy for GloVe (11%). Similarly to the LSTM model, using custom trained vectors shows an increase of 28% in accuracy for Word2Vec and 9% for FastText. However, the GloVe embedding shows a drop of 6% in accuracy. Since the discriminator network that does the actual classification is based on an LSTM, we would expect results at least in the same range as the LSTM. This expectation is supported when comparing the results of the custom trained embeddings of RLANS with the LSTM, where we see a higher accuracy for RLANS for each of the embedding employed.

2) AGGRESSIVE TEXT PREPROCESSING RESULTS

Table 6 presents the results for Aggressive Text Preprocessing. For Logistic Regression, we see that the accuracy when using the Generic Word Embeddings drops by approximately 1%, but the accuracy for the Specific Word Embeddings increased by 19% for GloVe, 6% for Word2Vec, and 15% for FastText. This shows that punctuation and stopwords introduce noise in the Logistic Regression model.

The RNN shows more consistent results across embeddings when using the Generic Word Embeddings. However, the use of the Specific Word Embeddings does not show an increase in accuracy for all embeddings. Instead, for Word2Vec and FastText we see a drop of 2% in accuracy, while for GloVe we observe an increase of 4%.

For the BiRNN model, the aggressive preprocessing shows a drop of 1% in accuracy when using the Generic Word Embeddings. Similarly, when using the Generic Word Embeddings, we see an increase in accuracy, especially for Word2Vec (6%) over the baseline.

The Attention RNN shows an increase in accuracy for Aggressive Text Preprocessing compared to Lemma Text Preprocessing regardless the vectors were obtained using Generic or Specific Word Embeddings. We observe that for the generic GloVe vectors, the accuracy is higher with 10%, for Word2Vec with 2%, and for FastText with 1%. Moreover, Word2Vec shows one more time that custom trained vectors on the dataset bring an increase in accuracy (10%).

Similarly to the Attention RNN, the GRU also has higher accuracy. For Specific Word Embeddings, the accuracy increases with 1% when using the GloVe embedding. For Generic Word Embeddings vectors, the accuracy increases with 2% and 5% when using the Word2Vec and FastText, respectively. Again, custom trained Word2Vec shows an improvement of the model accuracy, increasing it with 6%.

The results for the Attention GRU are similar, regardless of the preprocessing pipeline. They are in the range of 82-83% accuracy for the Generic Word Embeddings, dropping to 81-82% for Specific Word Embeddings. For LSTM, we observe that an higher accuracy is obtained when comparing the two preprocessing pipelines with Generic Word Embedding. For GloVe the increase is 15%. Word2Vec obtains an increase in accuracy of 2%, while FastText an increase of 3%. When using custom trained GloVe, the accuracy drops by 15% when compared to the accuracy obtained by the pre-trained vectors. The network trained with Word2Vec obtains an increases in accuracy of 25%, while with FastText the increase is 18%.

The Attention LSTM shows a higher pre-trained Word2Vec accuracy (2%) but a lower pre-trained FastText accuracy (3%). The results for custom trained vectors are similar, regardless of the preprocessing, excepting GloVe, where we see a 2% lower accuracy.

The results of the CNN are lower for the aggressive preprocessing, with a 2% drop for GloVe and a 2% drop for FastText. The Specific Word Embeddings improve the accuracy by 1% for GloVe and 3% for Word2Vec. For FastText, the accuracy decreases by 6%. These results show that the context window of the CNN finds important information in stopwords and punctuation.

Similarly to the CNN, the RCNN also shows a drop of 1-2% in accuracy for all embeddings when using the aggressive preprocessing, which supports the hypothesis that the context window extract information from punctuation and stopwords. The results of the custom trained Word2Vec compared to the pre-trained Word2Vec are consistent with the other models, where we see an increase in accuracy larger than the other embeddings (2.5%).

The RLANS network shows a lower accuracy for pre-trained GloVe (12%), but a higher one for pre-trained Word2Vec (2%). The pre-trained FastText behaves the same. When Specific Word Embeddings are used, the network accuracy drops by 2% for GloVe vectors, while for Word2Vec

TABLE 6. Aggressive text preprocessing results.

Algorithm	Word Embedding	Generic Raw Text			Specific Raw Text		
		Accuracy	Recall	Precision	Accuracy	Recall	Precision
LogReg	GloVe	53.97	53.96	58.68	49.62	49.63	37.00
	Word2Vec	52.92	52.92	58.92	44.92	44.92	43.61
	FastText	55.08	55.07	61.91	37.60	37.60	48.14
RNN	GloVe	51.69	51.69	54.06	55.54	55.54	55.23
	Word2Vec	53.04	53.04	57.03	51.33	51.33	54.87
	FastText	54.22	54.21	56.09	52.56	52.56	55.46
BiRNN	GloVe	70.30	70.29	73.59	73.69	73.70	74.86
	Word2Vec	68.28	68.29	71.23	74.02	74.01	76.59
	FastText	71.41	71.40	75.50	73.95	73.95	74.61
Attr + RNN	GloVe	61.16	61.17	68.92	60.55	50.56	57.34
	Word2Vec	51.59	51.60	48.35	61.48	61.47	48.80
	FastText	59.03	59.03	60.17	52.30	52.30	56.54
GRU	GloVe	80.81	80.80	81.77	79.87	79.87	81.35
	Word2Vec	75.66	75.66	77.50	84.09	84.09	84.72
	FastText	81.04	81.04	82.30	82.32	82.32	81.19
Attr + GRU	GloVe	82.93	82.93	83.69	81.87	81.87	83.16
	Word2Vec	83.28	83.28	84.59	82.19	82.19	83.63
	FastText	82.06	82.06	83.92	82.34	82.35	83.33
LSTM	GloVe	75.43	75.43	77.08	60.69	60.69	67.38
	Word2Vec	58.38	58.38	61.18	83.06	83.05	81.06
	FastText	63.70	63.70	67.16	81.50	81.50	68.59
Attr + LSTM	GloVe	81.39	81.39	82.27	78.62	78.62	83.68
	Word2Vec	81.25	81.26	82.55	81.15	81.15	83.10
	FastText	78.89	79.89	82.37	81.09	81.10	82.48
CNN	GloVe	78.53	78.52	80.32	79.78	79.77	83.28
	Word2Vec	76.73	76.73	77.58	79.69	79.70	81.40
	FastText	80.24	80.24	81.27	74.71	74.71	75.85
RCNN	GloVe	85.35	85.35	85.73	84.83	84.83	86.78
	Word2Vec	83.03	83.03	83.82	**85.69**	**85.69**	**88.11**
	FastText	85.54	85.54	86.04	84.85	84.85	86.55
RLAN	GloVe	59.07	59.07	62.49	57.05	57.05	70.52
	Word2Vec	54.45	54.45	59.03	72.08	72.80	81.87
	FastText	59.72	59.71	64.67	79.19	79.19	68.77

vectors increases it with 18%, and FastText with 20%. This behavior is similar to the raw text preprocessing pipeline.

VI. DISCUSSION

In this section, we discuss the findings and analyze the results of our benchmark. First, we discuss and compare with current literature our findings w.r.t. the dataset. Second, we analyze the results from the text preprocessing perspective and compare our results with current approaches. Third, we examine the importance of pre-trained and trained word embeddings and present the advantages and drawbacks of current techniques. Lastly, we address the impact of the preprocessing pipeline and word embeddings on the models' classification quality.

A. DATASET

The misinformation detection problem is a complex task that requires large datasets to accurately determine news content veracity. In the current literature, most papers test their proposed solution on relatively small datasets,e.g., 3 004 news articles in [16], 422 news articles in [44], 4 180 and 6 728 news articles in [27], etc., with the largest containing under 50 000 news articles [24]. Although we emphasize that some of them also use larger corpora, e.g., 103,219 documents in [38]. Likewise, in some cases, the dataset consists of short news statements, e.g., the PolitiFact dataset with 12.8K short texts, which directly influences the accuracy of the prediction model, as can be seen in [30], where the best accuracy is $\approx 40\%$.

We chose to address this shortcoming by using a large dataset containing 100 000 news articles. Furthermore, the current research articles only address the misinformation detection problem using either a binary approach, i.e., the news articles' veracity is either true or false, e.g., [41], [43], [44], etc., or levels of veracity [24]. None of the current approaches uses a dataset that contains fake news labeled with different categories.

Our approach brings new insights regarding the detection of different types of misinformation. We observe that some neural networks manage to obtain a score of over 80% for precision and recall. We can conclude that a clear separability between the classes can be achieved between real and different kinds of misinformation articles, even though 90% for the dataset contains verifiable false news. Thus, Neural Networks together, with Generic and Specific Word Embeddings trained on Lemma and Aggressive Text Preprocessing, manage to clearly differentiate between real and fake news as well as different types of misinformation.

B. LEMMA vs. AGGRESSIVE TEXT PREPROCESSING

Text preprocessing is an import step employed for both Text Analysis and Natural Language Processing tasks. Text preprocessing is used to remove terms that bring no information gains and to minimize the vocabulary. This step can greatly influence the accuracy and quality of Machine and Deep Learning models.

In the current research, the majority of the proposed approaches for fake news detection do not employ this very important step [16], [22], [27], [30], [34] or use very shallow text preprocessing, e.g., in [24] the authors eliminated question marks, website addresses, links, e-mail addresses, duplicate named entities (the most repetitive words of this type were eliminated from the entire dataset).

In our approach, we employ two different text preprocessing pipelines to determine the influence upon the classification models. The two approaches are Lemma Text Preprocessing and Aggressive Text Preprocessing, described in detail in Section IV. The experiments (Table 4) show that the Aggressive Text Preprocessing pipeline obtains better results w.r.t. the Machine Learning algorithm and word embedding. The results show 15 <*Model, Text Preprocessing, Word Embedding*> pairs that obtain the highest accuracy when using Lemma Text Preprocessing and 18 pairs with the highest accuracy when using Aggressive Text Preprocessing. Furthermore, the results are also highly dependent on the word embedding.

Figures 2, 3, and 4 present the comparison between Lemma and Aggressive Text Preprocessing when using Generic Word Embeddings. Figures 5, 6, and 7 present the comparison between Lemma and Aggressive Text Preprocessing when using Specific Word Embeddings.

When comparing the two preprocessing techniques, w.r.t. the Generic Word Embeddings, we find the following results. When using the Generic GloVe embedding (Figure 2), the highest accuracy is obtained by the RCNN model for both preprocessing techniques, where the Lemma Text Preprocessing presents slightly better results than the Aggressive Text Preprocessing, i.e., 86.86% vs. 85.35%. For the Generic Word2Vec (Figure 3), we achieve the highest accuracy for Lemma Text Preprocessing when using RCNN, i.e., 84.72%. While the highest accuracy for the Aggressive Text Preprocessing with this embedding is obtained by the Attr + GRU, i.e., 83.28%. When employing the Generic FastText embedding (Figure 4), we observe a similar trend as when using the GloVe embedding. Again, the RCNN model obtains the highest accuracy with 86.14% for Lemma Text Preprocessing and 85.54% for Aggressive Text Preprocessing.

We obtain the following results when comparing the two preprocessing techniques, w.r.t. the Specific Word Embeddings. The highest accuracy is achieved by RCNN when using the Specific GloVe embedding for both text preprocessing techniques (Figure 5), with 86.20% and 84.83% for Lemma Text Preprocessing, Aggressive Text Preprocessing, respectively. We observe a similar pattern when employing

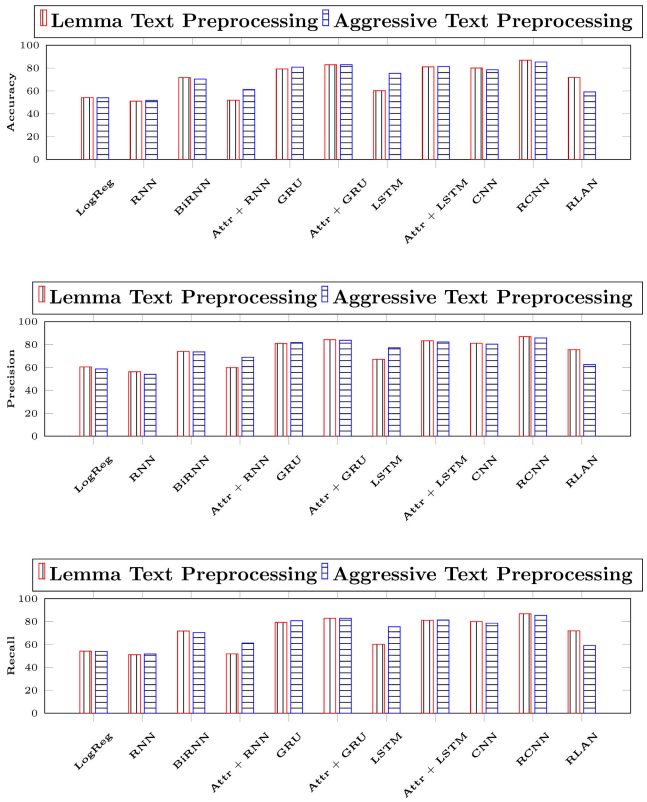

FIGURE 2. Lemma vs. aggressive text preprocessing using the generic GloVe embedding.

Word2Vec (Figure 6) and FastText (Figure 7). Both models have a higher accuracy than GloVe w.r.t. the best configuration. The overall best model is again RCNN. For Specific Word2Vec, the accuracy of RCNN is 87.56% and 85.69% for Lemma Text Preprocessing, Aggressive Text Preprocessing, respectively. While for Specific FastText, the accuracy of the model is 85.68% and 84.85%, respectively.

By analyzing only the combinations < *Model, Text Preprocessing, Word Embedding* > that obtain the highest accuracy, we can conclude that the Lemma Text Preprocessing gives better results than Aggressive Text Preprocessing. Thus, we observe an improvement of Lemma Text Preprocessing over Aggressive Text Preprocessing between 0.60% (for the combination < *RCNN, Lemma Text Preprocessing, Generic* FastText >) to 1.87% (for the combination < *RCNN, Lemma Text Preprocessing, Specific* Word2Vec >).

C. GENERIC vs. SPECIFIC WORD EMBEDDINGS

Pre-trained Word2Vec was employed in the models presented in [27] and [16]. Moreover, the authors in [16] also experimented with GloVe, claiming that they did not observe important differences. Pre-trained GloVe embeddings were also used by [22], [34]. Furthermore, in [22], the authors claim that they also tested their models using Word2Vec but obtained lower results. Pre-trained BERT-BASE was used in [24] and [30]. All these models use word embeddings

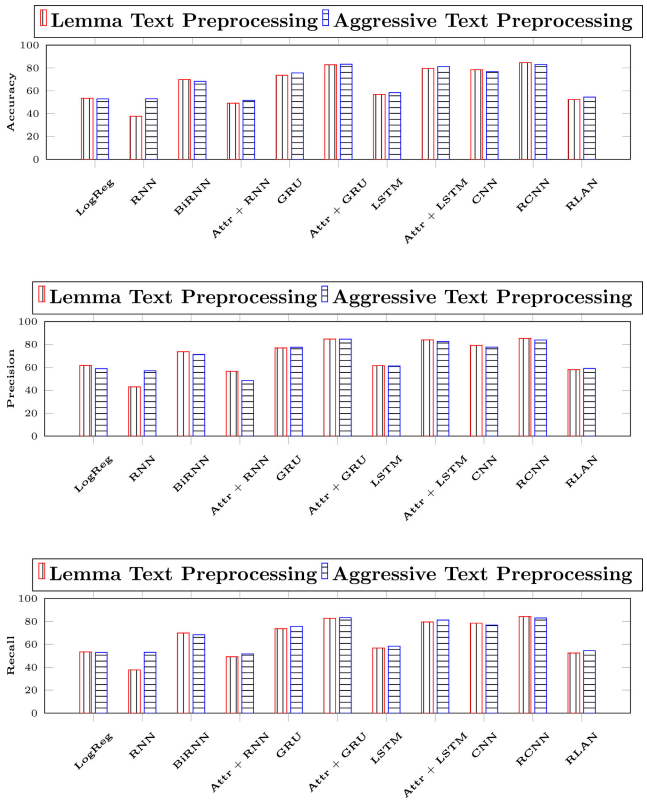

FIGURE 3. Lemma vs. aggressive text preprocessing using the generic Word2Vec embedding.

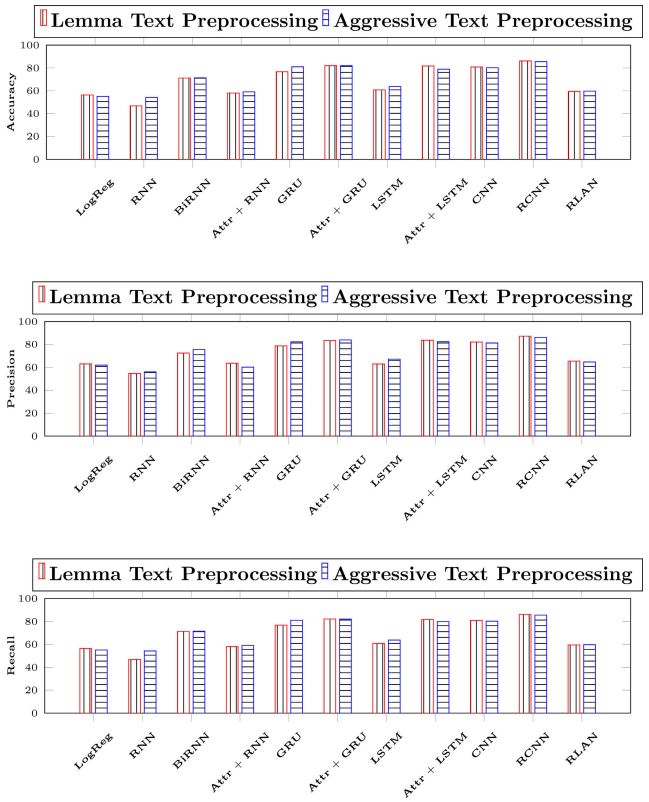

FIGURE 4. Lemma vs. aggressive text preprocessing using the generic FastText embedding.

as black boxes none of them were trained on the dataset or fine-tuned the parameter. Furthermore, the current literature lacks a comprehensive discussion comparing different word embeddings and transformers.

To test the performance of word embeddings for the task of misinformation detection, we use three word embeddings Word2Vec, FastText, and GloVe. Both Word2Vec and FastText are trained using the Skip-Gram model. We also used two approaches for our word embeddings: Generic Word Embedding (pre-trained) and Specific Word Embeddings (trained on our corpus).

The experiments show that both approaches for creating word embedding, i.e., Generic and Specific, provide similar results. Although, we obtain the overall best result using Specific Word Embedding, i.e., RCNN with Word2Vec trained with the Aggressive Text Preprocessing output.

The Word2Vec trained models obtain the overall best results. Although FastText has lower evaluation scores than Word2Vec, it outperforms GloVe on a series of models.

Figures 8, 9, and 10 present the comparison between Generic and Specific word Embeddings for each employed model when using Lemma Text Preprocessing. Figures 11, 12, and 13 present the comparison between Generic and Specific word Embeddings for each employed model when using Aggressive Text Preprocessing.

When comparing Generic vs. Specific Word Embeddings with Lemma Text Preprocessing, we obtain the following results. Regardless of the embedding, we obtain the highest accuracy with the RCNN model. Generic GloVe (Figure 8) and FastText (Figure 10) perform better than their Specific versions. For GloVe, we achieve small differences between the accuracy w.r.t. model. The LogReg and the attention-based models (i.e., textscAttr + RNN, Attr + GRU, Attr + LSTM) work better with Generic FastText. While, the simple Recurrent Networks (i.e., RNN, BiRNN, GRU, LSTM), the Convolution Networks (i.e., CNN, RCNN) and RLAN present higher performance when employing the Specific FastText Embedding. Specific Word2Vec (Figure 9) obtains a higher accuracy than Generic Word2Vec. We observe considerable differences in accuracy when combining Word2Vec with the Recurrent Networks (i.e., RNN, GRU, LSTM) and RLAN model.

We achieve similar patterns when comparing the Generic and Specific Embeddings for Aggressive Text Preprocessing. Generic GloVe (Figure 11) and FastText (Figure 13) obtain higher accuracy than their Specific conteparts. For these two embeddings, the highest accuracy is obtained with the RCNN model. For GloVe, we observe considerable differences in performance (i.e., $\sim$15%) when employing the LSTM model. FastText presents similar results as in the case of Lemma Text Preprocessing. Generic FastText obtains higher results

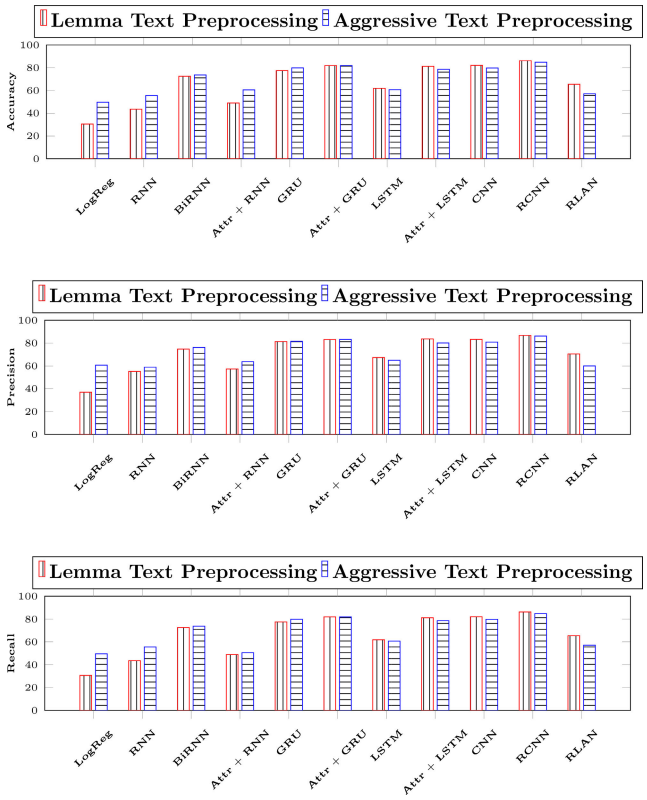

FIGURE 5. Lemma vs. aggressive text preprocessing using the specific GloVe embedding.

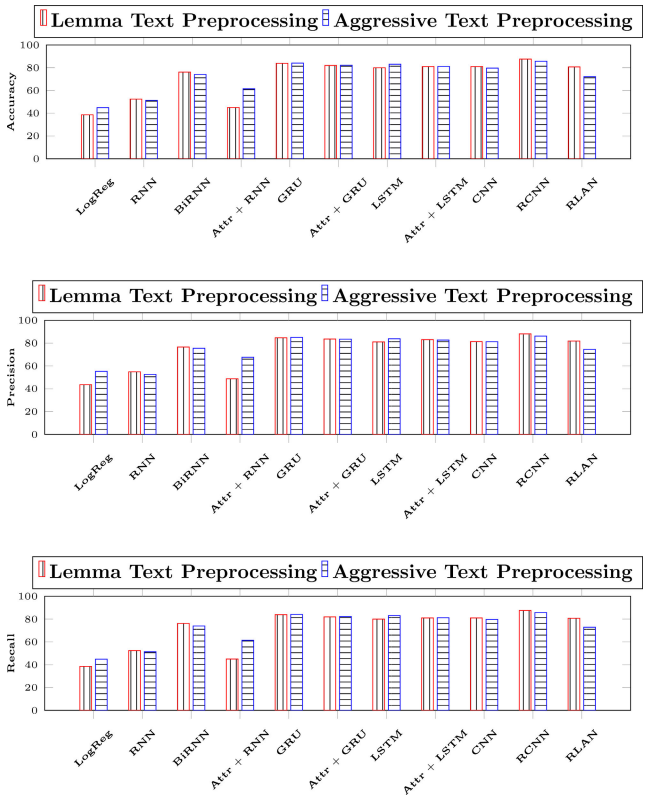

FIGURE 6. Lemma vs. aggressive text preprocessing using the specific Word2Vec embedding.

for LogReg, RNN and Attr + RNN models. While, Specific FastText perform better with LSTM and RLAN The highest accuracy for Specific Word2Vec (Figure 12) is achieved with RCNN, while Generic Word2Vec with Attr + GRU. The LogReg, RNN, Attr + GRU, and Attr + LSTM models obtain better results with the Generic Word2Vec. For Attr + LSTM, we observe similar results regardless of the embedding training mode. Furthermore, we observe high differences in performance between Generic and Specific Word2Vec for the LogReg, Attr + RNN, GRU, Attr + LSTM, and RLAN models.

In conclusion, for the Generic Embeddings, we obtain the highest accuracy for the pairs that use RCNN with the Generic GloVe and Generic FastText, regardless if we employ the Lemma or the Aggressive Text Preprocessing. The difference in accuracy range between 0.46% (for < RCNN, Lemma Text Preprocessing, Generic GloVe Embedding >) and 0.69 (for < RCNN, Aggressive Text Preprocessing, Generic FastText Embedding >). When using the Specific Embeddings, the highest accuracy is obtained for the pairs that employ Specific Word2Vec. Thus, the difference in accuracy range between 2.41% (for < *RCNN, Aggressive Text Preprocessing, Specific* Word2Vec *Embedding* >) and 2.84% (for < *RCNN, Lemma Text Preprocessing, Specific* Word2Vec *Embedding* >).

D. MACHINE vs. DEEP LEARNING ALGORITHMS

In the current literature, solutions for the fake news detection task include models using classical Machine Learning algorithms [16], [22], [27], as well as Deep Learning architectures [22], [24], [30], [34]. These approaches rarely use hyperparameter tuning or present the hyperparameters used for their architectures [16], [27], [30]. Furthermore, many neural network models are trained for a very small number of epochs, e.g., the authors in [22] train a CNN for 5 epochs and an LSTM for 10.

In our proposed architecture, we use Logistic Regression and multiple neural network models. Logistic Regression is used as a baseline for the misinformation detection task. For all our models, we employ an early stopping mechanism. Using hyperparameter tuning, we obtain the best values for both our Specific Word Embeddings and classification models. We note that grid search is an exhaustive process and should be finely tuned when searching for best hyperparameters values. The results show that the performance of RNN is greatly improved using either Bidirectional and Attention mechanisms. All three architectures achieve the best results using Specific Word Embeddings: RNN with GloVe, while BiRNN and RNN with an Attention mechanism with Word2Vec. The text preprocessing also has an impact on the models, RNN with and without the Attention mechanisms manages to produce better results using the Aggressive Text Preprocessing, while BiRNN achieves better results with the Lemma Text Preprocessing. We can observe that these networks learn better document representations when the word embeddings are trained on the specific dataset and that the

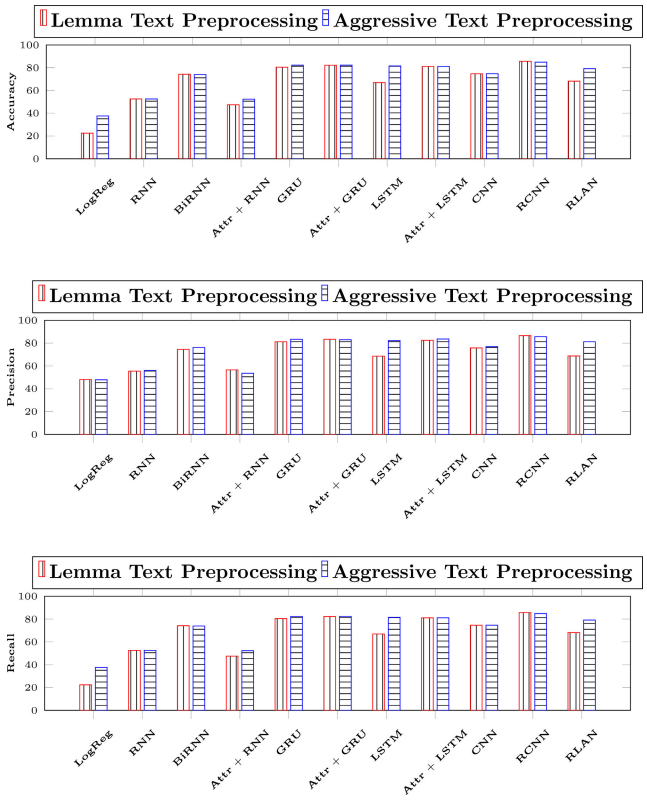

FIGURE 7. Lemma vs. aggressive text preprocessing using the specific FastText embedding.

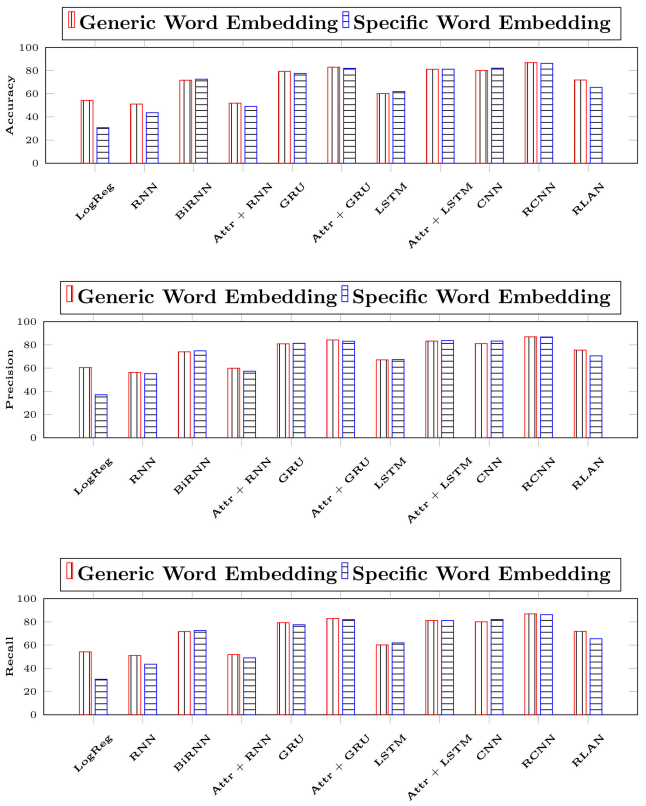

FIGURE 8. Generic vs. specific GloVe embeddings for lemma text preprocessing.

RNN model performs better when the word co-occurrence is encoded in the embedding. Thus, the word context is better preserved for the documents used. Furthermore, for Bidirectional networks, punctuation and stopwords seam to play an important role due to the nature of the network of analyzing the document form two directions, i.e., right to left and left to right.

Similarly to the RNN models, GRU and LSTM performance are improved using Attention mechanisms. For the LSTM and the GRU with and without an Attention mechanism, the best results are obtained using the Aggressive Text Preprocessing pipeline and Word2Vec. The best results are achieved by LSTM with an Attention mechanism using the Lemma Text Preprocessing Pipeline and FastText. Both architectures that use Attention obtain better results with the Generic Word Embeddings, while the other two with the Specific Word Embeddings. We can conclude the following. Firstly, the context added by the co-occurrence matrix brings no additional information to any of the networks. Thus, all these architectures require a word embedding that for a target word (in the case of Word2Vec) or and n-gram (in the case of FastText) manages to predict the source context. The results prove that complex RNN networks require the textual context to output quality models. Secondly, for complex recurrent networks, Specific Word Embeddings work better due to the context extracted from textual data that was stripped of any irrelevant information. Thirdly, when an Attention mechanism is employed, the models work better with pre-trained word embedding as they provide a wider and more complex representation of terms. This complex representation is needed by the Attention layer that minimizes the information loss by looking over all the information the original text holds and then generating the proper word according to the context of the current word it works on. Finally, for the LSTM with an Attention mechanism, the Lemma text enhances the information reacquired by the Attention layer to generate the proper words by adding into the context stopwords and punctuation.

For the CNN, RCNN, and RLANS models, the best results are obtained using the Lemma Text Preprocessing pipeline and Specific Word Embeddings.

The CNN model achieves the best results using the GloVe word embedding. We can observe that due to its convolution and MaxPooling mechanisms, the CNN architecture manages to learn better document representation when additional information, i.e., stopwords, punctuation, word co-occurrence, is stored within the embedding. Thus, the convolution operator manages to correctly store the word context and preserver document representation by creating an accurate classification model, although it significantly reduces the dimension of an embedding.

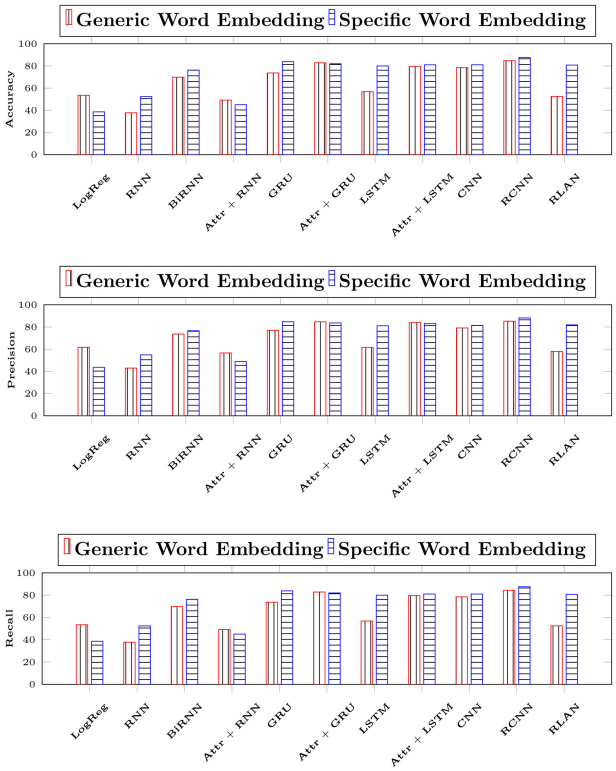

FIGURE 9. Generic vs. specific Word2Vec embeddings for lemma text preprocessing.

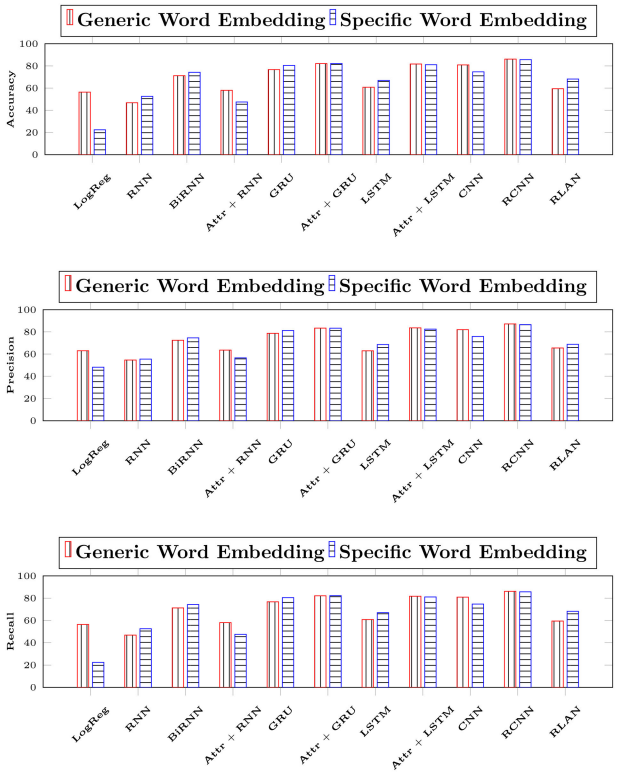

FIGURE 10. Generic vs. specific FastText embeddings for lemma text preprocessing.

The RCNN and RLANS architectures obtain the best results using the Word2Vec embedding. As in the case of the LSTM and GRU networks, we observe that these models produce better results when the word context is the target of the embedding model and the word embedding is not pre-trained. Furthermore, the models learn better document representations when additional information is preserved within the word embedding, i.e., stopwords and punctuation.

E. MISINFORMATION DETECTION

For the task of misinformation detection, the size of the dataset impacts the performance of Machine and Deep Learning algorithms. As the size of the dataset increases, the chances of overfitting decreases as the algorithms manage to learn better representations that improve generalization. Moreover, the size of a training dataset directly impacts the quality of the mapping function approximated by deep neural networks.

The size of the vocabulary is another important factor that impacts the misinformation detection task. Although some information is lost, by lemmatizing the words we observe that the models manage to differentiate to provide good results that differentiate between the types of misinformation. Also, by minimizing the vocabulary, the runtime of the Machine Learning algorithms decreases. We observe that depending on the model some networks learn better using aggressive text preprocessing while others learn better using specific text preprocessing. This behavior is highly influenced by the

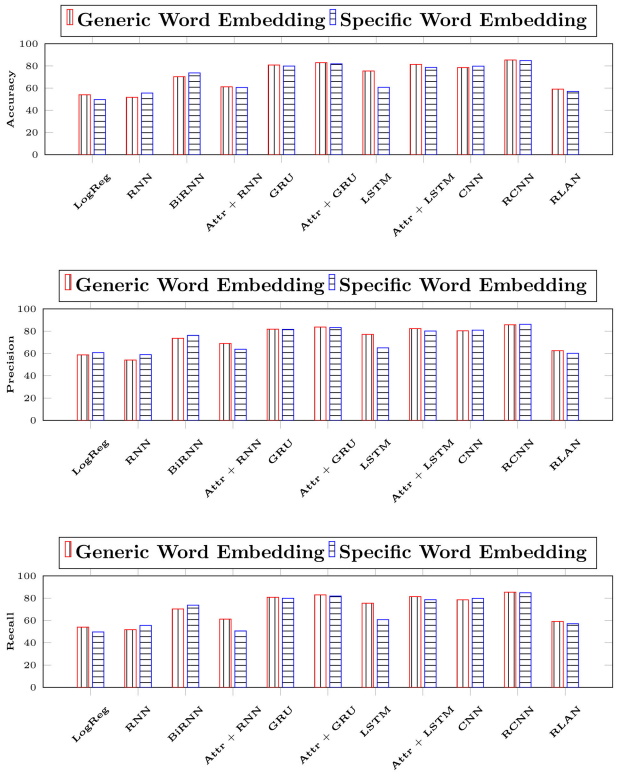

FIGURE 11. Generic vs. specific GloVe embeddings for aggressive text preprocessing.

activation functions used by cells as well as the different types of layers employed for developing the model.

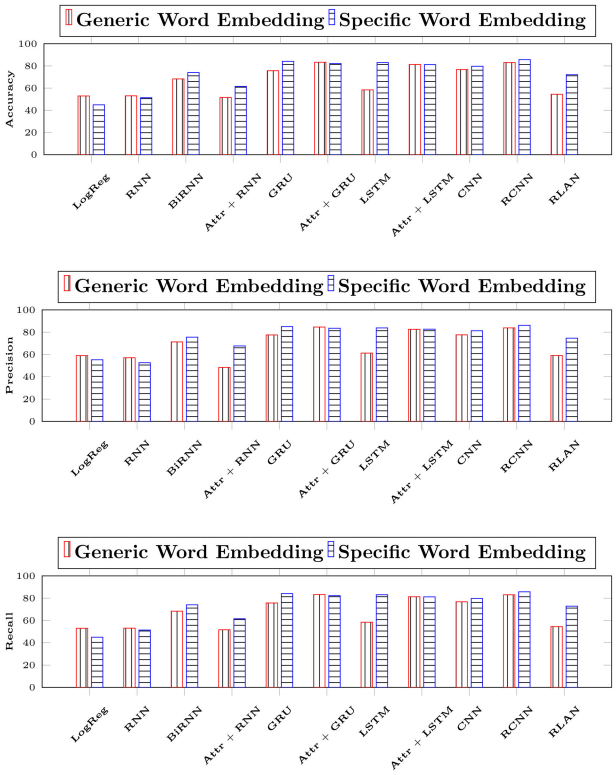

FIGURE 12. Generic vs. specific Word2Vec embeddings for aggressive text preprocessing.

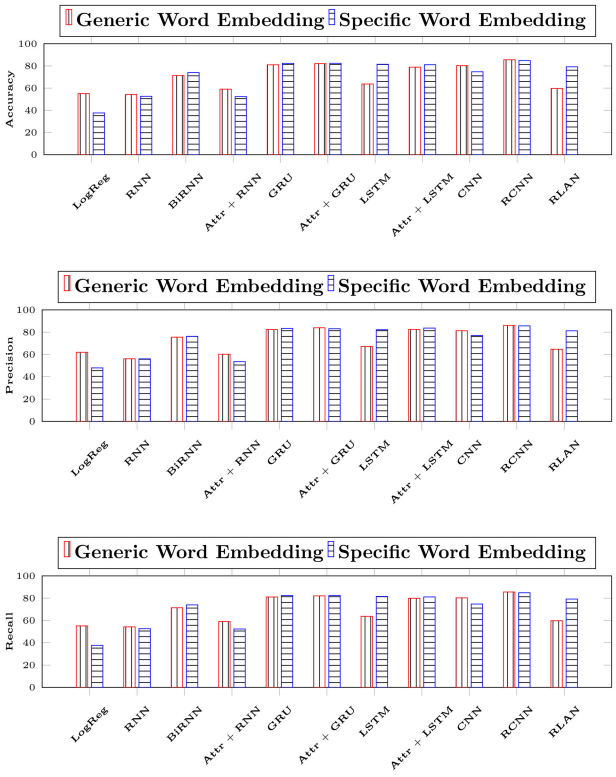

FIGURE 13. Generic vs. specific FastText embeddings for aggressive text preprocessing.

Linguistic features also play an important role [7] for misinformation detection, as different pieces of news use specific terms, e.g., science and technology related news articles, while others use more generic, e.g., political or click bate related articles. These local (at document-level) and global (at dataset-level) linguistic features are encapsulated together with semantics, syntactics, and context using word embeddings, although there is only one representation for a word using word embeddings. By using specific word embedding, we manage to better represent the misinformation linguistic features within the dataset. Using generic word embeddings these representations add information from the datasets they were trained on, datasets that are not collected for the task of misinformation.

The current Machine and Deep Learning methods employed do far for fake news detection have shown promising results. Although these methods obtain high accuracy for the misinformation detection problem using either a binary approach or levels of veracity, there is still a place for improvements when dealing with multi-class misinformation detection. Firstly, hyperparameters chose to solve binary approaches do not work well for multi-class ones [8]. Thus, the hyperparameters determined to discriminate between two classes or between classes with different veracity levels might be over-represented in the classification task. This can lead to detecting the wrong discriminative features used in the case of multi-class misinformation classification. Secondly, the features used for binary approaches or different veracity levels do not work as well for multi-class approaches. If articles from different types of fake news are group together under the same class, some important features that are representative for some labels get underrepresented when building models that use only one label for the entire group. Lastly, Machine Learning algorithms that are trained on small binary datasets with multiple features tend to overfit instead of creating generalized models [12]. By adding multiple labels, overfitting is minimized.

VII. CONCLUSION

In this paper, we present our findings regarding Context-Aware Misinformation Detection using Deep Learning Architectures. We employ two text preprocessing pipelines (i.e., Lemma and Aggressive Text Preprocessing), three context-aware word embeddings (i.e., Word2Vec, FastText, and GloVe), and ten Neural Networks for performing multi-class classification. The context-aware word embeddings are either pre-trained (Generic Word Embeddings) or custom trained on our dataset (Specific Word Embeddings). We propose a preprocessing and classification pipeline based on our findings. The dataset used for the experimental validation contains 100 000 news articles labeled either as real or with different types of fake news (e.g., Conspiracy Theory, Junk Science, Hate News, etc.). Furthermore, we provide a thorough discussion of our results and an extended analysis of the performance of the employed classification algorithms w.r.t. text preprocessing and word embedding training approach and model.

Our benchmark experiments showed that each Deep Learning model, except the unidirectional RNN, performed better than Logistic Regression, which is used as a baseline. Moreover, it seems that the LSTM with Attention provides the best results in the set of recurrent models. CNN performed very well on its own, proving that the global context of the words is incorporated, i.e., the word co-occurrence matrix, in the misinformation classification task. This result confirms that CNN architectures obtain improved results when the global and local word contexts are encoded within the embedding.

The experimental results obtained by the benchmark show that most models have an increase in accuracy when using custom trained word vectors on our dataset. Word2Vec Skip-Gram model manages to better encode the local context when the Specific Word Embeddings are used. The models that use this embedding show a constant increases in accuracy. LSTM presents the highest increase in accuracy, with almost 20% when trained with the Specific over the Generic Word Embedding. We conclude that the performance gain of the neural networks when using the custom trained Word2Vec Skip-Gram model manage to learn an improved representation of documents for the classification task using local context. Thus, the misinformation detection performance is improved when there is no influence from outside information as the embedding better captures the word context within the dataset.

When comparing the performance of the models w.r.t. text preprocessing, we observe that the overall behavior is similar. The Word2Vec still benefits the most from changing the Generic Word Embeddings to Specific ones. The RCNN obtains the highest accuracy of all the models, regardless of the preprocessing. However, there are some differences between the two preprocessing pipelines. The results show that the task is simplified for Logistic Regression when stopwords and punctuation are removed, but the accuracy drops for the models that use convolution windows, like the CNN and the RCNN. We conclude that the context window extracts useful information from punctuation and stopwords.

Overall, the RCNN model with Specific Word2Vec Word Embeddings trained on the Lemma Text Preprocessing obtain the overall best results, i.e., the accuracy, precision, and recall are 87.56%, 87.55%, and 88.11%, respectively. We conclude that these results are obtained because of the local word context preserved within the Word2Vec embedding and of the added sentence context within the hidden layers of the network.

In the context of misinformation detection, it is also worth investigating other quantitative and qualitative methods for building textual features, such as sentiment analysis [15] or topic modeling [19]. These approaches can be used to better understand the dataset and determine if any bias in discriminating between different types of misinformation types is added through the integration of such features in the neural models.

One direction that shows promise for future research is to add an Attention mechanism to the bidirectional LSTM from the RCNN model, combining the improvements brought on by Attention mechanisms with the performance of the RCNN on its own. Also, we believe that better hyperparameter calibration is required so that the RLANS increases its results. Furthermore, we plan to create heatmaps for the attention-based models and analyze the output of the convolutional layers for CNNs to enhance the results' interpretability.

Another future direction would be to analyze how new embeddings perform within our architecture, e.g., Misspelling Oblivious Word Embeddings (MOE) [37], which shows great promise in the case of data with misspelled or Mittens [11], which is an extension of GloVe that learns domain-specialized representations. Furthermore, we plan to use transformers (e.g., BERT [10], XLNet [48], etc.) to determine if the accuracy of misinformation detection can be improved.

ACKNOWLEDGMENT

(Vlad-Iulian Ilie, Ciprian-Octavian Truică, and Elena-Simona Apostol contributed equally to this work.)

REFERENCES

[1] O. Ajao, D. Bhowmik, and S. Zargari, "Fake news identification on Twitter with hybrid CNN and RNN models," in *Proc. 9th Int. Conf. Social Media Soc.*, Jul. 2018, pp. 226–230.

[2] I. Assent, "Clustering high dimensional data," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 2, no. 4, pp. 340–350, 2012.

[3] M. T. Bastos and D. Mercea, "The Brexit BotNet and user-generated hyperpartisan news," *Social Sci. Comput. Rev.*, vol. 37, no. 1, pp. 38–54, Feb. 2019.

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.

[5] A. Bovet and H. A. Makse, "Influence of fake news in Twitter during the 2016 US presidential election," *Nature Commun.*, vol. 10, no. 1, pp. 1–14, Dec. 2019.

[6] D. P. Calvillo, B. J. Ross, R. J. B. Garcia, T. J. Smelter, and A. M. Rutchick, "Political ideology predicts perceptions of the threat of COVID-19 (and susceptibility to fake news about it)," *Social Psychol. Personality Sci.*, vol. 11, no. 8, pp. 1119–1128, Nov. 2020.

[7] A. Choudhary and A. Arora, "Linguistic feature based learning model for fake news detection and classification," *Expert Syst. Appl.*, vol. 169, May 2021, Art. no. 114171.

[8] G. Collell, D. Prelec, and K. R. Patil, "A simple plug-in bagging ensemble based on threshold-moving for classifying binary and multiclass imbalanced data," *Neurocomputing*, vol. 275, pp. 330–340, Jan. 2018.

[9] N. J. Conroy, V. L. Rubin, and Y. Chen, "Automatic deception detection: Methods for finding fake news," in *Proc. 78th ASIST Annu. Meeting, Inf. Sci. Impact, Res. Community*, vol. 52, no. 1, pp. 1–4, 2015.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019, pp. 4171–4186.

[11] N. Dingwall and C. Potts, "Mittens: An extension of GloVe for learning domain-specialized representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol., (Short Papers)*, vol. 2, 2018, pp. 212–217.

[12] V. Feldman, R. Frostig, and M. Hardt, "The advantages of multiple classes for reducing overfitting from test set reuse," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1892–1900.

[13] A. E. A. Gautam, "Fake news detection system using XLNet model with topic distributions," in *Proc. CONSTRAINT Shared Task (AAAI)*, 2021, pp. 189–200.

[14] A. Gelfert, "Fake news: A definition," *Informal Log.*, vol. 38, no. 1, pp. 84–117, Mar. 2018.

[15] B. Ghanem, P. Rosso, and F. Rangel, "An emotional analysis of false information in social media and news articles," *ACM Trans. Internet Technol.*, vol. 20, no. 2, pp. 1–18, May 2020.
[16] G. Gravanis, A. Vakali, K. Diamantaras, and P. Karadais, "Behind the cues: A benchmarking study for fake news detection," *Expert Syst. Appl.*, vol. 128, pp. 201–213, Aug. 2019.
[17] M. Hardalov, I. Koychev, and P. Nakov, "In search of credible news," in *Proc. Int. Conf. Artif. Intell., Methodol., Syst., Appl.*, 2016, pp. 172–180.
[18] J. Hartmann, J. Huppertz, C. Schamp, and M. Heitmann, "Comparing automated text classification methods," *Int. J. Res. Marketing*, vol. 36, no. 1, pp. 20–38, 2019.
[19] S. Helmstetter and H. Paulheim, "Weakly supervised learning for fake news detection on Twitter," in *Proc. IEEE/ACM Int. Conf. Adv. Social Netw. Anal. Mining (ASONAM)*, Aug. 2018, pp. 274–277.
[20] K. Higgins, "Post-truth: A guide for the perplexed," *Nature*, vol. 540, no. 7631, p. 9, Dec. 2016.
[21] J. Hua and R. Shaw, "Corona virus (COVID-19) 'infodemic' and emerging issues through a data lens: The case of China," *Int. J. Environ. Res. Public Health*, vol. 17, no. 7, p. 2309, Mar. 2020.
[22] R. K. Kaliyar, A. Goswami, P. Narang, and S. Sinha, "FNDNet—A deep convolutional neural network for fake news detection," *Cognit. Syst. Res.*, vol. 61, pp. 32–44, Jun. 2020.
[23] J. Y. Khan, M. T. I. Khondaker, S. Afroz, G. Uddin, and A. Iqbal, "A benchmark study of machine learning models for online fake news detection," *Mach. Learn. With Appl.*, vol. 4, Jun. 2021, Art. no. 100032.
[24] S. Kula, M. Choraś, and R. Kozik, "Application of the bert-based architecture in fake news detection," in *Proc. Conf. Complex, Intell., Softw. Intensive Syst.*, Burgos, Spain. Cham, Switzerland: Springer, 2020, pp. 239–249.
[25] L. Kurasinski and R.-C. Mihailescu, "Towards machine learning explainability in text classification for fake news detection," in *Proc. 19th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2020, pp. 775–781.
[26] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Proc. AAAI*, vol. 333, 2015, pp. 2267–2273.
[27] Q. Li, Q. Hu, Y. Lu, Y. Yang, and J. Cheng, "Multi-level word features based on CNN for fake news detection in cultural communication," *Pers. Ubiquitous Comput.*, vol. 24, no. 2, pp. 259–272, 2019.
[28] Y. Li and J. Ye, "Learning adversarial networks for semi-supervised text classification via policy gradient," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 1715–1723.
[29] T. Linzen, "Issues in evaluating semantic spaces using word analogies," in *Proc. 1st Workshop Evaluating Vector-Space Represent. (NLP)*, 2016, pp. 13–18.
[30] C. Liu, X. Wu, M. Yu, G. Li, J. Jiang, W. Huang, and X. Lu, "A two-stage model based on bert for short fake news detection," in *Proc. Int. Conf. Knowl. Sci., Eng. Manage.*, Athens, Greece. Cham, Switzerland: Springer, 2019, pp. 172–183.
[31] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019, *arXiv:1907.11692*.
[32] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Represent.*, 2013, pp. 1–12.
[33] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proc. Int. Conf. Lang. Resour. Eval.*, 2018, pp. 52–55.
[34] R. Mishra and V. Setty, "SADHAN: Hierarchical attention networks to learn latent aspect embeddings for fake news detection," in *Proc. ACM SIGIR Int. Conf. Theory Inf. Retr.*, Sep. 2019, pp. 197–204.
[35] T. Murayama, S. Wakamiya, and E. Aramaki, "Fake news detection using temporal features extracted via point process," 2020, *arXiv:2007.14013*.
[36] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1532–1543.
[37] A. Piktus, N. B. Edizel, P. Bojanowski, E. Grave, R. Ferreira, and F. Silvestri, "Misspelling oblivious word embeddings," in *Proc. Conf. North*, 2019, pp. 3226–3234.
[38] P. Przybyla, "Capturing the style of fake news," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 490–497.
[39] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 842–866, Jan. 2020.
[40] J. Roozenbeek and S. van der Linden, "The fake news game: Actively inoculating against the risk of misinformation," *J. Risk Res.*, vol. 22, no. 5, pp. 570–580, May 2019.
[41] V. Rubin, N. Conroy, Y. Chen, and S. Cornwell, "Fake news or truth? Using satirical cues to detect potentially misleading news," in *Proc. 2nd Workshop Comput. Approaches Deception Detection*, 2016, pp. 7–17.
[42] D. Ruths, "The misinformation machine," *Science*, vol. 363, no. 6425, p. 348, 2019.
[43] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu, "Fake news detection on social media: A data mining perspective," *ACM SIGKDD Explor. Newslett.*, vol. 19, no. 1, pp. 22–36, Sep. 2017.
[44] K. Shu, S. Wang, and H. Liu, "Beyond news contents: The role of social context for fake news detection," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 312–320.
[45] K. Shu, G. Zheng, Y. Li, S. Mukherjee, A. Hassan Awadallah, S. Ruston, and H. Liu, "Leveraging multi-source weak social supervision for early detection of fake news," 2020, *arXiv:2004.01732*.
[46] D. Teoh, "The power of social media for HPV vaccination–not fake news," *Amer. Soc. Clin. Oncol. Educ. Book*, vol. 39, pp. 75–78, May 2019.
[47] M. Umer, Z. Imtiaz, S. Ullah, A. Mehmood, G. S. Choi, and B.-W. On, "Fake news stance detection using deep learning architecture (CNN-LSTM)," *IEEE Access*, vol. 8, pp. 156695–156706, 2020.
[48] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized autoregressive pretraining for language understanding," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5753–5763.
[49] D. Yi, S. Ji, and S. Bu, "An enhanced optimization scheme based on gradient descent methods for machine learning," *Symmetry*, vol. 11, no. 7, p. 942, Jul. 2019.
[50] S. Zannettou, M. Sirivianos, J. Blackburn, and N. Kourtellis, "The web of false information," *J. Data Inf. Qual.*, vol. 11, no. 3, pp. 1–37, Jul. 2019.
[51] Y. Zhang, F. Liu, Y. H. Koura, and H. Wang, "Analysing rumours spreading considering self-purification mechanism," *Connection Sci.*, vol. 33, no. 1, pp. 81–94, 2020.
[52] X. Zhou, R. Zafarani, K. Shu, and H. Liu, "Fake news: Fundamental theories, detection strategies and challenges," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Jan. 2019, pp. 836–837.

VLAD-IULIAN ILIE received the bachelor's degree in automatic control and the master's degree in financial computing from the University POLITEHNICA of Bucharest, Romania, in 2018 and 2020, respectively, where he is currently pursuing the Ph.D. degree in distributed machine and deep learning. His master's thesis was on the subject of fake news detection using deep learning methods. His research interests include deep learning, distributed deep learning, and federated machine learning and aims that, through his research, he will bring value in tackling the current challenges in artificial intelligence.

CIPRIAN-OCTAVIAN TRUICĂ received the B.Sc. degree in computer science and electrical engineering, the M.Sc. degree in computer science engineering and information technology, and the Ph.D. degree in data management and text mining from the University POLITEHNICA of Bucharest, Romania, in 2011, 2013, and 2018, respectively, and the B.Sc. degree in computer science and mathematics from the University of Bucharest, in 2013. During his Ph.D. studies, he was an Invited Researcher at the ERIC laboratory, Université de Lyon, France, in 2015 and 2016, where he worked on data management, machine learning, and natural language processing. He was a Postdoctoral Researcher with the Data-Intensive Systems Group, Department of Computer Science, Aarhus University, Aarhus, Denmark, from 2019 to 2020, where he worked on big data analytics for time series. He is an Assistant Professor of computer science at the Computer Science and Engineering Department, Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest. His research interests include big data, data management, machine learning, text mining, natural language processing, and time series analysis.

ELENA-SIMONA APOSTOL received the Ph.D. degree from the University POLITEHNICA of Bucharest (UPB), Romania, in 2014. During her bachelor's and master's studies, she was an Intern and a Junior Research Engineer at the Fraunhofer FOKUS Institute, Berlin, Germany, where she worked on computer networking and telecommunications with a focus on mobile and service orientated architectures. She was an Invited Researcher during her Ph.D. studies at INRIA Rennes, France, working within the joint research team between KerData at INRIA and UPB on big data management and analytics. She was a Postdoctoral Researcher at the Microsoft Research Center, Paris, in collaboration with The French Institute for Research in Computer Science and Automation (INRIA), where she worked on state-of-the-art big data analysis, multi-site cloud computing, and bioinformatics. She is an Associate Professor of computer science at the Computer Science and Engineering Department, Faculty of Automatic Control and Computers, UPB. Her research interests include big data, data management, parallel and distributed algorithms, machine learning, and data science.

ADRIAN PASCHKE is the Head of the Corporate Semantic Web Group (AG-CSW) and the Chair of semantic data intelligence at the Department of Mathematics and Computer Science, Institute of Computer Science, Freie Universität Berlin (FUB). He is also the Director of the Data Analytics Center (DANA), Fraunhofer Institute for Open Communication Systems (FOKUS), the Director of RuleML Inc., Canada, a professional member at the Einstein Center Digital Future (ECDF), the Dahlem Center for Machine Learning and Robotics (DCMLR), and the Institut für Angewandte Informatik (InfAI), Leipzig University, and the Founder of the Berlin Semantic Web Meetup Group. With over 200 peer-reviewed scientific publications, he has made substantial scientific contributions in the field of semantic AI research and is active in standardization of semantic technologies, such as OASIS LegalRuleML, RuleML, OMG API4KB, W3C Semantic Web—W3C Rule Interchange Format, and W3C RDF Stream Processing. He also served as an expert for industry and several ministries and funding bodies, including the European Commission. He was an Organizer and the Chair of renowned conferences and workshops, including DEBS, RuleML, BIS, SWAT4HCLS, ODBASE, ESWC, Reasoning Web, Semantics, and edBPM, and an invited speaker for keynotes, tutorials, panels, and lectures.