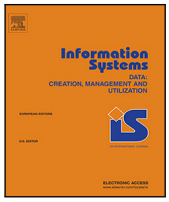
Information Systems 121 (2024) 102342

Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



Is text preprocessing still worth the time? A comparative survey on the influence of popular preprocessing methods on Transformers and traditional classifiers

Marco Siino *, Ilenia Tinnirello, Marco La Cascia

University of Palermo, Department of Engineering, Palermo, 90128 PA, Italy

ARTICLE INFO

Keywords:
Text preprocessing
Natural Language Processing
Fake news
SVM
Bayes
Transformers
Deep learning
LSTM
Convolutional neural networks

ABSTRACT

With the advent of the modern pre-trained Transformers, the text preprocessing has started to be neglected and not specifically addressed in recent NLP literature. However, both from a linguistic and from a computer science point of view, we believe that even when using modern Transformers, text preprocessing can significantly impact on the performance of a classification model. We want to investigate and compare, through this study, how preprocessing impacts on the Text Classification (TC) performance of modern and traditional classification models. We report and discuss the preprocessing techniques found in the literature and their most recent variants or applications to address TC tasks in different domains. In order to assess how much the preprocessing affects classification performance, we apply the three top referenced preprocessing techniques (alone or in combination) to four publicly available datasets from different domains. Then, nine machine learning models – including modern Transformers – get the preprocessed text as input. The results presented show that an educated choice on the text preprocessing strategy to employ should be based on the task as well as on the model considered. Outcomes in this survey show that choosing the best preprocessing technique – in place of the worst – can significantly improve accuracy on the classification (up to 25%, as in the case of an XLNet on the IMDB dataset). In some cases, by means of a suitable preprocessing strategy, even a simple Naïve Bayes classifier proved to outperform (i.e., by 2% in accuracy) the best performing Transformer. We found that Transformers and traditional models exhibit a higher impact of the preprocessing on the TC performance. Our main findings are: (1) also on modern pre-trained language models, preprocessing can affect performance, depending on the datasets and on the preprocessing technique or combination of techniques used, (2) in some cases, using a proper preprocessing strategy, simple models can outperform Transformers on TC tasks, (3) similar classes of models exhibit similar level of sensitivity to text preprocessing.

1. Introduction

Tasks related to Natural Language Processing (NLP) usually consist of preprocessing, tokenization, and several possible stages like classification, machine translation or, from a more generic perspective, understanding human languages [1]. The preprocessing step involves operations like lowercasing, stemming, stop words removal and many other [2] collected and presented in this work. Specifically, preprocessing can involve deleting content that is unnecessary for some tasks (such as removing stop words and non-alphabetic characters), merging semantically similar words to increase prediction power and decrease data sparsity (using stemming, lemmatizing, conversion to lowercase, expanding abbreviations, correcting misspellings), and enhancing the quantity of semantic information available (e.g., using strategies to manage negation words) [3]. This implies that preprocessing can potentially delete important data [4] (such as deleting stop words when they are pertinent to a particular study issue). Furthermore, several errors can be introduced into the specific NLP task's pipeline. For instance, when semantically distinct words conflate using stemming, the outcomes of a classification model can significantly change [5]. Another common issues when dealing with online text is the presence of noise due to spelling and grammar errors. This type of noise can be useful when profiling authors. In fact, an author's style can include similar misspelling and grammar errors useful for a profiling task based on Text Classification (TC). However, it could also lead to classification errors when the content to be classified is a single news article or a single tweet. In these cases – especially when employing pre-trained Transformers – misspelling and grammar errors could just alter the semantic of a word, misleading a classification model.

* Corresponding author.
E-mail address: marco.siino@unipa.it (M. Siino).

https://doi.org/10.1016/j.is.2023.102342
Received 3 March 2023; Received in revised form 21 July 2023; Accepted 21 December 2023
Available online 23 December 2023

Despite its importance, the text preprocessing stage is often underestimated in several text mining studies found in the literature [6]. However, there is a substantial quantity of noise in unstructured texts available on the internet [7]. In some cases, the amount of noise in a dataset can be so high that it could easily mislead a classifier [8]. The presence of noise could be caused by users who frequently utilize slangs and/or acronyms, as well as making spelling and grammar mistakes. To emphasize their emotions, users may also abuse punctuation marks [9]. For example, typing multiple exclamation marks instead of a single one. In this context, our definition of noise is related to any useless information for any text-based task to be performed after preprocessing a dataset. As demonstrated in this work, an incorrect choice to preprocess a text can lead to a difference of over the 25% in terms of accuracy in the classification performance, with the same model and dataset being used.

Preprocessing, in this sense, can be summed up as the process of cleaning and preparing texts that will be tokenized for subsequent operations. Thus, the necessity for data cleaning and normalization arises because the effectiveness of a model employed after the preprocessing stage depends critically on the quality of such data [10]. The crucial role that preprocessing has before and throughout the feature selection process, from our point of view, is of prominent importance and to be widely noted. Eventually, a preprocessing stage does not just remove noise and/or highlights important features, but also reduce the time required for training and testing a model. Unfortunately, past research has provided conflicting recommendations, mainly due to the datasets used, the techniques applied and/or the models evaluated.

In the literature there is no convention adopted, and each work tests some preprocessing techniques rather than others. In this work we want to report and discuss these techniques and subsequently, for the most commonly used, we evaluate the results obtained by their combination for TC tasks with respect to the model and the dataset considered. Our work aims at improving the text preparation stage and resolving inconsistencies in preprocessing advices, to offer guidelines and ideas for future text mining studies. We want to improve our comprehension of the theoretical and empirical factors that should influence preprocessing choices. Here we want to investigate how TC performance is affected by preprocessing choices using traditional Non-Deep Learning models (NDL), Deep Learning models (DL) and Transformers [11]. For a more consistent investigation, we conduct all of our experiments using four datasets from different domains. We can finally state that it is important to make an educated and context-dependent choice about which preprocessing method (or combination of methods) to employ and in what order.

All the results and the code used for our experimental evaluation are available on GitHub.[1]

1.1. Gaps in the literature

In this subsection, we briefly introduce some of the most referenced and comprehensive surveys reported in the literature about text preprocessing. A more in-depth discussion that also includes the most recent and relevant studies is provided in Section 2 concerning the related work. We conclude this subsection highlighting the gaps found in the literature.

In [12], the authors analyze the preprocessing impact on Twitter data, emphasizing how much the classifier performance is improved. They deleted URLs, user mentions, stop words, hashtags, punctuation and then they used n-grams to replace slang words with the corresponding regular words. The suggested preprocessing method binds slangs on already existing words to assess the meaning and sentiment interpretation of the slangs. The authors employ an SVM classifier and conclude their study wondering how effectively the suggested system would work with different classifiers on other text streams. Involving four conventional classifiers and a neural network in their experiments, the authors in [13] investigate how each preprocessing technique affects the performance of the models, using solely TF-IDF (unigram) to represent words. The authors demonstrate that while deleting punctuation does not improve the classification performance, preprocessing procedures like removing digits, expanding contractions to base words and lemmatization do. Additionally, their study shows how a small number of various preprocessing strategies interact reciprocally and highlights those that work best when combined. However, the authors conclude their article with an open question for future studies that could eventually test the preprocessing techniques employed also on datasets from different domains, such as news articles and product or movie reviews. In [14], the authors analyze twelve different preprocessing techniques on three datasets. The datasets are built from Twitter and focus on hate speech detection. The authors observe the impact of the preprocessing techniques on the classification tasks they support. However, they do not fully explore all the possible combination of the preprocessing techniques proposed but, after an inference process, a subset of all the combinations is considered. In fact, the authors suggest that future research should examine the impact of these and other preprocessing strategies in various domains, as well as other preprocessing technique combinations and their interactions.

Taking into account the above-mentioned studies and those discussed in Section 2, some areas regarding text preprocessing are outdated, still unexplored or under-explored. To summarize, the works described above or referenced in the following sections are characterized by at least one or more of the following aspects:

- Do not contain a detailed catalog of all the most common preprocessing techniques. Usually only a subset of all the available techniques is reported.
- More in-depth experimental evaluations on Transformers and on DL models are missing.
- There is a lack of experimental evaluations on models that can truly achieve valuable state-of-the-art results.
- One single task is addressed and/or a single preprocessing technique is evaluated.
- Similar datasets (e.g., similar text format for any sample) or datasets from the same domain are employed.
- There is not a clear explanation on why a subset of certain combination of preprocessing techniques is evaluated.

With our work we hope to better investigate the matter without neglecting any aspects or point of view reported above.

1.2. Research questions

In this subsection we list the research questions (RQs) addressed in our evaluative survey to fill the gaps brought to attention in the previous subsection. This is the first study, to the best of our knowledge, that addresses all of the following RQs as a whole and at the same time.

- **RQ1:** After collecting and discussing the text preprocessing techniques found in the literature and their possible and most recent variants, what are the outcomes of an evaluation of the impact of each of these techniques (alone or in combination) on the classification performance of state-of-the-art and traditional models using real world datasets from different domains?
- **RQ2:** How text preprocessing can affect the performance of modern pre-trained architectures based on attention (i.e., Transformers)?
- **RQ3:** Is the performance of simple classifiers comparable to the performance of Transformer-based models when text preprocessing is performed in accordance with the specific model and/or dataset used?

[1] https://github.com/marco-siino/text_preprocessing_impact

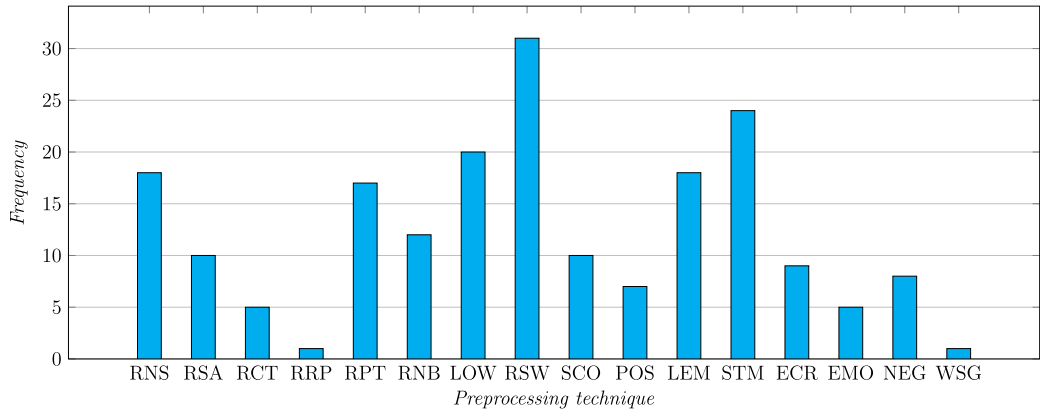


Fig. 1. Number of times that the techniques discussed in this article are found in related work. In Table 1 are reported the expanded versions of the acronyms under the bars.

1.3. Article organization

Our article is structured as follows. The recent literature on the impact of preprocessing techniques is discussed in Section 2, in Section 3 a complete discussion of the preprocessing techniques collected is presented, in Section 4 a presentation of the models employed in this study is provided, then we discuss the datasets and the procedure for the experimental evaluation, in Section 5 the results of our experiments are reported for each dataset and in Section 6 they are discussed to report common findings across the three datasets and to answer to our research questions. Finally, in Section 7 our conclusions and future work ideas are provided.

2. Related work

We report in this section the results of some of the most relevant and recent studies employing text preprocessing techniques. The following are those that, in addition to employ preprocessing techniques, have also carried out a comparative evaluation using one or more models and/or datasets. For a detailed discussion on the preprocessing techniques and the corresponding related work, please refer to Section 3.

Recently, the authors in [15] use a variety of deep neural architectures – except Transformers – to examine the impact of preprocessing on a pre-trained BERT model when fine-tuning it as the first embedding layer. The authors find that text preprocessing had negligible influence on the majority of models tested. It is worth mention that authors conduct the study on a single Indonesian dataset containing 3,217 instances from the Water Resources Agency of Jakarta, to classify the textual reports into five categories (i.e., drain closure, waterways, flood mitigation, infiltration well and others). The authors use an Indonesian pre-trained version of BERT for the embedding. Since there were substantial changes in performance outcomes between the model with and without text preprocessing, the authors suggest that future studies should examine the impact of each text preprocessing step. In this sense, to investigate the effects of different preprocessing techniques, authors in [16] make use of fourteen text preprocessing approaches that have been applied to datasets from Twitter, Facebook, and YouTube. The authors use text preprocessing algorithms in a particular order. In the study, the authors use SVM to assess the variation in terms of accuracy on sentiment classification employing the preprocessing strategies proposed. Results show that by consistently utilizing all the preprocessing approaches, it is possible to reach the 82.57% accuracy using unigram representations. Even if the proposed preprocessing strategy proved to be effective on the selected dataset, an in-depth investigation employing deep learning models misses. The performance of an SVM classifier is also evaluated in [17] on a Twitter dataset for sentiment classification (i.e. the Stanford Twitter Sentiment Dataset). The authors explore some of the combinations of the preprocessing techniques proposed. The researchers discovered that the use of URL features reservation, repeated letters normalization and negation transformation increases the accuracy of sentiment classification. Instead, the accuracy descends if stemming and lemmatization are used. Furthermore, by adding bigrams and emotion features to the initial feature space, a superior outcome is obtained. Also in [18], the authors employ NDL models like Naïve Bayes, SVM, K-means and Fuzzy logic algorithms. Specifically, on a Twitter dataset, three basic preprocessing methods (i.e., tokenization, removing of stop words and stemming) are explored. The findings indicate that preprocessing has a relevant impact on reducing the dimensionality of data, which leads to higher performance in sentiment analysis classification tasks. Also for unstructured product review data, the authors in [19] demonstrate that the correctness of a classifier prediction depends on a suitable text preprocessing sequence. The records in the dataset used for training were made up of product reviews from Amazon. To assign a binary output label (positive or negative) to each sample, ratings of one or two stars are collapsed into the class of negative reviews. Ratings of four or five have been classified as positive. Also in this study, authors employ NDL models (namely, Naïve Bayes, Decision Tree and SVM). Four NDL classifiers (i.e., Naïve Bayes, Logistic Regression, SVM and Random Forest) are also employed in [20], where authors explore the impact of six preprocessing techniques using five different Twitter datasets. They discovered that by utilizing the preprocessing techniques of extending acronyms and substituting negation, as opposed to eliminating URLs, removing numerals, or removing stop words, the classification results, in terms of F1-measure and accuracy, are enhanced. The Transformers are used in [21] where, before applying TFIDF, authors remove stop words and keep only features appearing in, at least, two documents. The experimental findings show that in the smaller datasets, the shallow and most straightforward non-neural methods achieve some of the best results. On the other hand, Transformers perform better in terms of classification performance in the larger datasets. However, the study marginally focuses on the impact of text preprocessing.

Regarding a Twitter related task about irony detection, authors in [22] perform a case-folding preprocess of tweets before tokenizing with the TokTokTokenizer from NLTK. Then, generic labels replace hashtags, user mentions and URLs (i.e., *hashtag*, *user* and *url*, respectively). Then, elongated words are shortened, to limit a vowel to only appear twice in a token after each other (e.g., yeeee mapped into yee). While the authors employed BERT as the classification model, they only use the preprocessing strategy discussed above. Also authors in [23] introduce and apply a new preprocessing strategy based on three new steps (i.e., lowering dimensionality, rising sparseness and reducing the

Table 1
Acronyms for the preprocessing techniques and real case examples, raw and preprocessed.

Acronym	Technique	Raw	Preprocessed
DON	Do Nothing	"Like a Rolling Stone"	"Like a Rolling Stone"
RNS	Replace Noise	*"@Obama tells #metoo! bit.ly/–"*	*"USER tells HASHTAG! URL"*
RSA	Replace Slang/Abbreviations	*"omg you are so nice!"*	*"Oh my God you are so nice!"*
RCT	Replace Contraction	*"I don't like butterflies."*	*"I do not like butterflies."*
RRP	Remove Repeated Punctuation	*"I like her!!!"*	*"I like her multiExclamation"*
RPT	Removing Punctuation	*"You. are. cool."*	*"You are cool"*
RNB	Remove Numbers	*"You are gr8."*	*"You are gr."*
LOW	Lowercasing	*"You Rock! YEAH!"*	*"you rock! yeah!"*
RSW	Remove Stop Words	*"This is nice"*	*"is nice"*
SCO	Spelling Correction	*"1lenia is so kind!"*	*"Ilenia is so kind!"*
POS	Part-of-Speech Tagging	*"Kim likes you"*	*"Kim (PN) likes (VB) you (N)"*
LEM	Lemmatization	*"I am going to shopping"*	*"I be go to shop"*
STM	Stemming	*"Girl's shirt with different colors"*	*"Girl shirt with differ color"*
ECR	Remove Elongation	*"You are cooool!"*	*"You are cool!"*
EMO	Emoticon Handling	*":)"*	*"happy"*
NEG	Negation Handling	*"I am not happy today!"*	*"I am sad today!"*
WSG	Word Segmentation	*"#sometrendingtopic"*	*"some+trending+topic"*

number of training samples). These steps proved to improve performance and/or reduce time of execution. A significant finding reported in the study is that a proper data preprocessing is more crucial than the classification algorithm itself to obtain the best performance at the lowest possible cost (trade-off among effectiveness-efficiency).

3. Preprocessing techniques

In this section are presented the preprocessing techniques found in the literature using the following methodology. In one of the last comparative surveys [13], the authors present an evaluation of several text preprocessing techniques on two datasets built to perform sentiment analysis classification on Twitter. The article was used as the foundation for our work because – as shown in Table 7 – it proved *a posteriori* to contain the largest number of techniques presented. In order to obtain the list of related works on preprocessing techniques, all the works cited or citing the aforementioned work that discussed at least three different preprocessing techniques were included in our study. Techniques not discussed in [13] were added as columns to Table 7 and also included and discussed in our survey. Studies with less than three techniques are not shown in Table 7 but, if targeting some specific technique with a novel or deeper point of view, they have been briefly discussed in Section 3. At this point, for each of the study added from time to time to our reference list, the papers cited or citing each work in the Table 7 were included, as long as they discussed at least three different preprocessing techniques. Thanks to this approach, we can state that, to the best of our knowledge, the preprocessing techniques most frequently found in the literature have been included in this survey.

All the preprocessing techniques presented here represent the very first stage for any task related to TC. The next step, after text preprocessing, consists in splitting text into n-grams. Before providing the preprocessed text to a model, it is necessary to tokenize it. In some studies, tokenization is presented as a preprocessing technique. However, more than altering the syntactic and semantic content of a text, tokenization is the necessary procedure to fragment a dataset and to be able to supply it to subsequent stages. From this point of view, we do not include the tokenization among the techniques presented here. Nevertheless, we introduce it in the rest of this section.

Tokenization is discussed in [24–27] and splits texts into characters, words, group of words or other pieces called *tokens*. Although common forms of tokenization are performed at word-level, several sub-word tokenization strategies are discussed in the literature [28–30]. Regardless the size of the tokenization window used, tokenization consists in segmenting text. Usually, only alphanumeric or alphabetic characters separated by non-alphanumeric characters are taken into account when segmenting data (e.g. white spaces, tab, punctuation). The purpose of tokenization is to output single units of information – i.e. the tokens – to be mapped into numerical representations. The token list serves as the starting point for additional processing, such as text mining, parsing, or classification. Either linguistics (in which tokenization is a method to segment text) or computer science (where it is a component of lexical analysis) can benefit from tokenization. Depending on the language syntax, the tokenization process can be challenging. For example, the majority of words in languages like Italian and English are delimited and separated from one another by white spaces. Otherwise, languages like Chinese are not segmented since the borders between the words are not obvious. Also, the writing system and typographic form of words both affect tokenization.

One final note is about the order of application of several preprocessing techniques in combination. While some preprocessing approaches (such as removing stop words and punctuation) can be used independently of one another, others necessitate more careful thought about the order in which they are used to providing consistent results. For the tagger to function properly, POS tagging, for instance, should be applied before stemming, and negation should be done before eliminating stop words. Eventually, as reported in [31], it is worth notice that it is not necessary to perform preprocessing on both the training and test sets.

Finally, given the methodology reported in the previous section as well as in the rest of this article, the histogram in Fig. 1 displays a list of the preprocessing techniques that have been documented in the literature. The histogram also shows the number of times the reported techniques have been used in the related work.

3.1. Replace noise

The definition of noise varies significantly according to the literature, with regard to removing and/or replacing noise. The authors in [31] do not explicitly mention noise removal. However, they apply a few text preprocessing techniques at the beginning of their evaluation. These techniques involve removing HTML tags and special characters from text, such as "%*=()/". Furthermore, not all datasets are provided as plain text. Unwanted strings and Unicode, which are regarded as crawling byproducts, can add further noise to the data. For this reason, some authors employ regular expressions to eliminate Unicode strings and non-English words. Additionally, user-posted tweets may include URLs, user mentions or hashtags (such as "@username" or "#music"), or both. In this way, users can link their tweet to a certain subject or user, and these strings of characters can also convey emotion. In the literature are described a number of methods to deal with this additional data supplied by users. In [32], the authors replace all URLs with a tag *U*, and replace user mentions (e.g. "@brucespringsteen") with a tag *T*. The majority of academics believe that URLs do not

reveal anything about the sentiment of a tweet [33–36]. Other scholars expand short URLs into full URLs before tokenization [37,38]. The tweet text is then refined by removing any URLs that match the tokens. In short, no general rules apply in the definition and in managing the noise. Definition and operations performed vary from a study to another.

3.2. Replace slang and abbreviation

Considering the character count restrictions in social networks (e.g. Twitter), abbreviations, informal writing styles, short words and slangs are frequently used [39]. These words can also be managed (e.g. replacing "OMG" with "Oh My God"). By handling these informal words in the text and changing them to reflect their actual meaning, an automated classifier may perform better while preserving information. These words and sentences can be managed in order to impute their meaning accurately. In [40] slangs and abbreviations are converted into word meanings that can be comprehended by utilizing conventional text analysis methods. In [13] authors manually compile a lookup database with these words, phrases, and their replacements. However, it is worth noting that word embedding-based models could eventually manage slang and abbreviation as-is, understanding from the context, during the training phase, their original meaning.

3.3. Replace contraction

Contractions are short-form words that are used to reduce the number of characters in a tweet/post [41]. An apostrophe is used in contractions to replace one or more missing letters. One preprocessing method consists in performing contraction replacement (e.g., *can't* replaced by *cannot*).

Expanding contractions could or could not be a beneficial preprocessing technique before performing tokenization. In a word embedding layer which splits words at a space character, further meaning could be provided, keeping the word *can't* instead of *cannot*. This way, a single word can incorporate what is expressed by the two single consecutive words *can* and *not*. However, words like *not* could be of prominent importance for subsequent stages coming later, like the ones that replace negations with antonyms. Otherwise, if the splitting of the words is performed at punctuation, tokenization would create the tokens *can* and *'t*. In this last example, as it matches other negative forms in the text, this tokenization could deteriorate performance.

3.4. Remove repeated punctuation

In [13], the authors distinguish three punctuation signs: stop marks, question and exclamation. These punctuation marks, according to authors, indicate the presence of emotion in the text considered. Because of this, the authors substitute a representative tag in its place. For instance, "multiQuestionMark" is used in place of the token "???". This procedure is performed before deleting punctuation. However, in the not pre-trained models evaluated in this work, if there is not any space between repeated punctuation marks, a separated word is created in the dictionary. As an example, given the sentence: "*Are you sure???*", three different words will be considered as separated tokens (i.e. *Are, you* and *sure???*). In the case of a single and/or multiple spaces (i.e. "*Are you sure ???*"), four words/tokens will be added to the dictionary (i.e. *Are, you, sure* and *???*). These different splitting strategies would lead to different behaviors of a subsequent classifier.

3.5. Remove punctuation

In written texts, punctuation can be used to express sentiment and emotion [42] (e.g. "*We are late! Hurry up!!!*"). Even if this punctuation use can be easily understood by humans, it could not be so for an automatic classification tool. Furthermore, punctuation can be useless when dealing with certain TC task. For this reason, punctuation removal is often applied in many preprocessing tasks for automated TC. However, punctuation symbols can also denote sentiment. In [43], the authors detect punctuation signs like "*!!!*" and replace them with the label "*multiexclamation*". An application where punctuation is removed can be found in [44]. A model that includes a word embedding layer, in the case of the token *up!!!* as in our previous example, could be able to catch the meaning related to invoke someone to move faster. Removing punctuation from the sentence and replacing it with a single space (i.e. "*We are late Hurry up*"), would result in the change of some latent information maybe of interest for certain TC tasks (e.g. author profiling) [45].

3.6. Remove numbers

Despite the fact that numbers can offer helpful data to obtain a performance gain of a classifier, it is usual to delete them during the preprocessing stage [44,46]. Such a practice could be due to historical reasons, where computational power and traditional machine learning classifiers required a stricter preprocessing phase to lighten datasets. However, other scholars [4,45] argue that numbers are useful, indeed they do not remove them from the original source text.

In fact, the sentence: "*I won 2 dollars on bets.*" compared to: "*I won 2,000,000 dollars on bets.*" will become: "*I won dollars on bets.*". However, the resulting sentence has lost the intended meaning of the user who has written it. This meaning could be considered differently by an attention based model or even by a shallow neural network to provide the correct prediction. Even in the case of author profiling tasks, the use of numbers could characterize a user based on the quantity expressed by the numbers in text. Removing numbers could lead to another type of information loss. For instance, the removal of *4* from the sentence: "*I did it 4 you*" (i.e., "*I did it you*") would alter the original true meaning of the sentence even for a human classifier. Finally, removing the number *8* from the word *w8*, again, could lead to a loss of information and to a deterioration in performance as well as in the previous example.

3.7. Lowercasing

Among others, lowercasing (i.e. converting uppercase letters to lowercase) is one of the most common techniques to perform preprocessing on a source text before further steps.

Lowercasing is discussed in [47] and consists in converting to lowercase each character of a text (e.g. "*Your band sounds like Rolling Stones*" - "*your band sounds like rolling stones*"). Before the classification step, the authors in [48] change capital letters from uppercase to lowercase. According to authors, the classification's performance has improved. Lowercasing has been a common method employed in many deep and non-deep architectures presented in the literature due to its simplicity. Lowercasing may have undesirable effects on system performance since it increases ambiguity despite the fact that it reduces vocabulary size and sparsity [49]. In our example reported above – regarding the rock band *The Rolling Stones* – lowercasing could produce for a non-human classifier an ambiguity, comparing the sound of a band to a set of stones rolling[2] instead of comparing the same sound to the one of the popular rock band.

Lowercasing, on the other hand, conflates multiple spellings of words that are based on case. The diversity of capitalization in the dataset may interfere with classification and degrade performance. This could be the case of a single misspelled word in a dataset (e.g. "*houSe*"). In this case, a word embedding layer trained from scratch could assign

[2] ...and in this case you should look for a new drummer.

a new embedding vector instead of using the most properly semantic-related word "*house*".

Differences in experimental results across various works in the literature can be simply explained based on the domains considered. In our work, several datasets and models are tested, so we are confident on the general impact of this technique using modern classifiers and different datasets.

3.8. Remove stop words

The removal of stop words, according to our study, is the most employed preprocessing method found in the literature. Stop words are typically frequent terms in a language and are assumed to be the least informative [50] (i.e., stop words alone do not provide meaning to document). Stop words are language-specific and cannot be considered as keywords in text mining applications, so they could be useless in information retrieval. Stop words often appear in writings without being related to a specific subject (e.g. prepositions, articles, conjunctions, pronouns etc.). Before performing the TC task, stop words are typically removed. The size of a dataset is actually decreased after removing stop words from it. Example of stop words are: "of", "a", "the", "in", "an", "with", "and", "to". Depending on the list used, usually there are between 400 and 500 stop words in the English language.

The first study considering stop words is conducted in [51]. There, the author makes the suggestion that words in written texts can be split into terms considered as keyword or non-keyword using a stop list. In [52], the authors employ data from six different Twitter datasets to use different stop word detection algorithms and examine how eliminating stop words impacts the effectiveness of two popular supervised sentiment classification techniques. By tracking changes in the classification performance, in the amount of data sparsity and in the size of the feature space of the classifier, the authors evaluate the effects of eliminating stop words. The authors compare results between static stop word removal techniques (e.g. based on pre-compiled lists) versus dynamic stop word removal techniques [53] (e.g. based on dynamic detection of stop words in a document). The results demonstrate that the performance is adversely affected by the usage of pre-compiled stop words list. Otherwise, the best strategy to retain significant performance while lowering data sparsity and significantly condensing the space of the features appears to be the dynamic creation of stop word lists by deleting the uncommon words appearing rarely in the dataset. The researchers have found that a word's relevance can be inferred from its frequency in a data collection. This finding led to the exploration of various well-liked stop word removal techniques in the literature. While some approaches consider both the top and bottom-ranked words to be stop words, others make the assumption that stop words correspond to the most frequently occurring words. Another well-liked alternative to using the raw frequency of terms has also been discussed in the literature: Inverse Document Frequency (IDF).

Eventually, four different stop word removal techniques are reported here.

- *The traditional approach*. The traditional approach [54] relies on removing stop words gleaned from pre-compiled lists.
- *Approaches based on Zipf's law*. Three approaches for creating stop words that are moved by Zipf's law exist, besides the conventional stop words list [53,55]. Among these are the words that are most frequently used and words that only appear once, or singletons. Additionally, terms having a low inverse document frequency are thought to be removed (IDF).
- *The mutual information method*. A notion of how informative a term can be about a certain class is supplied by a supervised technique that determines the amount of information that each word and document class share [56]. A lower mutual information score indicates that the word has a weak ability in improving discrimination performance, hence it needs to be dropped.
- *Random sampling of data chunks*. It was initially suggested in [57] to use this technique to manually identify stop words in web publications. This approach operates by repeatedly processing different, randomly chosen data chunks. It then uses the Kullback–Leibler divergence [58] metric to order the terms in each chunk according to how informative they are.

3.9. Spelling correction

It is common that texts shared online by users contain spelling errors. For instance, tweets frequently contain typos as well as grammatical errors. The unintended consequence of having the same term transcribed differently is lessened by correcting spelling and grammar errors. Examples of misspelled words are: *absense*, *decieve*, *noticable*. After a spelling correction step, the mentioned words would be substituted respectively by: *absence*, *deceive*, *noticeable*. In [59] it is proven that correcting spelling errors can improve classification effectiveness. Although other type of errors could be introduced after performing a spelling correction, this technique generally improves performance.

Eventually, an interesting way to perform spell-checking is presented in [60] where a spell checker is employed to improve stemming, and synonyms of related tokens are combined.

3.10. Part-of-speech tagging

The word class is identified via Part-of-Speech (POS) tagging, which takes into account the word's placement in the sentence [61]. A POS tag is then given to any word in a sentence. Noun (NN), proper plural noun (NNPS), verb (VB), adverb (RB), superlative adverb (RBS), third-person verb (VBZ), and other tags are examples of tags.[3] It has been demonstrated that four POS classes – namely, nouns, adjectives, verbs, and adverbs – are more informative than other classes. Several purposes of POS tagging in preprocessing are discussed in related work. In [13] the use of POS tagging allows some parts of speech to be excluded since they do not express the suitable sentiment for the purpose at hand. Only verbs, adverbs, and nouns were kept in the study. In [62], in order to tag opinion statements with sentiments, the authors employ POS tags as pointers.

Specially in deep learning-based models, the process of assigning POS tags to each term is helpful to increase semantic informativeness in text. However, due to its impact on diminishing accuracy, some authors choose to omit POS tagging for certain tasks [63], while others found POS tagging beneficial [46].

3.11. Lemmatization

Lemmatization is used to replace a word with its corresponding lemma, or dictionary form. By analyzing a word's location in a sentence and removing its inflectional ending, this technique creates the lemma as it appears in a dictionary (e.g. *Performance is greatly improved*, replaced by *Performance be greatly improve*). In [64], lemmatization reduces various word forms to the same lemma to enhance user sentiment extraction effectiveness. Lemmatization is discussed in [47] and, in the context of an SVM model, in [65]. In [66] authors address the issue of ambiguation after lemmatization. The authors use lemmatization in combination with POS disambiguation to alleviate the problem.

Lemmatization has long been a common preprocessing step for NDL models. Since DL models have started to be employed, lemmatization has rarely been used as a preprocessing stage. Lemmatization's major goal is to reduce sparsity because a dataset may contain various inflected versions of the same lemma. Furthermore, in the context of author profiling tasks, lemmatization can lead to ignore relevant writing style details [67]. Eventually, it is worth reporting that in inflexionless language (e.g. Chinese), words are only in one form. For inflexionless languages, techniques like lemmatization or stemming, does not provide any change to the text.

[3] An example from Twitter is the case of a retweet replaced by the tag *RT*.

3.12. Stemming

To obtain stem versions of derived words, a process known as stemming is used. For instance, stemming techniques can reduce word variations like *easy*, *easily*, *easier*, *easiest* to the word *easy*. The dimensionality of dictionaries is decreased, since many words are collapsed to the same one. This procedure reduces entropy and raises the significance of the concept behind a word like the one from the previous example (i.e. *easy*). In the end, stemming enables the same consideration of nouns, verbs, and adverbs that share the same stem. Word frequencies are commonly calculated after stemming, since derived words share semantic similarities with their root forms.

The first known stemming algorithm was presented in 1968 and discussed in [68]. Going forward, the algorithm for stemming introduced in [69] is often employed by a multitude of scholars. It is likely the most popular and effective stemming technique for the English language.

Stemming is applied in [70] and also discussed in [71]. The goal of stemming in both studies is to find, for any derived word, its corresponding stem. As discussed in [72], the stemming algorithm depends on the language considered (i.e., Turkish in this case). The library commonly used for Turkish language is discussed in [73]. Considering that Turkish is an agglutinative language, stemming helps in reducing data sparsity. Otherwise, for the same language, the fixed-prefix approach described in [74] is a computationally straightforward yet highly efficient stemming tool. The performance and efficacy of stemming in applications like spelling checkers across languages are examined by the authors in [75]. Although advanced algorithms employ morphological understanding in creating a stem from the words, a typical simple stemming technique would involve deleting suffixes using a list of frequently occurring suffixes. The study provides a comprehensive overview of known stemmers for Indian languages, as well as popular stemming strategies.

Truncating approaches, statistical methods, and mixed methods are typically used to categorize stemmed algorithms. The mechanism used by each of these divisions to determine the word variations' stems is different. Below is a discussion of a few of these techniques. For further discussion on stemming techniques, a deep overview is presented in [76].

- *Truncating techniques* involve removing a word's prefixes or suffixes, referred to as affixes. Truncating a word at the nth character, is the simplest basic stemmer (i.e. it consists in keeping n letters and removing the remaining). Words that are shorter than n are left untouched using this strategy. When the word length is short, there is a greater chance of over stemming.
- *Porters stemmer* is one of the most well-known stemming algorithms and developed in 1980 [69]. On the fundamental algorithm, numerous alterations, improvements, and suggestions have been proposed. It is predicated considering that in the English language the suffixes are usually composed of groupings of simple and small suffixes. The algorithm is performed along five steps. Each step applies the rules until one of them satisfies the criteria. If a match is found, the suffix is then removed and the subsequent action is evaluated. At the end of the last stage, the resultant stem is returned.
- *Lovins stemmer* was proposed in 1968 [68], when Lovins made the first practical stemmer suggestion with his stemmer. The Lovins stemmer eliminates a word's longest suffix. Each word is altered, checking a different table that performs numerous alterations to turn the stems into acceptable words after the ending has been deleted. Due to the fact that it is a one pass method, it can never remove more than one suffix from a word. This algorithm has the following benefits: (1) it is extremely quick; (2) it can handle changing letters doubled for words as *getting* into *get* and (3) it can handle plurals that are irregular (e.g. "mouse" and "mouses", "die" and "dice" etc.). It is worth reporting that the Lovins stemmer, although being a heavier stemmer, results in superior data reduction. With its extensive suffix collection, the Lovins method only requires two significant stages to delete a suffix. The algorithm by Lovins is quicker than the Porter one, based on five iterations. Due to its extremely long endings list, it is larger than the Porter method.
- *Paice/Husk Stemmer* is introduced in [77] and is an ongoing method using one database that has more than one hundred rules and use the final character of a suffix as index. It tries to determine the relevant rule based on the final character of a word. Rules detail the substitution or deletion of a word ending. If any rule does not match, the algorithm ends. The algorithm ends also if the first character of a word is a vowel and no more than two or three letters remain in the word. If not, the rule is followed and the procedure is repeated. The benefit is that both deletion and replacement as per the rule are applied at every iteration. However, because of the weight of this stemmer, over stemming can happen.

The two primary categories of stemming cons are over- and under-stemming. If two words having different stems are replaced by the same root, then a case of over-stemming occurs. Another term for this is a false positive. On the other hand, the act of giving two words that ought to share the same root a different root is called under-stemming. This is also known as a false negative.

3.13. Removing elongation

A character that is repeated once or more times can be found in elongated words (e.g. *cooooool*, *greeeeeeat*, *goooood* etc.). Tweets and other social media posts frequently contain words with repeated letters [78]. Character repetitions are employed by users to emphasize and express their sentiments. The preprocess step of removing elongation consists in replacing elongated words with their source words, so they can be considered as the same entity. Repeated characters are reduced to a single one to prevent the learner from considering lengthened words differently from their basic form. If not, a classifier could interpret them as distinct words, and the longer words are likely to be underestimated because of their lower frequency in the text.

3.14. Emoticon handling

On the internet and in social networks, emotional icons are frequently used to denote users' sentiment [79]. Users use emoticons (e.g. :), :(etc.), to express opinion too. Not to be confused with emoticon, emoji are pictographs of objects, faces, and symbols. However, in a generic preprocessing step, the same operations used for emoticons can be applied to emoji too. Depending on the considered task, it could also be important to capture information provided by emoticons or emoji to perform TC.

In [80], the authors study and evaluate the impact of emoticons on sentiments of tweets. The authors demonstrate the value of emotional icons in conveying messages on social media. In [81], the usefulness of processing emoticons on user-generated content is highlighted by authors.

Emoticons could also be replaced with scores that express a score against a polarity, but they can also be translated into text in the corresponding word. For example, for a specific sentiment classification task, the words *pos* and *neg* can be used in place of the positive and negative icons, respectively. In other studies, emoticons are substituted with the words that best describe them, such as *:-(* with the word *sad*. However, for instance, the irony in the usage of a sad emoticon while texting something positive, can revert the original meaning of a sentence.

In [32], the authors employ emoticons as features and associate words to a value of pleasantness from one to three. Emoticons are

scored similarly to other words and are broken down into the following classes: extremely negative, negative, neutral, positive and extremely positive.

Keeping as-is emoticons in any text, for word-embedding based models, lead to the generation of a word vector with an associated semantic as for any other word in the dataset.

3.15. Negation handling

As stated in [31], one of the best preprocessing methods for tackling tasks involving sentiment analysis is negation handling. A crucial stage in sentiment analysis is dealing with negations, such as "*not nice*". One of the most relevant causes of misclassification is the omission of negation words, which can affect the tone of all the surrounding words. One way to perform negation handling is removing negative forms in text to reduce ambiguities of the classified sentences. Specifically, when facing with sentiment analysis tasks, negation is significant because, in many circumstances, the polarity of words or sentences can be affected by negation words, which can cause the polarity to invert. The most typical method of handling negation is to look for terms that are similar to "*not*" in each sentence, then see if the next word has an antonym. The word "*sad*" will be used in place of phrases like "*not happy*" for instance. To perform the replacement of words with the corresponding antonyms, it is generally used *WordNet*, presented in [82].

In [31] authors handle negation performing the following steps. At first, they compile an antonym dictionary using the WordNet dataset. In their work, the authors explain how to manage the three possible cases when looking for antonyms (i.e. a single antonym, multiple antonyms or no antonyms). The word's antonym is then randomly selected from the antonym dictionary. Eventually, the negation terms in tokenized text are identified by the authors. In the event that is discovered a negation word, the token that follows it (i.e., the word to be negated) is selected, and the antonym of that word is searched in the dictionary of the antonyms. The negated word and the negation word are swapped out if an antonym is found. In their work, the authors provide a running example where the sentence *"I am not happy today"* is replaced by the sentence *"I am sad today"*.

Handling negations can generally improve performance for sentiment analysis-related tasks based on sentence classification. However, a comprehensive study on the effect of handling negations for author profiling tasks (i.e. classifying a whole dataset related to an author instead of performing classification of single sentences) is still missing.

3.16. Word segmentation

It is quite common to find different words merged together in online texts. Such a case can be due both to a typing error or to a deliberate choice. In the first case, a user could wrongly type the word "*Beyoncelemonade*" instead of the two different words "*Beyoncé Lemonade*". The merged word represents noise and could likely be the only token in the dataset. In a tweet like: "*I like beyoncelemonade*" a model could not understand the topic (i.e. *music*) of the sentence. Considering the same merged word, a user could deliberately write *#beyoncelemonade* as hashtag within the shared post. In this case, segmentation would change the desired usage of the author. Nevertheless, segmenting merged words has proved to be helpful in understanding and better classifying contents of tweets and posts [83,84].

In other cases, a model could benefit from processing words grouped together. It is the case of words like "*United States*", where splitting single words as different tokens could make it harder for a model to catch the underlying concept of the single word "*UnitedStates*". In the second case, word embedding-based architectures could get the meaning of a whole sentence, understanding the reference to the specific country (i.e. United States of America).

4. Material and methods

To assess the impact of the three most common techniques (i.e., lowercasing, removing stop words and stemming) we performed several experiments. We evaluated the impact of every single technique but also the impact of all the possible combinations of them. In Fig. 2 is shown the process we applied for our experiments. As can be seen from the figure, the application order of each technique is relevant. For this reason, we evaluated all the preprocessing techniques in order. Even if in Fig. 2 we only show a running example using one, two or three techniques, in the result section we present and discuss the effect of using all the possible combination of two and three techniques. The libraries we used to apply the techniques were already presented and referenced in Section 3.

4.1. Evaluated models

We introduce the models evaluated in our experiments in this section. For the NDL models we provide the reference to the libraries used and in Section 4.3 we further discuss parameters setup for the tested configuration. For the DL models, we provide the reference to the original study reporting changes if applied. Transformers models are shortly discussed, and the specific pre-trained version used is reported. The steps applied before feeding each model are:

1. Preprocessing each sample's text
2. Word-by-word breakdown (at space characters) of the text in each preprocessed sample
3. Mapping each word (ngram) to a token
4. Associating a unique integer value (index of the token) to each token
5. Using these indices to translate each text into a sequence of integers

Then, two different operations can be performed following the step 5) with respect to NDL and deep models. For the NDL models the vector of ints is translated into a bag-of-words representation,[4] while for deep models the vector of ints is used as-is by the following word embedding layer. In the case of the DL models, the word embeddings are trained from scratch during the training phase. For the Transformers, the pre-trained embedding of each model is used. The fine-tuning is performed accordingly to each reference paper. The models evaluated are briefly discussed in the rest of this section.

- **Logistic Regression (LR)**. LR is commonly employed in TC for several tasks [85]. Despite its name, LR is actually a linear classification model. Maximum-entropy classification, logit regression and log-linear classifier are common terms to refer to LR. The LR is based on a logistic function that is employed to approximate the likelihoods of the possible results of an experiment. LR is also used for ensemble of text classifiers, as reported in [86]. For our experiments, we used the *sklearn* Logistic Regression implementation.[5] We used an L2 penalty, a C value equal to 1.0 and the *lbfgs* solver as discussed in [87].
- **Naïve Bayes (NB)**. As reported in [88] and experimentally demonstrated over time by outcomes from various TC tasks [89], NB is one of the most effective model to employ for classification. We evaluated a multinomial NB classifier from the *sklearn* MultinomialNB implementation.[6] Data are commonly expressed as word vector counts.

[4] An array containing at the *n*th index, corresponding to the *n*th int value, a counter of the occurrences of the corresponding n-gram.

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[6] https://scikit-learn.org/stable/modules/generated/sklearn.Na\"\ive_bayes.MultinomialNB.html

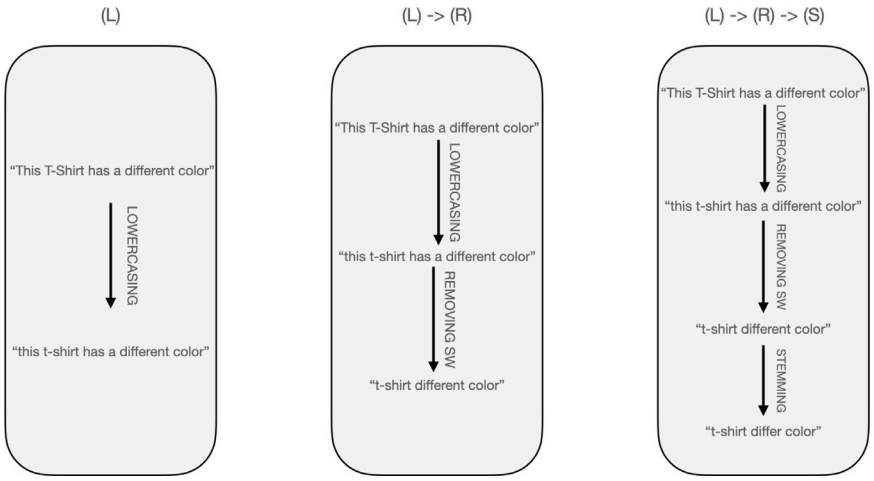


Fig. 2. From the left to the right, it is shown the preprocessing applied using a single technique and a combination of two and three techniques, respectively. As can be seen from the figure, the application order of each technique is relevant. In our experiments, we evaluated all the combination of the three most common techniques.

- **SVM**. As reported in [90] and in [91], classifiers based on SVM are well-established methods for TC tasks. SVM are also employed in ensemble-based text classifier, as reported in [92]. Thanks to SVM models, classification results compared to other classification methods have been greatly improved.
  Based on [93], we tested the *sklearn* SVC implementation.[7] As a regularization parameter, we used a value of 1.0 with a linear kernel type.
- **Artificial Neural Network (ANN)**. Starting from the research investigation on how coupled brain cells in the human brain could generate complex patterns, or neurons [94] along with the development of the perceptron [95], nowadays, ANN are widely implemented on a wide range of tasks. TC is no exception. The architecture we implemented for our experiments consists of an embedding layer, a dropout layer, three dense/dropout pairs of layers, a global average pooling layer, a dropout layer and a final single dense unit layer. The network architecture is shown in Fig. 3.
- **Convolutional Neural Network (CNN)**. The CNN evaluated here is the one presented in [45] and also used in [96–98]. In this case we do not report further details or the image of the network architecture which can be found in the above-mentioned papers. This CNN includes a single convolutional layer. As demonstrated by its results, this CNN outperforms Transformers and others proposed models as stated in [99] on a classification task similar to the ones proposed in this work.
- **Bidirectional LSTM (BiLSTM)**. In place of feed-forward networks, recurrent neural networks are widely utilized to categorize text data. In [100], the authors discuss how to perform TC using LSTM network and their variants like Bi-LSTM and GRU. For our work, we developed a simplified version of the bidirectional LSTM discussed in [101]. Also in this case, we do not report further details or the image of the network architecture which is presented in [101]. The model consists of two bidirectional LSTM layers. We did not employ any activation functions for any of the dense layers. We used a binary cross-entropy loss and the optimization algorithm by Adam [102] to train our model.
- **RoBERTa**. The authors in [103] – by offering a replication study on the pre-training of BERT – improve the performance of the BERT model by changing the pre-training stage. These adjustments consist of the following: (1) training the model for more time using larger batch size; (2) ignoring the objective of predicting next sentence; (3) using longer sequences for training; (4) altering the pattern for masking used on the training instances in a dynamic way. The version of RoBERTa we used is presented in [103].
- **ELECTRA**. According to what stated in [104], ELECTRA suggests replacing certain tokens with possible replacements taken from a small generator network, instead of masking the input like in BERT. Then, a discriminative model is trained to predict whether each token in the corrupted input was replaced by a generator sample or not, as opposed to developing a model that predicts the original identities of the corrupted tokens. Along with graph neural network, ELECTRA can also be employed as an embedding layer as in [105]. In our experiments, the original version of ELECTRA, presented in [104], was used.
- **XLNet**. A generalized autoregressive pretraining strategy is the one suggested in [106]. By optimizing the predicted likelihood across all combination of the factorization order, it enables learning bidirectional contexts. XLNet surpasses BERT, frequently by a significant margin, on a number of tasks, including question answering, sentiment analysis, document ranking and natural language inference. For our study we used the pre-trained XLNet using zero-shot cross lingual transfer discussed in [107].

A recent rise in the application of classification techniques based on graphs is noteworthy. A recent study can be found in [105] for TC but, recently, graph-based methods are also used for traffic prediction [108], computer vision [109] and social networking [110]. However, most of these methods are not yet able to outperform models evaluated in our study and discussed here.

4.2. The datasets

We present here the four datasets evaluated in our study and coming from different domains. We describe their structure, content and their respective size. All the four datasets used are publicly available and used in recent literature for TC tasks. We used datasets with varying sizes and distinct classification objectives to examine how each preprocessing strategy affects different classification tasks. The details of each dataset are shortly discussed and presented in the rest of this section. In Table 2 are reported the statistics of the datasets.

[7] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

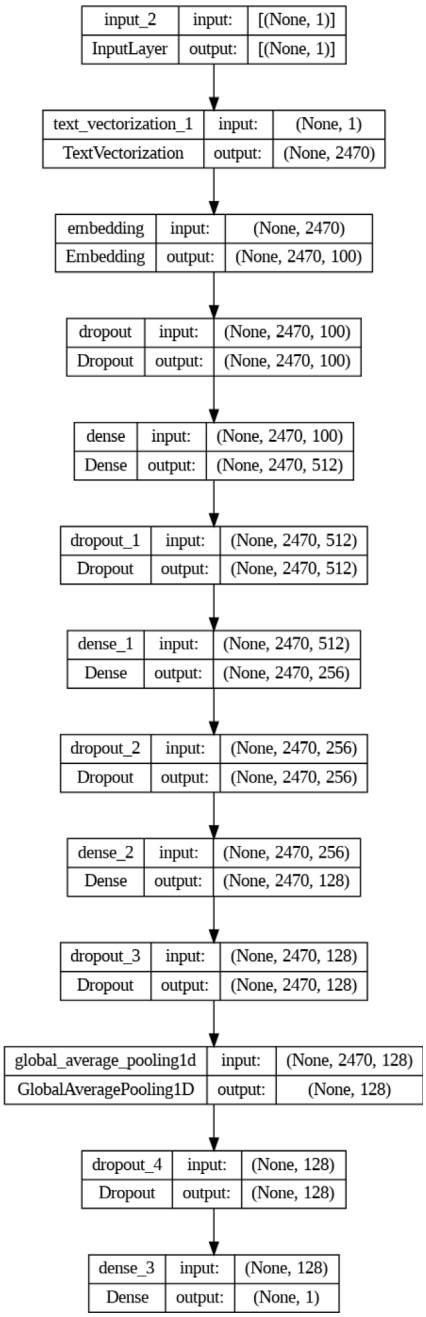


Fig. 3. The ANN architecture implemented for our experiments. Numbers in brackets indicate the tensor sizes. Layers as depicted on our Google Colab Notebook.

4.2.1. Fake news spreaders

The Fake News Spreaders (FNS) dataset is presented and discussed in [111] and available under request.[8] The dataset was used for the international shared task at PAN.[9] The task's organizers want to find out if it is possible to distinguish between authors who have previously disseminated fake news and those who have not. The task is in a series of shared task that from 2013 to 2019 considered a number of issues of author profiling on social media. Some of them included gender and language variation, age and gender coupled with personality, bot detection, and gender from a multimodality standpoint.

The dataset consists of tweets in Spanish and English. However, to test the original version of the Transformers evaluated in our study, we considered only the English ones. In the dataset, there are one hundred tweets representing each author and the corresponding class label for the author (i.e., 1 if the author has shared one or more fake news in the past and 0 otherwise). There are one hundred fifty authors per class in the training set, and one hundred authors per class in the test set. Resuming, the dataset samples (i.e., 500 authors) provide a total number of tweets (i.e., 50,000) suitable for the experiments with the models evaluated here.

4.2.2. Patronizing and condescending language

Described in detail in [112], the dataset for Patronizing and Condescending Language (PCL) is from the detection PCL task hosted at SemEval-2022. This task is an emerging one about detecting PCL [113]. PCL occurs when language implies superiority toward others, talks down to them, or kindly depicts them or their circumstances, elicits feelings of sorrows and compassion. PCL is often involuntary and unconscious and based on good intentions. The assignment is a classification problem where a classifier has to predict whether PCL is present in a given text. Below are reported two representative samples for the two label classes that a text classifier has to predict.

- **Non-PCL Sample Text:** *"In 2017, more than 150 people packed 1,100 hampers for Foodbank to provide to those in need during the holiday season".*
- **PCL Sample Text:** *"Housing Minister Grant Shapps added : The plight of homeless people should be on our minds all year round - not just at Christmas".*

The Task 4 participants at SemEval-2022 were given a dataset comprising sentences in context (paragraphs) that had been taken from news articles. Despite coming from various countries, the news pieces were all delivered in English. The News on Web (NoW)[10] originally supplied all the excerpts from news articles from media in 20 English-speaking nations. The 10,469 paragraphs make up the dataset, which served as the SemEval task's training set. Using the same procedure, organizers annotated 3,898 more texts to build the test set. To gather paragraphs, organizers employed a keyword-based technique, concentrating on texts that make mention of vulnerable communities (e.g. refugees or homeless). The dataset is available on GitHub.[11]

4.2.3. Internet Movie Database

A dataset for binary sentiment classification was first described in [114] as the Internet Movie Database (IMDB). It comprises 25,000 reviews of highly divisive movies for testing and 25,000 for training. Additional unlabeled data is also available for use. The collection includes binary sentiment polarity labels for the movie reviews that go along with them. It is meant to act as a baseline for sentiment analysis.

The total of 50,000 reviews are divided in 25,000 reviews each for training and testing, and make up the core dataset. The reviews are balanced for the two classes (i.e. 25,000 are positive and 25,000 are negatives). For unsupervised learning, a further 50,000 documents without labels are also included.

Often for the same movie, reviews have correlated ratings, so there can never be more than 30 reviews for any film in the entire collection. Additionally, because the train and test sets contain different movies, learning movie-specific terms and having them correspond to observed labels does not significantly improve performance. A poor review has a score of less than four out of ten in the designated train and test sets, while a score of more than seven out of ten is considered as a positive review. The train/test sets therefore do not contain reviews with more neutral ratings. Reviews of any rating are included in the unsupervised

[8] https://zenodo.org/record/4039435
[9] https://pan.webis.de
[10] https://www.english-corpora.org/now/
[11] https://github.com/Perez-AlmendrosC/dontpatronizeme

Table 2
Characterization and statistics of the datasets.

Dataset	Task	#Total documents	#Documents per sample	#Train samples	#Test samples
PCL	Content classification	14,367	1	10,469	3898
FNS	Author profiling	50,000	100	300	200
IMDB	Polarity detection	100,000	1	25,000	25,000
20N	Newsgroup post categorization	18,846	1	11,314	7532

set, and there are an even number of reviews with ratings between 5 and less than 5. Below are two examples of reviews, one each from a negative and positive class.

- **Negative review:** *"I and a friend rented this movie. We both found the movie soundtrack and production techniques to be lagging. The movie's plot appeared to drag on throughout with little surprise in the ending. We both agreed that the movie could have been compressed into roughly an hour giving it more suspense and moving plot".*
- **Positive review:** *"Maybe it's because I saw the movie before reading the book, but I really love this movie. I've seen it many many times and will be watching it many times more. It's a compelling story, that's interesting from the beginning to the end. It has everything: action, romance etc".*

The IMDB dataset is available online.[12]

4.2.4. The 20 Newsgroup data

To the best of our knowledge, the 20 Newsgroup data (20N) is discussed and used for the first time in [115] by Ken Lang. A total of around 18,000 messages from 20 different newsgroups make up this dataset. By newsgroup group, 1,000 messages were selected at random from each of the twenty newsgroups.[13] For our experiments, we used the splits between training and test set reported here.[14] This is the only non-binary dataset used for our experiments. One out of twenty categories needs to be predicted by the classifiers. Considering that this is a multi-classification problem, a model has to predict to which category a sample belongs. Below we report a part of a sample belonging to the category *electronics*.

- *"Pink noise has constant power per geometric frequency increment (octave, 1/3 octave, etc.). Thus the 10 kHz–20 kHz octave has the same amount of noise power as the 10 Hz–20 Hz octave. White noise has constant power per arithmetic frequency increment (Hz, kHz, etc.). Thus the 10 kHz–10.1 kHz band has the same amount of noise power as the 10 Hz–110 Hz band (both bands are 100 Hz wide). Pink noise can be made by passing white noise thru a -3db/octave filter (usually approximated by a network of several RC pairs). Note: you can't get -3db/octave by using half a -6db/octave network :- Pink noise is commonly used in audio power response measurements. It shows up on audio spectrum analyzers (with octave-related bands) as a flat line across the bands".*

4.3. Experimental setup

Our experiments were performed using TensorFlow on an NVIDIA GeForce RTX 2080 GPU on our local machine and on a Tesla T4 from Google Cloud. For the Transformers we used the *Simple Transformers*[15] library. All the Transformers used came from the library of Transformers provided in [116]. The batch size for all models was 1. For 10 epochs, we fine-tuned the Transformers-based models, early stopping in accordance with the test set accuracy. The best accuracies of the Transformers models used, as suggested in reference work, were generally obtained before the tenth epoch of fine-tuning. The DL architectures (ANN, CNN and BiLSTM) were trained for 20 epochs. In this case too, there were no benefits in training over 20 epochs; on the test set, the best accuracies were always obtained before epoch 20. We followed the protocol used in [103] to evaluate the performance of each deep model tested. So, we initialize each model with random weights and then we run the training and the evaluation phases for five times (with early stopping for each run), then we report the median accuracy along the five runs, as the representative result of each model. Furthermore, in Section 5 we report the maximum gap from such a median considering the five runs. The Jupyter notebooks hosted on GitHub can be used to study the outcomes of our experiments. For the three NDL models, we use the implementations discussed in Section 4.1 and because of their deterministic nature there is no need of performing multiple runs. We have previously provided references to each model's original implementation, along with each architecture's experimental setup.

5. Results

We report our comments on the results in this section. The results reported in this section show the effect of the three most common preprocessing techniques. Our experiments investigated not only the effect of single techniques but also in combination. On the three dataset, from Table 3 to Table 6 the results of our experiments are reported. In each table is shown the binary accuracy measured as the number of correct predictions divided on the number of all the predictions provided. In evaluating the preprocessing impact, the *DON* strategy represents the case where no preprocessing is applied. It means that each sample in the datasets is provided *as-is* to the learning model. Likewise, the results associated to *LOW* show the impact of lowercasing each character in the dataset samples. Also, the impact of the combination of the three techniques is evaluated. In the second block of each table, we show the results obtained using two techniques in combination. For example, the case *(L)-(R)* shows the performance when each sample in the dataset is lowercased and, after that, each stop word in the sample is removed. For the third block of rows in the tables, we report the results obtained using the combination of the three techniques.

Furthermore, even if already stated, it is worth repeating that for deep models the median over five runs with random initialization is reported. Next to the median is reported the gap between the median and the lowest/highest accuracy obtained along the five runs. The best result (i.e., the best median on the five runs) is reported in bold black, while the worst result is shown in bold red. Eventually, for the deep architectures tables, the acronyms of the preprocessing techniques are abbreviated for readability purposes. In Fig. 4 and in Fig. 5 we show the box-and-whisker plots for the three evaluated datasets and for each model tested. The distributions used to build up each plot are taken from the Tables from 3 to 6 and each box represents the result distribution for the model indicated on the *x-axis*.

5.1. IMDB

For the IMDB dataset, the results of the deep models are shown in the Table 3. The best performance is obtained by ELECTRA with lowercasing as preprocessing technique. The use of the same model

[12] https://ai.stanford.edu/~amaas/data/sentiment/
[13] https://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/news20.html
[14] http://qwone.com/~jason/20Newsgroups/
[15] https://simpleTransformers.ai/about/

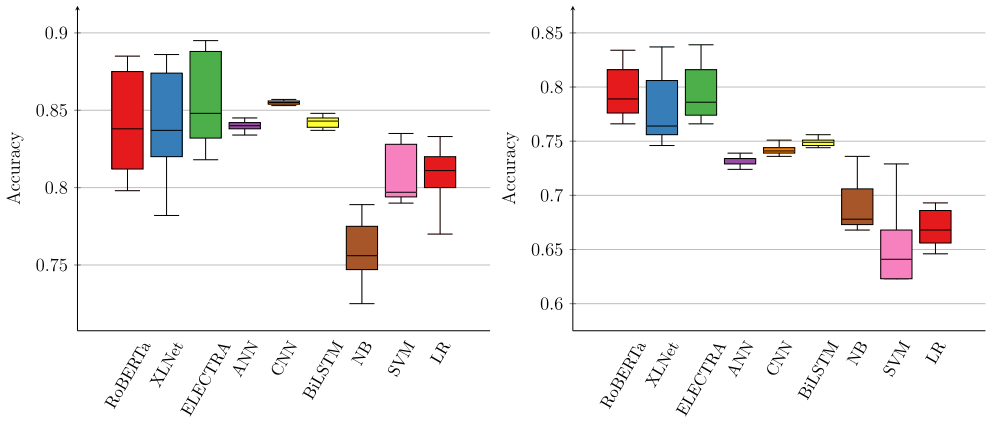


Fig. 4. Box plot for the nine models evaluated on the IMDB (left) and on the PCL (right) datasets. The DL models are the less sensitive to the preprocessing strategy employed, while the Transformers are the most sensitive.

Table 3
Median accuracy and maximum gap from the median accuracy of the six deep models on the IMDB dataset. In bold black and bold red are shown the best and the worst results, respectively, for each model.

	IMDB					
Preprocessing	RoBERTa	XLNet	ELECTRA	ANN	CNN	BiLSTM
DON (D)	0.884 ± 0.00	0.885 ± 0.00	0.888 ± 0.00	0.835 ± 0.01	0.856 ± 0.00	0.847 ± 0.00
LOW (L)	0.877 ± 0.00	0.881 ± 0.01	**0.895 ± 0.04**	0.842 ± 0.01	**0.857 ± 0.00**	0.843 ± 0.01
RSW (R)	**0.885 ± 0.00**	**0.886 ± 0.00**	0.890 ± 0.07	0.840 ± 0.01	0.855 ± 0.00	0.843 ± 0.01
STM (S)	0.853 ± 0.00	0.852 ± 0.03	0.857 ± 0.05	**0.834 ± 0.01**	0.856 ± 0.00	**0.837 ± 0.02**
(L)→(R)	0.875 ± 0.04	0.878 ± 0.01	0.888 ± 0.01	0.840 ± 0.01	0.854 ± 0.00	0.844 ± 0.01
(L)→(S)	0.849 ± 0.00	0.847 ± 0.01	0.860 ± 0.03	**0.845 ± 0.00**	0.855 ± 0.00	0.845 ± 0.02
(R)→(L)	0.876 ± 0.04	0.874 ± 0.00	0.890 ± 0.01	0.844 ± 0.01	0.855 ± 0.00	0.847 ± 0.01
(R)→(S)	0.826 ± 0.02	0.823 ± 0.32	0.832 ± 0.02	0.839 ± 0.00	0.855 ± 0.00	0.844 ± 0.02
(S)→(L)	0.849 ± 0.00	0.845 ± 0.03	0.864 ± 0.01	0.839 ± 0.00	0.854 ± 0.00	0.840 ± 0.01
(S)→(R)	**0.798 ± 0.07**	0.817 ± 0.01	0.832 ± 0.01	0.843 ± 0.01	0.854 ± 0.00	0.843 ± 0.01
(L)→(S)→(R)	0.806 ± 0.04	0.782 ± 0.12	0.824 ± 0.01	0.837 ± 0.01	0.855 ± 0.00	0.839 ± 0.34
(L)→(R)→(S)	0.838 ± 0.34	0.820 ± 0.02	0.837 ± 0.04	0.842 ± 0.01	0.854 ± 0.00	0.845 ± 0.00
(S)→(L)→(R)	0.812 ± 0.01	**0.645 ± 0.18**	**0.818 ± 0.02**	0.840 ± 0.01	0.856 ± 0.00	0.845 ± 0.01
(S)→(R)→(L)	0.818 ± 0.02	0.820 ± 0.05	0.837 ± 0.01	0.843 ± 0.01	**0.853 ± 0.00**	0.839 ± 0.01
(R)→(L)→(S)	0.829 ± 0.03	0.837 ± 0.17	0.825 ± 0.05	0.838 ± 0.01	0.855 ± 0.00	**0.848 ± 0.01**
(R)→(S)→(L)	0.806 ± 0.03	0.822 ± 0.07	0.848 ± 0.01	0.838 ± 0.01	**0.857 ± 0.00**	0.838 ± 0.34

employing stemming, lowercasing and removing stop words as combination leads to a gap of over the 7% in the classification performance. Also for XLNet the same preprocessing combination consistently degrades performance. In this case, the gap between the best and the worst result in the table is above the 25%. The worst result for RoBERTa involves stemming and removing stop words. This result seems to highlight that pre-trained model do not benefit by reducing words to their corresponding stem. Nevertheless, while the Transformers' performance improves using word variations, stop words appear to be not necessary at all. In fact, the best results for RoBERTa and XLNet are obtained removing stop words. Even the second-best result of ELECTRA is obtained removing stop words with a very low gap from the best result. So, removing stop words has to be taken into account when dealing with Transformers on datasets similar to the IMDB one evaluated here. Using DL models there is not a substantial difference between the worst and the best results while varying the combination of techniques applied. This finding can also be noted in Fig. 4. In fact, the size of the boxes related to the result distributions of the DL models are significantly smaller. The CNN performs consistently better than ANN and BiLSTM. Furthermore, lowercasing is always involved in any best result obtained by the DL models. Finally, it is worth mentioning that while the deviation from the median along the five runs changes for any model and any preprocessing technique, the CNN obtains an impressive and consistent null variation for any preprocessing technique considered along the five runs. CNN is also the only model with a variation under the 2% considering the best and the worst combination of preprocessing techniques in terms of accuracy. In fact, the worst result is 0.853 using combination of stemming, stop words removal and lowercasing while the best one is 0.857 using lowercasing or combination of removing stop words, stemming and lowercasing. It can be stated that, even if stemming is one of the most studied and employed preprocessing technique discussed in the literature, it does not appear to be involved in any of the best combination of techniques considered here.

The results of the NDL models on the same dataset are reported in Table 6. The best results are obtained by the SVM without using any preprocessing technique or removing stop words and lowercasing combination. Considering that for the SVM the gap between the best and the worst result is below 5% it should be evaluated if the preprocessing stage is worth the additional complexity. Similar gaps between the best and the worst results happen for the other two models (i.e. NB and LR). Finally, as already stated, the worst results for each model involve stemming.

5.2. PCL

The results of the deep models for the PCL dataset are reported in Table 4. Only for this dataset it happens that the best performance is obtained using a single technique or no preprocessing at all. The best performance is obtained by ELECTRA with lowercasing as preprocessing technique. This result is aligned with the one obtained in the case of the IMDB dataset. With a very similar behavior, in this case too, the use of stemming, lowercasing and removing stop word as combination leads to the worst result. However, the gap with the best result, in this case, is more than the 9%. The worst result is obtained

Table 4
Median accuracy and maximum gap from the median accuracy of the six deep models on the PCL dataset. In bold black and bold red are shown the best and the worst results, respectively, for each model.

	PCL					
Preprocessing	RoBERTa	XLNet	ELECTRA	ANN	CNN	BiLSTM
DON (D)	**0.834 ± 0.01**	**0.837 ± 0.01**	0.832 ± 0.02	0.734 ± 0.01	0.746 ± 0.01	0.746 ± 0.02
LOW (L)	0.816 ± 0.01	0.829 ± 0.01	**0.839 ± 0.01**	0.731 ± 0.00	0.741 ± 0.01	0.749 ± 0.01
RSW (R)	0.827 ± 0.01	0.811 ± 0.03	0.816 ± 0.01	**0.739 ± 0.01**	0.741 ± 0.01	**0.756 ± 0.03**
STM (S)	0.804 ± 0.03	0.796 ± 0.30	0.799 ± 0.00	0.734 ± 0.00	**0.751 ± 0.01**	0.749 ± 0.02
(L)→(R)	0.824 ± 0.01	0.806 ± 0.31	0.822 ± 0.02	0.731 ± 0.01	0.741 ± 0.01	0.751 ± 0.01
(L)→(S)	0.811 ± 0.02	0.796 ± 0.02	0.794 ± 0.01	0.736 ± 0.01	0.739 ± 0.00	0.749 ± 0.02
(R)→(L)	0.822 ± 0.01	0.809 ± 0.31	0.827 ± 0.02	0.729 ± 0.00	0.739 ± 0.01	**0.744 ± 0.01**
(R)→(S)	0.779 ± 0.04	0.754 ± 0.03	0.774 ± 0.01	0.734 ± 0.01	0.744 ± 0.01	0.751 ± 0.02
(S)→(L)	0.809 ± 0.01	0.804 ± 0.01	0.806 ± 0.02	0.729 ± 0.01	0.741 ± 0.01	0.746 ± 0.01
(S)→(R)	0.786 ± 0.02	0.756 ± 0.26	0.776 ± 0.02	0.736 ± 0.01	0.741 ± 0.01	0.749 ± 0.01
(L)→(S)→(R)	0.776 ± 0.05	0.759 ± 0.02	0.766 ± 0.06	**0.721 ± 0.02**	0.739 ± 0.02	0.749 ± 0.01
(L)→(R)→(S)	0.774 ± 0.01	0.754 ± 0.02	0.774 ± 0.04	0.731 ± 0.01	0.749 ± 0.01	0.751 ± 0.01
(S)→(L)→(R)	**0.766 ± 0.01**	**0.746 ± 0.13**	**0.766 ± 0.01**	0.724 ± 0.01	0.744 ± 0.01	0.751 ± 0.00
(S)→(R)→(L)	0.789 ± 0.01	0.759 ± 0.01	0.786 ± 0.06	0.734 ± 0.01	**0.736 ± 0.00**	0.746 ± 0.00
(R)→(L)→(S)	0.771 ± 0.03	0.756 ± 0.06	0.781 ± 0.01	0.736 ± 0.01	0.741 ± 0.01	**0.744 ± 0.01**
(R)→(S)→(L)	0.786 ± 0.02	0.764 ± 0.01	0.771 ± 0.02	0.734 ± 0.01	0.746 ± 0.00	**0.744 ± 0.01**

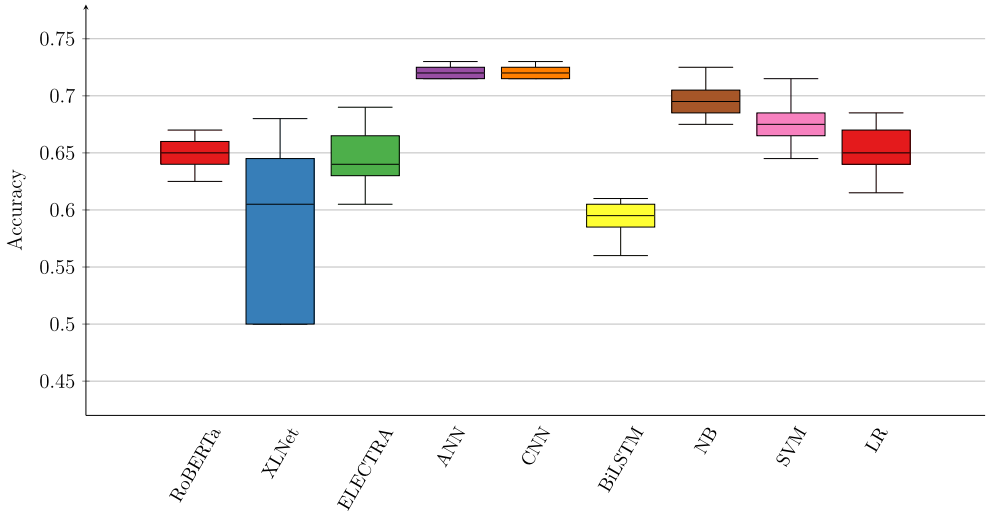


Fig. 5. Box plot for the nine models evaluated on the FNS dataset. On this dataset, eight out of nine models show minimal sensitiveness to the preprocessing strategies.

by the ANN using lowercasing, stemming and removing stop words as a combination of techniques (i.e., 0.721). Considering the ANN, there is not any substantial improvement selecting the best combination (i.e. removing stop words, 0.739). Finally, for the first time, one of the best results involves stemming as preprocessing technique (this is the case of the CNN with stemming as combination). However, even for the PCL dataset, DL models do not exhibit a significant difference between the worst and the best results while varying the combination of techniques applied. The CNN performs consistently better than ANN and BiLSTM. Consistently with the results reported in the literature, stop word removal is involved in the best results obtained by the ANN and the BiLSTM. It is interesting that no combination of multiple techniques are involved in the best results obtained on this dataset. Finally, even for this dataset, the deviation from the median along the five runs changes smoother for the shallow models with respect to the Transformers-based ones.

The results obtained by the three NDL models are reported in Table 6. The best result (i.e., 0.736) is reached by the NB employing lowercasing as preprocessing technique. For this dataset, performance is more responsive to the combination employed. As instance, for the SVM the gap between the best and the worst result (i.e., last four rows in the table) is above the 10%. This should lead to further attention when selecting a proper preprocessing technique for an SVM if dealing with similar tasks. Even for this dataset, the worst results for each model involve stemming.

5.3. FNS

The results of the deep models are reported in Table 5 for the FNS dataset. The best performance of 0.730 is obtained by a simple CNN applying only stop word removal as preprocessing technique. The same result is obtained by the ANN using removing stop words, stemming and lowercasing as a combination. However, results along the five runs are more consistent in the case of the ANN. The worst results (i.e., 0.500) are obtained by the XLNET with several combinations of techniques. However, also in this case XLNET is very sensitive to the combination of techniques employed. This is proved by the gap between the best and the worst results (i.e. 18%). This can also be noted from the size of the box plot in Fig. 5. As shown in the table, stop word removal is involved in four best results over six. In the remaining two best results, stemming and lowercasing are involved.

It is worth repeating that this dataset is very different in the numbers and shape of the samples with respect to other datasets. In fact, any sample consists of the last 100 tweets of a Twitter user. As widely discussed in [117], NDL and DL perform consistently better than Transformers. Also on this dataset, stop word removal could be generally considered as a proper preprocessing method when dealing with deep models. Even for this dataset, DL models do not exhibit a great difference between the worst and the best results, while varying the

Table 5
Median accuracy and maximum gap from the median accuracy of the six deep models on the FNS dataset. In bold black and bold red are shown the best and the worst results, respectively, for each model.

	FNS					
Preprocessing	RoBERTa	XLNet	ELECTRA	ANN	CNN	BiLSTM
DON (D)	0.695 ± 0.02	0.620 ± 0.12	0.605 ± 0.09	0.720 ± 0.00	0.725 ± 0.02	0.585 ± 0.11
LOW (L)	0.655 ± 0.04	0.645 ± 0.04	**0.690 ± 0.02**	0.730 ± 0.01	0.720 ± 0.01	0.610 ± 0.08
RSW (R)	**0.705 ± 0.01**	**0.680 ± 0.18**	**0.560 ± 0.02**	0.725 ± 0.01	**0.730 ± 0.01**	0.595 ± 0.07
STM (S)	0.660 ± 0.03	0.500 ± 0.13	0.665 ± 0.01	0.715 ± 0.02	0.720 ± 0.03	0.610 ± 0.05
(L)→(R)	0.665 ± 0.02	0.645 ± 0.14	0.680 ± 0.14	0.720 ± 0.02	0.715 ± 0.01	0.565 ± 0.02
(L)→(S)	**0.625 ± 0.04**	0.510 ± 0.15	0.670 ± 0.05	0.720 ± 0.01	0.715 ± 0.01	0.595 ± 0.07
(R)→(L)	0.670 ± 0.02	0.650 ± 0.05	0.665 ± 0.03	0.725 ± 0.01	0.720 ± 0.01	**0.560 ± 0.05**
(R)→(S)	0.650 ± 0.15	**0.500 ± 0.00**	0.645 ± 0.00	0.715 ± 0.01	0.720 ± 0.01	0.595 ± 0.07
(S)→(L)	0.660 ± 0.13	0.500 ± 0.17	0.665 ± 0.02	0.725 ± 0.00	0.725 ± 0.01	**0.645 ± 0.04**
(S)→(R)	0.660 ± 0.15	0.515 ± 0.13	0.630 ± 0.03	**0.715 ± 0.00**	0.725 ± 0.01	0.605 ± 0.07
(L)→(S)→(R)	0.640 ± 0.10	0.575 ± 0.07	0.630 ± 0.12	0.715 ± 0.01	0.715 ± 0.01	0.585 ± 0.08
(L)→(R)→(S)	0.645 ± 0.01	0.625 ± 0.12	0.635 ± 0.07	0.715 ± 0.01	0.720 ± 0.01	0.600 ± 0.06
(S)→(L)→(R)	0.640 ± 0.14	0.645 ± 0.14	0.640 ± 0.14	0.725 ± 0.01	**0.715 ± 0.00**	0.585 ± 0.06
(S)→(R)→(L)	0.640 ± 0.14	0.500 ± 0.15	0.610 ± 0.11	0.720 ± 0.00	0.720 ± 0.01	0.610 ± 0.08
(R)→(L)→(S)	0.645 ± 0.12	0.660 ± 0.16	0.635 ± 0.05	0.720 ± 0.02	0.720 ± 0.02	0.570 ± 0.11
(R)→(S)→(L)	0.640 ± 0.01	0.605 ± 0.10	0.655 ± 0.15	**0.730 ± 0.00**	0.725 ± 0.01	0.590 ± 0.06

Table 6
Accuracies for the three non-deep models on the four test dataset used. In bold black and bold red are shown the best and the worst results, respectively, for each model. For NB on 20N, we avoid black bold for most of the column because of the same results.

	IMDB			PCL			FNS			20N		
Preprocessing	NB	SVM	LR	NB	SVM	LR	NB	SVM	LR	NB	SVM	LR
DON	0.767	**0.835**	0.798	0.726	**0.729**	**0.693**	0.685	**0.630**	0.640	0.040	**0.160**	**0.140**
LOW	0.771	0.831	0.801	**0.736**	0.696	0.668	0.695	0.665	0.650	0.040	0.140	0.100
RSW	0.787	0.831	**0.833**	0.719	0.651	0.686	0.705	**0.715**	0.660	**0.020**	**0.100**	0.060
STM	0.741	0.794	0.773	0.683	0.678	0.691	**0.675**	0.645	0.640	0.040	**0.160**	0.080
LOW → RSW	0.787	0.828	**0.833**	0.706	0.671	0.683	0.720	0.690	0.680	0.040	0.140	0.040
LOW → STM	**0.725**	0.803	**0.770**	0.678	0.668	0.688	0.700	0.665	**0.615**	0.040	0.120	0.100
RSW → LOW	**0.789**	**0.835**	0.820	0.721	0.663	0.691	**0.725**	0.690	0.675	0.040	0.120	**0.020**
RSW → STM	0.780	0.794	0.811	**0.671**	0.641	0.656	0.680	0.695	0.675	**0.020**	**0.160**	0.100
STM → LOW	**0.725**	0.803	0.800	0.678	0.668	0.673	0.700	0.665	0.635	0.040	0.120	0.060
STM → RSW	0.775	**0.790**	0.821	0.681	0.641	**0.646**	0.675	0.675	0.670	**0.020**	0.140	0.120
LOW → STM → RSW	0.750	0.799	0.820	0.678	**0.623**	0.648	0.695	0.680	0.645	0.040	0.140	0.080
LOW → RSW → STM	0.747	0.794	0.821	0.668	0.636	0.661	0.700	0.685	0.650	0.040	0.140	0.080
STM → LOW → RSW	0.749	0.797	0.814	0.678	**0.623**	0.661	0.690	0.675	0.645	0.040	0.140	0.080
STM → RSW → LOW	0.749	0.797	0.814	0.678	**0.623**	0.661	0.690	0.685	0.655	0.040	0.140	0.080
RSW → LOW → STM	0.757	0.797	0.807	0.673	**0.623**	0.678	0.720	0.670	0.655	0.040	0.140	0.120
RSW → STM → LOW	0.756	0.797	0.803	0.673	**0.623**	0.651	0.720	0.675	**0.685**	0.040	**0.160**	0.080

combination of techniques applied. Considering the DL models, deviation from the median along the five runs is more consistent also for this dataset if compared with Transformers.

The results obtained by the three NDL models on the same dataset are reported in Table 6. The best result is obtained by the NB classifier using the removing stop words and lowercasing combination as a preprocessing technique. The gap between the best and the worst results for each model is still under the 5% also for this dataset. The worst results for the NB model involve stemming. However, as in the case of the LR and of the SVM, the worst performance is obtained performing no preprocessing at all.

5.4. 20N

The results obtained by the three NDL models on the 20N dataset are reported in Table 6. It is worth repeating that this dataset entails a multi-class classification, and the accuracies reported are related to the performance in assigning the correct category to a newsgroup article. The best result of 0.160 is obtained by the SVM using different preprocessing strategies. Even if stemming has rarely proved to be an effective preprocessing choice, in this case it allows the SVM to perform at its best. However, the results using stemming, removing stop words and stemming and removing stop words stemming and lowercasing are the same obtained with no preprocessing applied. There is no point in using any preprocessing with NB. In this case, the gap between the best and the worst result is irrelevant, and we do not even highlight the best results obtained almost in every preprocessing combination. The LR shows the most variable behavior in terms of results. In fact, the gap between the worst and the best case is of the 12% and the best result is obtained when no preprocessing is applied. Contrary to what happens for the IMDB dataset, removing stop words and lowercasing is the worst preprocessing combination. From a general perspective, the preprocessing impact on the 20N datasets is similar to the one exhibited on the PCL dataset. In two out of three models used, there are no benefits in applying some preprocessing to the data.

Regarding the 20N dataset, we do not show the table about the DL and the Transformer models. If this table had been shown it would be, in most cases, a set of full red and black bold numbers. For the same reason, we do not show the box plot for all the models. In fact, for this dataset, we have found very small variations applying different preprocessing strategies. While the range 0.080–0.012 of the accuracies for every model is very similar to the one shown for the NDL models, employing the deep learning classifiers the results are often more consistent and around 0.100 regardless of the preprocessing strategy applied. However, it is worth noting that the CNN for the DL models and RoBERTa for the Transformers are the top performing models using removing stop words as a preprocessing strategy. As already stated, the detailed results of our experiments are available on GitHub.

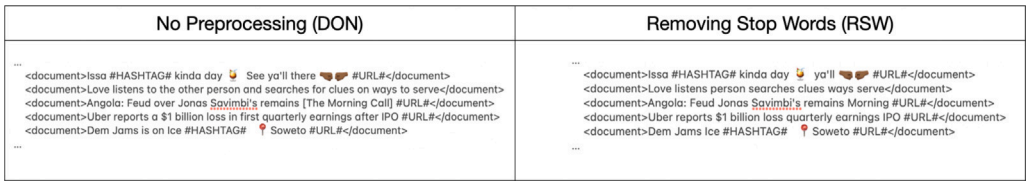


Fig. 6. Effect of no-preprocessing and of one of the preprocessing strategy on a small part of a single sample from the FNS dataset.

6. Discussion

From a theoretical point of view, we have empirically shown that the text preprocessing strategy can affect the performance of any modern classifiers, including the most recent Transformers-based architectures (RQ1). Along the different datasets used, it can be seen that preprocessing only marginally affects DL models, while the most significant impact is on the Transformers (RQ2). It is likely that this result depends on the word embedding for the two classes of models. While the Transformers take advantage of a pretraining phase, the embedding trained from scratch in the case of the DL models could be the main cause of the less sensitivity to the preprocessing strategy applied. In Fig. 4 similar results between the IMDB and the PCL dataset can be observed. Interestingly, the NDL models are also sensitive to the preprocessing strategy applied, but not as much as the Transformers. It is worth mentioning that while for the IMDB and the PCL datasets the impact of the preprocessing strategy can significantly affect the outcomes, in the case of the FNS dataset, the only model really sensitive to the preprocessing strategy is the XLNet. In the other cases the result distributions prove that the most common preprocessing strategies, alone or in combination, do not significantly change the outcomes. This fact could be due to the sample size in the FNS dataset. As already stated, each sample is made up by the last one hundred tweets of an author. So the impact of preprocessing could be less significant because of the more information available in each sample with respect to the samples in the IMDB and in the PCL dataset.

As a consequence of the high impact of the preprocessing, even simple classification methods can achieve state-of-the-art results, outperforming more complex and recent pre-trained architectures (i.e., the Transformer-based ones) (RQ3). We discovered that also for pre-trained architectures, the preprocessing step plays a significant role, and it is able to drastically revert the final outcome of a classifier. In other words, this confirms that different and simple preprocessing strategies constitute a critical aspect in the pipeline of any TC task. Eventually, the preprocessing stage can also affect the classification performance more than the classification model itself.

With regard to the box plots, it can be stated that appear irrelevant to focus on preprocessing when dealing with DL models without pre-trained word embedding like the ones evaluated here. Similar observations can be likely extended to datasets containing samples with long text instead of just few sentences, as in the case of the IMDB or the PCL datasets. Consequently, the preprocessing strategy to apply when dealing with Transformer-based models should be carefully evaluated, considering that the most used techniques not necessarily lead to improvements compared to not performing preprocessing at all. On the other hand, it is evident from the box plots in Fig. 4 that a wrong preprocessing strategy in place of the best one can significantly change the outcomes of the same model.

As proved by the results provided, the impact of preprocessing is increasingly important depending on the size of the dataset samples. In fact, looking at the box plots, the larger the samples of the dataset are (as in the case of FNS) the less the chosen preprocessing strategy matters. Furthermore, Transformers-based models are the less sensitive to the preprocessing combination employed, with respect to not performing any preprocessing. Finally, while lowercasing can be considered as the first choice when dealing with ELECTRA, removing stop words and do not performing preprocessing should be considered when using RoBERTa or XLNet. On the other hand, stemming should be carefully employed when in combination with other techniques. In fact, as discussed in the previous section, for any deep model used in this study it often degrades performance. The only interesting and surprisingly result is the case of the CNN on the PCL dataset. In this case, the use of stemming leads to the best result obtained by the CNN.

For the multi-class classification task regarding the 20N dataset, we have found a similar impact of preprocessing when looking at the PCL dataset. This could be motivated by a similar structure of the samples in the two datasets or, eventually, to similar contents. For this reason, given different preprocessing strategy applied, a certain model could respond similarly in terms of performance gap.

6.1. Qualitative analysis

In this section, we conduct a brief qualitative analysis to show how the text preprocessing alters dataset samples and how the specific strategy affects the performance.

As previously reported, the minor impact of preprocessing is visible on the FNS dataset regardless of the model used. An example without and with preprocessing (i.e., removing stop words) is shown in Fig. 6. Any snippet of text enclosed within tags *document* represents tweets from the same author. In this case, the classification task is about classifying an author as an FNS or as a non-FNS. To accomplish the task, 100 tweets written by the author are evaluated.

From the example shown, it is possible to understand that the impact of preprocessing is minimal. This result is confirmed by the results of the experiments. Especially considering that the one reported is only an extract of a sample relating to an author and not the entire sample including the other tweets. Therefore, what emerged from the results of the experiments on the FNS dataset can be motivated by the fact that having to classify an author using the set of tweets he wrote, regardless of the preprocessing applied, the stylistic information that allows to classify an author as FNS or non-FNS is however present. Therefore, the impact of the preprocessing strategy used is minimal and on a dataset of this type, preprocessing could be neglected.

On the other hand, looking at Fig. 7, it is easy to understand the reason preprocessing is so relevant on datasets similar to the IMDB one. The goal of classification in this case is to understand whether a single review is positive or negative. In this case, the average length of the single sample is shorter than that of the samples of the FNS dataset. Therefore, one preprocessing choice rather than another can drastically change the results obtained while maintaining the same model. For example, in the case of XLNet, the best classification result was obtained using the removal of stop words as the only preprocessing strategy. Instead, the worst result was obtained by the strategy that involves the removal of stop words followed by stemming. In this case, the accuracy gap is above 18%. This result is easy to be understood, looking at the three samples from the dataset shown in Fig. 7.

7. Conclusion and future works

In this study, we have presented the most popular preprocessing techniques found in the literature. We have then evaluated and compared the effect of the three most common preprocessing techniques

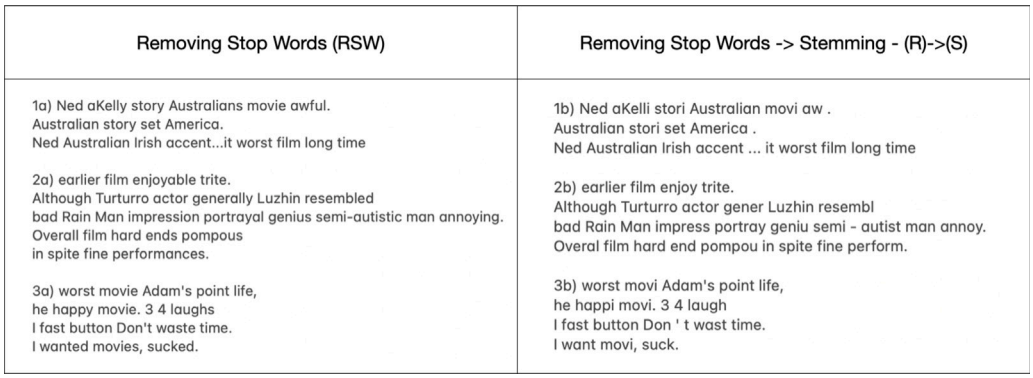


Fig. 7. Effect of removing stop words only and of stemming after removing stop words on the IMDB dataset. Three different samples from the dataset are shown.

Table 7
Techniques discussed in related work that proposes at least three different preprocessing methods.

Article	RNS	RSA	RCT	RRP	RPT	RNB	LOW	RSW	SCO	POS	LEM	STM	ECR	EMO	NEG	WSG
Alam (2019) [118]								X				X		X		
Albalawai (2021) [119]	X				X	X		X			X	X	X			
Alzahrani (2021) [120]	X				X	X		X								
Anandarajan (2019) [46]	X				X	X	X	X		X	X	X				
Angiani (2016) [6]	X						X	X	X			X			X	
Araslanov (2020) [121]	X					X	X	X			X	X				
Babanejad (2020) [31]	X				X	X	X	X	X	X		X			X	
Bao (2014) [17]	X										X	X	X		X	
Denny (2018) [4]					X	X	X	X				X				
Duong (2021) [122]		X			X	X	X	X	X	X			X	X	X	
Hacohen (2020) [123]					X		X	X	X				X			
Haddi (2013) [124]		X						X				X			X	
Hickman (2022) [3]		X	X				X	X	X		X	X		X	X	
Jianqiang (2017) [20]	X	X	X			X		X					X			
Kadhim (2018) [125]	X							X				X				
Kathuria (2021) [2]	X	X			X	X	X	X		X	X	X	X		X	
Koopman (2020) [126]					X		X	X			X	X				
Kowsari (2019) [127]	X	X					X	X			X	X				
Kumar (2019) [128]	X	X			X			X	X		X	X		X		
Kunilovskaya (2021) [129]					X		X	X			X		X			
Lison (2017) [130]					X			X		X	X					
Mohammad (2018) [131]	X		X				X	X			X	X				
Naseem (2021) [14]	X	X	X		X	X	X	X	X		X		X	X		X
Petrovic (2019) [132]					X		X	X				X				
Pradha (2019) [133]	X				X		X	X	X		X	X				
Rosid (2020) [134]							X	X				X				
Singh (2016) [12]	X	X			X			X	X							
Smelyakov (2020) [135]								X			X	X				
Symeonidis (2018) [13]	X	X	X	X	X	X	X	X	X	X	X	X	X		X	
Toman (2006) [136]						X	X	X			X	X				
Uysal (2014) [48]							X	X				X				
Zong (2021) [137]	X							X		X	X	X				

on four datasets from different domains. To determine the impact of various preprocessing combination on various datasets, extensive testing was done. Nine machine learning models were used to evaluate each preprocessing method. The article also lists the worst- and best-performing strategies in terms of the dataset and the model evaluated, and it suggests techniques that, whether employed alone or in combination, consistently outperform the others. Results vary also in relation to the different algorithm, which demonstrates that selecting a learning algorithm that is appropriate for the task at hand is crucial for enhancing the TC performance. The best preprocessing strategies, either separately or in combination, that produce the best classifier performance are suggested following tests with various strategies and observation of the interactions of the preprocessing method employed. Our analysis emphasizes how crucial preparing data is to ensure consistency when comparing various learning models. Moreover, our research highlights that, depending on the preprocessing method selected, the results are highly variable, also using modern Transformers. Our findings ought to motivate researchers to pick their preprocessing choices carefully and to document those choices when assessing or contrasting various models.

While one could conclude that removing stop words and lowercasing are two well-performing preprocessing technique, based on our study, it should be noticed that performing no preprocessing at all, is rarely the best choice for optimal results. The recent significant growth in model understanding capabilities (e.g., Transformers) has caused the emphasis to shift away from data and toward the evolution and development of newer and more powerful models. With this work, we aimed to draw attention to and explore the importance of the impact of the source data and associated preprocessing, which should not be disregarded. Specific preprocessing can aid in both increasing effectiveness and performance to better understand the behavior of the most recent Transformers-based NLP models, such as ChatGPT. Because of the very automated and promising performance of Transformers, the current trend is to underestimate the best preprocessing method of text (and this is proven by the increasing lack of attention on the subject). However, it is just on the Transformers that we have found the greatest gap between the best and the worst combination of preprocessing techniques used. This increased understanding could lead to the creation of newer models that are not only improved in performance but also developed more consciously.

Future research in this area can further look into the impact of these and other preprocessing approaches for NLP tasks others than TC. Also, other preprocessing technique combinations and how they interact could be further investigated. Future studies could eventually

investigate other classes of models and the impact of the preprocessing in relation to the samples size in the dataset evaluated. In fact, as also noted in [99,111,138], concerning three different author profiling tasks, the best performance obtained by the NDL and the DL models in place of Transformers should be investigated. For all of these author profiling datasets, the impact of preprocessing could be investigated to further corroborate the findings reported in our study. Finally, different preprocessing methods could also be used to further investigate the behaviors of DL and Transformer-based models. The benefit could be to unveil some interesting mechanisms happening under the hood, with particular regard to the field of the deep learning.

CRediT authorship contribution statement

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

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data that supports the findings of this study are available from the references specified in the Section 4.

Acknowledgments

The authors would like to thank Elisa Di Nuovo for assisting in the proofreading of this paper. The authors would also like to thank the Editor and the anonymous reviewers for their valuable advices.

References

[1] D.W. Otter, J.R. Medina, J.K. Kalita, A survey of the usages of deep learning for natural language processing, IEEE Trans. Neural Netw. Learn. Syst. 32 (2) (2021) 604–624, http://dx.doi.org/10.1109/TNNLS.2020.2979670.

[2] A. Kathuria, A. Gupta, R. Singla, A review of tools and techniques for preprocessing of textual data, Comput. Methods Data Eng. (2021) 407–422.

[3] L. Hickman, S. Thapa, L. Tay, M. Cao, P. Srinivasan, Text preprocessing for text mining in organizational research: Review and recommendations, Organ. Res. Methods 25 (1) (2022) 114–146.

[4] M.J. Denny, A. Spirling, Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it, Political Anal. 26 (2) (2018) 168–189.

[5] F.S. Al-Anzi, D. AbuZeina, Stemming impact on arabic text categorization performance: A survey, in: 2015 5th International Conference on Information & Communication Technology and Accessibility, ICTA, IEEE, 2015, pp. 1–7.

[6] G. Angiani, L. Ferrari, T. Fontanini, P. Fornacciari, E. Iotti, F. Magliani, S. Manicardi, A comparison between preprocessing techniques for sentiment analysis in Twitter, in: CEUR Workshop Proceedings. Vol. 1748, KDWeb, 2016, pp. 1–11.

[7] S. Agarwal, S. Godbole, D. Punjani, S. Roy, How much noise is too much: A study in automatic text classification, in: Seventh IEEE International Conference on Data Mining, ICDM 2007, IEEE, 2007, pp. 3–12.

[8] H. Uğuz, A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm, Knowl.-Based Syst. 24 (7) (2011) 1024–1032.

[9] J.T. Hancock, C. Landrigan, C. Silver, Expressing emotion in text-based communication, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2007, pp. 929–932.

[10] H. Jamshed, S.A. Khan, M. Khurram, S. Inayatullah, S. Athar, Data preprocessing: A preliminary step for web data mining, 3c Tecnol. Glosas Innov. Apl. Pyme 8 (1) (2019) 206–221.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).

[12] T. Singh, M. Kumari, Role of text pre-processing in twitter sentiment analysis, Procedia Comput. Sci. 89 (2016) 549–554.

[13] S. Symeonidis, D. Effrosynidis, A. Arampatzis, A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis, Expert Syst. Appl. 110 (2018) 298–310.

[14] U. Naseem, I. Razzak, P.W. Eklund, A survey of pre-processing techniques to improve short-text quality: a case study on hate speech detection on twitter, Multimedia Tools Appl. 80 (28) (2021) 35239–35266.

[15] A. Kurniasih, L.P. Manik, On the role of text preprocessing in BERT embedding-based DNNs for classifying informal texts, Int. J. Adv. Comput. Sci. Appl. 13 (6) (2022) 927–934, http://dx.doi.org/10.14569/IJACSA.2022.01306109.

[16] U.H. Hair Zaki, R. Ibrahim, S. Abd Halim, I.I. Kamsani, Text detergent: The systematic combination of text pre-processing techniques for social media sentiment analysis, in: International Conference of Reliable Information and Communication Technology, Springer, 2022, pp. 50–61.

[17] Y. Bao, C. Quan, L. Wang, F. Ren, The role of pre-processing in twitter sentiment analysis, in: International Conference on Intelligent Computing, Springer, 2014, pp. 615–624.

[18] N. Garg, K. Sharma, Text pre-processing of multilingual for sentiment analysis based on social network data., Int. J. Electr. Comput. Eng.(2088-8708) 12 (1) (2022).

[19] M. Arief, M.B.M. Deris, Text preprocessing impact for sentiment classification in product review, in: 2021 Sixth International Conference on Informatics and Computing, ICIC, IEEE, 2021, pp. 1–7.

[20] Z. Jianqiang, G. Xiaolin, Comparison research on text pre-processing methods on twitter sentiment analysis, IEEE Access 5 (2017) 2870–2879.

[21] W. Cunha, V. Mangaravite, C. Gomes, S. Canuto, E. Resende, C. Nascimento, F. Viegas, C. França, W.S. Martins, J.M. Almeida, T. Rosa, L. Rocha, M.A. Gonçalves, On the cost-effectiveness of neural and non-neural approaches and representations for text classification: A comprehensive comparative study, Inf. Process. Manage. 58 (3) (2021) 102481, http://dx.doi.org/10.1016/j.ipm.2020.102481, URL https://www.sciencedirect.com/science/article/pii/S0306457320309705.

[22] J.Á. González, L.-F. Hurtado, F. Pla, Transformer based contextualization of pre-trained word embeddings for irony detection in Twitter, Inf. Process. Manage. 57 (4) (2020) 102262, http://dx.doi.org/10.1016/j.ipm.2020.102262, URL https://www.sciencedirect.com/science/article/pii/S0306457320300200.

[23] W. Cunha, S. Canuto, F. Viegas, T. Salles, C. Gomes, V. Mangaravite, E. Resende, T. Rosa, M.A. Gonçalves, L. Rocha, Extended pre-processing pipeline for text classification: On the role of meta-feature representations, sparsification and selective sampling, Inf. Process. Manage. 57 (4) (2020) 102263, http://dx.doi.org/10.1016/j.ipm.2020.102263, URL https://www.sciencedirect.com/science/article/pii/S030645731931461X.

[24] M. Hassler, G. Fliedl, Text preparation through extended tokenization, WIT Trans. Inf. Commun. Technol. 37 (2006).

[25] P. McNamee, J. Mayfield, Character n-gram tokenization for European language text retrieval, Inf. Retr. 7 (1) (2004) 73–97.

[26] S. Vijayarani, R. Janani, Text mining: open source tokenization tools-an analysis, Adv. Comput. Intell. Int. J.(ACII) 3 (1) (2016) 37–47.

[27] L.A. Mullen, K. Benoit, O. Keyes, D. Selivanov, J. Arnold, Fast, consistent tokenization of natural language text, J. Open Source Softw. 3 (23) (2018) 655.

[28] R. Sennrich, B. Haddow, A. Birch, Neural machine translation of rare words with subword units, in: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2016, pp. 1715–1725.

[29] T. Kudo, Subword regularization: Improving neural network translation models with multiple subword candidates, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2018, pp. 66–75.

[30] M. Schuster, K. Nakajima, Japanese and korean voice search, in: 2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, IEEE, 2012, pp. 5149–5152.

[31] N. Babanejad, A. Agrawal, A. An, M. Papagelis, A comprehensive analysis of preprocessing for word representation learning in affective tasks, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, 2020, pp. 5799–5810, http://dx.doi.org/10.18653/v1/2020.acl-main.514, URL https://aclanthology.org/2020.acl-main.514.

[32] A. Agarwal, B. Xie, I. Vovsha, O. Rambow, R. Passonneau, Sentiment analysis of twitter data, in: Proceedings of the Workshop on Language in Social Media, LSM 2011, 2011, pp. 30–38.

[33] L. Ketsbaia, B. Issac, X. Chen, Detection of hate tweets using machine learning and deep learning, in: 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom, IEEE, 2020, pp. 751–758.

[34] S. Indra, L. Wikarsa, R. Turang, Using logistic regression method to classify tweets into the selected topics, in: 2016 International Conference on Advanced Computer Science and Information Systems, Icacsis, IEEE, 2016, pp. 385–390.

[35] A. Aljebreen, W. Meng, E. Dragut, Segmentation of tweets with urls and its applications to sentiment analysis, in: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, 2021, pp. 12480–12488.

[36] F. Resyanto, Y. Sibaroni, A. Romadhony, Choosing the most optimum text preprocessing method for sentiment analysis: Case: iphone tweets, in: 2019 Fourth International Conference on Informatics and Computing, ICIC, IEEE, 2019, pp. 1–5.

[37] E. Borra, B. Rieder, Programmed method: Developing a toolset for capturing and analyzing tweets, Aslib J. Inf. Manag. 66 (3) (2014) 262–278.

[38] S. Benzarti, R. Faiz, EgoTR: Personalized tweets recommendation approach, in: Intelligent Systems in Cybernetics and Automation Theory: Proceedings of the 4th Computer Science on-Line Conference 2015 (CSOC2015), Vol 2: Intelligent Systems in Cybernetics and Automation Theory, Springer, 2015, pp. 227–238.

[39] L. Tan, H. Zhang, C. Clarke, M. Smucker, Lexical comparison between wikipedia and twitter corpora by using word embeddings, in: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers), 2015, pp. 657–661.

[40] E. Kouloumpis, T. Wilson, J. Moore, Twitter sentiment analysis: The good the bad and the omg!, in: Proceedings of the International AAAI Conference on Web and Social Media, Vol. 5, 2011, pp. 538–541.

[41] D. Sagolla, 140 Characters: A Style Guide for the Short Form, John Wiley & Sons, 2009.

[42] M. Thelwall, The heart and soul of the web? Sentiment strength detection in the social web with SentiStrength, in: Cyberemotions, Springer, 2017, pp. 119–134.

[43] A. Balahur, Sentiment analysis in social media texts, in: Proceedings of the 4th Workshop on Computational Approaches To Subjectivity, Sentiment and Social Media Analysis, 2013, pp. 120–128.

[44] C. Lin, Y. He, Joint sentiment/topic model for sentiment analysis, in: Proceedings of the 18th ACM Conference on Information and Knowledge Management, 2009, pp. 375–384.

[45] M. Siino, E. Di Nuovo, T. Ilenia, M. La Cascia, Detection of hate speech spreaders using convolutional neural networks, in: PAN 2021 Profiling Hate Speech Spreaders on Twitter@ CLEF, vol. 2936, CEUR, 2021, pp. 2126–2136.

[46] M. Anandarajan, C. Hill, T. Nolan, Text preprocessing, in: Practical Text Analytics, Springer, 2019, pp. 45–59.

[47] J. Camacho-Collados, M.T. Pilehvar, On the role of text preprocessing in neural network architectures: An evaluation study on text categorization and sentiment analysis, in: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, 2018, pp. 40–46.

[48] A.K. Uysal, S. Gunal, The impact of preprocessing on text classification, Inf. Process. Manag. 50 (1) (2014) 104–112.

[49] N. Djuric, J. Zhou, R. Morris, M. Grbovic, V. Radosavljevic, N. Bhamidipati, Hate speech detection with comment embeddings, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 29–30.

[50] M. Gerlach, H. Shi, L.A.N. Amaral, A universal information theoretic approach to the identification of stopwords, Nat. Mach. Intell. 1 (12) (2019) 606–612.

[51] H.P. Luhn, Key word-in-context index for technical literature (kwic index), Am. Document. 11 (4) (1960) 288–295.

[52] H. Saif, M. Fernandez, Y. He, H. Alani, On stopwords, filtering and data sparsity for sentiment analysis of Twitter, in: Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC'14, 2014, pp. 810–817.

[53] M. Makrehchi, M.S. Kamel, Automatic extraction of domain-specific stopwords from labeled documents, in: Advances in Information Retrieval: 30th European Conference on IR Research, ECIR 2008, Glasgow, UK, March 30-April 3, 2008. Proceedings 30, Springer, 2008, pp. 222–233.

[54] C. Van Rijsbergen, Information Retrieval, second ed., Butterworth-Heinemann Newton, MA, USA, 1979.

[55] C. Courseault Trumbach, D. Payne, Identifying synonymous concepts in preparation for technology mining, J. Inf. Sci. 33 (6) (2007) 660–677.

[56] T.M. Cover, Elements of Information Theory, John Wiley & Sons, 1999.

[57] R.T.-W. Lo, B. He, I. Ounis, Automatically building a stopword list for an information retrieval system, in: Journal on Digital Information Management: Special Issue on the 5th Dutch-Belgian Information Retrieval Workshop (DIR), vol.5, 2005, pp. 17–24.

[58] J.M. Joyce, Kullback-leibler divergence, in: International Encyclopedia of Statistical Science, Springer, 2011, pp. 720–722.

[59] T. Mullen, R. Malouf, A preliminary investigation into sentiment analysis of informal political discourse, in: AAAI Spring Symposium: Computational Approaches To Analyzing Weblogs, 2006, pp. 159–162.

[60] D. Virmani, S. Taneja, A text preprocessing approach for efficacious information retrieval, in: Smart Innovations in Communication and Computational Sciences, Springer, 2019, pp. 13–22.

[61] C.D. Manning, H. Schütze, G. Weikurn, Foundations of statistical natural language processing, SIGMOD Rec. 31 (3) (2002) 37–38.

[62] L. Barbosa, J. Feng, Robust sentiment detection on Twitter from biased and noisy data, in: Proceedings of the 23rd International Conference on Computational Linguistics: Posters, COLING '10, Association for Computational Linguistics, USA, 2010, pp. 36–44.

[63] E. Boiy, P. Hens, K. Deschacht, M. Moens, Automatic sentiment analysis in online text, in: Openness in Digital Publishing: Awareness, Discovery and Access - Proceedings of the 11th International Conference on Electronic Publishing Held in Vienna - ELPUB 2007, Vienna, Austria, June 13-15, 2007. Proceedings, 2007, pp. 349–360, URL https://nbn-resolving.org/urn:nbn:se:elpub-138_elpub2007.

[64] E. Guzman, W. Maalej, How do users like this feature? A fine grained sentiment analysis of app reviews, in: 2014 IEEE 22nd International Requirements Engineering Conference, RE, 2014, pp. 153–162, http://dx.doi.org/10.1109/RE.2014.6912257.

[65] E. Leopold, J. Kindermann, Text categorization with support vector machines. How to represent texts in input space? Mach. Learn. 46 (1) (2002) 423–444.

[66] I. Kuznetsov, I. Gurevych, From text to lexicon: Bridging the gap between word embeddings and lexical resources, in: Proceedings of the 27th International Conference on Computational Linguistics, 2018, pp. 233–244.

[67] D.I. Hernández Farías, R.M. Ortega-Mendoza, M. Montes-y Gómez, Exploring the use of psycholinguistic information in author profiling, in: Mexican Conference on Pattern Recognition, Springer, 2019, pp. 411–421.

[68] J.B. Lovins, Development of a stemming algorithm., Mech. Transl. Comput. Linguist. 11 (1–2) (1968) 22–31.

[69] M.F. Porter, An algorithm for suffix stripping, Program Electron. Libr. Inf. Syst. 14 (3) (1980) 130–137.

[70] V. Srividhya, R. Anitha, Evaluating preprocessing techniques in text categorization, Int. J. Comput. Sci. Appl. 47 (11) (2010) 49–51.

[71] S. Vijayarani, M.J. Ilamathi, M. Nithya, Preprocessing techniques for text mining-an overview, Int. J. Comput. Sci. Commun. Netw. 5 (1) (2015) 7–16.

[72] F. Gemci, K.A. Peker, Extracting turkish tweet topics using LDA, in: 2013 8th International Conference on Electrical and Electronics Engineering, ELECO, IEEE, 2013, pp. 531–534.

[73] A.A. Akın, M.D. Akın, Zemberek, an open source NLP framework for turkic languages, Structure 10 (2007) 1–5.

[74] F. Can, S. Kocberber, E. Balcik, C. Kaynak, H.C. Ocalan, O.M. Vursavas, Information retrieval on turkish texts, J. Am. Soc. Inf. Sci. Technol. 59 (3) (2008) 407–421.

[75] V. Gupta, G.S. Lehal, Punjabi language stemmer for nouns and proper names, in: Proceedings of the 2nd Workshop on South Southeast Asian Natural Language Processing, WSSANLP, 2011, pp. 35–39.

[76] C. Moral, A. de Antonio, R. Imbert, J. Ramírez, A survey of stemming algorithms in information retrieval., Inf. Res. Int. Electron. J. 19 (1) (2014).

[77] C.D. Paice, Another stemmer, SIGIR Forum 24 (3) (1990) 56–61, http://dx.doi.org/10.1145/101306.101310.

[78] A. Bakliwal, P. Arora, S. Madhappan, N. Kapre, M. Singh, V. Varma, Mining sentiments from tweets, in: Proceedings of the 3rd Workshop in Computational Approaches To Subjectivity and Sentiment Analysis, 2012, pp. 11–18.

[79] A. Hogenboom, D. Bal, F. Frasincar, M. Bal, F. De Jong, U. Kaymak, Exploiting emoticons in sentiment analysis, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 703–710.

[80] H. Wang, J.A. Castanon, Sentiment expression via emoticons on social media, in: 2015 Ieee International Conference on Big Data, Big Data, IEEE, 2015, pp. 2404–2408.

[81] S. Pecar, M. Simko, M. Bielikova, Sentiment analysis of customer reviews: Impact of text pre-processing, in: 2018 World Symposium on Digital Intelligence for Systems and Machines, DISA, 2018, pp. 251–256, http://dx.doi.org/10.1109/DISA.2018.8490619.

[82] G.A. Miller, WordNet: a lexical database for english, Commun. ACM 38 (11) (1995) 39–41.

[83] D.D. Palmer, A trainable rule-based algorithm for word segmentation, in: 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics, 1997, pp. 321–328.

[84] H. Yamaguchi, K. Tanaka-Ishii, Text segmentation by language using minimum description length, in: Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 2012, pp. 969–978.

[85] K. Shah, H. Patel, D. Sanghvi, M. Shah, A comparative analysis of logistic regression, random forest and knn models for the text classification, Augment. Hum. Res. 5 (1) (2020) 1–16.

[86] M. Siino, I. Tinnirello, M. La Cascia, T100: A modern classic ensemble to profile irony and stereotype spreaders, in: CEUR Workshop Proceedings, Vol. 3180, CEUR, 2022, pp. 2666–2674.

[87] R.H. Byrd, P. Lu, J. Nocedal, C. Zhu, A limited memory algorithm for bound constrained optimization, SIAM J. Sci. Comput. 16 (5) (1995) 1190–1208.

[88] A. McCallum, K. Nigam, A comparison of event models for naive bayes text classification, in: AAAI-98 Workshop on Learning for Text Categorization, Vol. 752, Citeseer, 1998, pp. 41–48.

[89] S. Raschka, Naive bayes and text classification i-introduction and theory, 2014, arXiv preprint arXiv:1410.5329.

[90] F. Colas, P. Brazdil, Comparison of SVM and some older classification algorithms in text classification tasks, in: IFIP International Conference on Artificial Intelligence in Theory and Practice, Springer, 2006, pp. 169–178.

[91] Z. Liu, X. Lv, K. Liu, S. Shi, Study on SVM compared with the other text classification methods, in: 2010 Second International Workshop on Education Technology and Computer Science, Vol. 1, IEEE, 2010, pp. 219–222.

[92] D. Croce, D. Garlisi, M. Siino, An SVM ensamble approach to detect irony and stereotype spreaders on Twitter, in: CEUR Workshop Proceedings, Vol. 3180, CEUR, 2022, pp. 2426–2432.

[93] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 1–27.

[94] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, Bull. Math. Biophys. 5 (4) (1943) 115–133.

[95] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain., Psychol. Rev. 65 (6) (1958) 386.

[96] S. Mangione, M. Siino, G. Garbo, Improving irony and stereotype spreaders detection using data augmentation and convolutional neural network, in: CEUR Workshop Proceedings, Vol. 3180, CEUR, 2022, pp. 2585–2593.

[97] M. Siino, I. Tesconi, Profiling cryptocurrency influencers with few-shot learning using data augmentation and electra, in: CEUR Workshop Proceedings, Vol. 3497, CEUR, 2023, pp. 2772–2781.

[98] M. Siino, I. Tinnirello, Xlnet with data augmentation to profile cryptocurrency influencers, in: CEUR Workshop Proceedings, Vol. 3497, CEUR, 2023, pp. 2763–2771.

[99] F. Rangel, G.L. De la Peña Sarracén, B. Chulvi, E. Fersini, P. Rosso, Profiling hate speech spreaders on Twitter task at PAN 2021., in: CLEF (Working Notes), 2021, pp. 1772–1789.

[100] J. Nowak, A. Taspinar, R. Scherer, LSTM recurrent neural networks for short text and sentiment classification, in: International Conference on Artificial Intelligence and Soft Computing, Springer, 2017, pp. 553–562.

[101] M. Siino, M. La Cascia, I. Tinnirello, Mcrock at SemEval-2022 task 4: Patronizing and condescending language detection using multi-channel CNN, hybrid LSTM, distilBERT and XLNet, in: Proceedings of the 16th International Workshop on Semantic Evaluation, SemEval-2022, Association for Computational Linguistics, Seattle, United States, 2022, pp. 409–417, http://dx.doi.org/10.18653/v1/2022.semeval-1.55, URL https://aclanthology.org/2022.semeval-1.55.

[102] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[103] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, 2019, arXiv preprint arXiv:1907.11692.

[104] K. Clark, M.-T. Luong, Q.V. Le, C.D. Manning, Electra: Pre-training text encoders as discriminators rather than generators, 2020, arXiv preprint arXiv:2003.10555.

[105] F. Lomonaco, G. Donabauer, M. Siino, COURAGE at CheckThat! 2022: Harmful tweet detection using graph neural networks and ELECTRA, in: Working Notes of CLEF 2022—Conference and Labs of the Evaluation Forum, CLEF '2022, Bologna, Italy, 2022, pp. 573–583.

[106] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R.R. Salakhutdinov, Q.V. Le, Xlnet: Generalized autoregressive pretraining for language understanding, Adv. Neural Inf. Process. Syst. 32 (2019).

[107] G. Chen, S. Ma, Y. Chen, L. Dong, D. Zhang, J. Pan, W. Wang, F. Wei, Zero-shot cross-lingual transfer of neural machine translation with multilingual pretrained encoders, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 2021, pp. 15–26.

[108] Y. Li, R. Yu, C. Shahabi, Y. Liu, Diffusion convolutional recurrent neural network: Data-driven traffic forecasting, 2017, arXiv preprint arXiv:1707.01926.

[109] P. Pradhyumna, G.P. Shreya, Mohana, Graph neural network (GNN) in image and video understanding using deep learning for computer vision applications, in: 2021 Second International Conference on Electronics and Sustainable Communication Systems, ICESC, IEEE, 2021, pp. 1183–1189.

[110] M. Siino, M. La Cascia, I. Tinnirello, WhoSNext: Recommending Twitter users to follow using a spreading activation network based approach, in: 2020 International Conference on Data Mining Workshops, ICDMW, IEEE, 2020, pp. 62–70.

[111] F. Rangel, A. Giachanou, B.H.H. Ghanem, P. Rosso, Overview of the 8th author profiling task at pan 2020: Profiling fake news spreaders on twitter, in: CEUR Workshop Proceedings, Vol. 2696, Sun SITE Central Europe, 2020, pp. 1–18.

[112] C. Pérez-Almendros, L.E. Anke, S. Schockaert, SemEval-2022 task 4: Patronizing and condescending language detection, in: Proceedings of the 16th International Workshop on Semantic Evaluation, SemEval-2022, Association for Computational Linguistics, 2022, pp. 298–307.

[113] C. Pérez-Almendros, L.E. Anke, S. Schockaert, Don't patronize me! an annotated dataset with patronizing and condescending language towards vulnerable communities, in: Proceedings of the 28th International Conference on Computational Linguistics, 2020, pp. 5891–5902.

[114] A.L. Maas, R.E. Daly, P.T. Pham, D. Huang, A.Y. Ng, C. Potts, Learning word vectors for sentiment analysis, in: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Association for Computational Linguistics, Portland, Oregon, USA, 2011, pp. 142–150, URL http://www.aclweb.org/anthology/P11-1015.

[115] K. Lang, Newsweeder: Learning to filter netnews, in: Machine Learning Proceedings 1995, Elsevier, 1995, pp. 331–339.

[116] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, A. Rush, Transformers: State-of-the-art natural language processing, in: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2020, pp. 38–45.

[117] M. Siino, E. Di Nuovo, I. Tinnirello, M. La Cascia, Fake news spreaders detection: Sometimes attention is not all you need, Information 13 (9) (2022) 426.

[118] S. Alam, N. Yao, The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis, Comput. Math. Organ. Theory 25 (3) (2019) 319–335.

[119] Y. Albalawi, J. Buckley, N.S. Nikolov, Investigating the impact of pre-processing techniques and pre-trained word embeddings in detecting arabic health information on social media, J. Big Data 8 (1) (2021) 1–29.

[120] E. Alzahrani, L. Jololian, How different text-preprocessing techniques using the BERT model affect the gender profiling of authors, 2021, arXiv preprint arXiv:2109.13890.

[121] E. Araslanov, E. Komotskiy, E. Agbozo, Assessing the impact of text preprocessing in sentiment analysis of short social network messages in the Russian language, in: 2020 International Conference on Data Analytics for Business and Industry: Way Towards a Sustainable Economy, ICDABI, IEEE, 2020, pp. 1–4.

[122] H.-T. Duong, T.-A. Nguyen-Thi, A review: preprocessing techniques and data augmentation for sentiment analysis, Comput. Soc. Netw. 8 (1) (2021) 1–16.

[123] Y. HaCohen-Kerner, D. Miller, Y. Yigal, The influence of preprocessing on text classification using a bag-of-words representation, PLoS One 15 (5) (2020) e0232525.

[124] E. Haddi, X. Liu, Y. Shi, The role of text pre-processing in sentiment analysis, Procedia Comput. Sci. 17 (2013) 26–32.

[125] A.I. Kadhim, An evaluation of preprocessing techniques for text classification, Int. J. Comput. Sci. Inf. Secur.(IJCSIS) 16 (6) (2018).

[126] C. Koopman, A. Wilhelm, The effect of preprocessing on short document clustering, Arch. Data Sci. A 6 (1) (2020) 01.

[127] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, D. Brown, Text classification algorithms: A survey, Information 10 (4) (2019).

[128] P. Kumar, L. Dhinesh Babu, Novel text preprocessing framework for sentiment analysis, in: Smart Intelligent Computing and Applications, Springer, 2019, pp. 309–317.

[129] M. Kunilovskaya, A. Plum, Text preprocessing and its implications in a digital humanities project, in: Proceedings of the Student Research Workshop Associated with RANLP 2021, 2021, pp. 85–93.

[130] P. Lison, A. Kutuzov, Redefining context windows for word embedding models: An experimental study, in: Proceedings of the 21st Nordic Conference on Computational Linguistics, 2017, pp. 284–288.

[131] F. Mohammad, Is preprocessing of text really worth your time for toxic comment classification? in: Proceedings on the International Conference on Artificial Intelligence, ICAI, The Steering Committee of The World Congress in Computer Science, Computer ..., 2018, pp. 447–453.

[132] D. Petrović, M. Stanković, The influence of text preprocessing methods and tools on calculating text similarity, Facta Univ. Ser. Math. Inform. 34 (2019) 973–994.

[133] S. Pradha, M.N. Halgamuge, N.T.Q. Vinh, Effective text data preprocessing technique for sentiment analysis in social media data, in: 2019 11th International Conference on Knowledge and Systems Engineering, KSE, IEEE, 2019, pp. 1–8.

[134] M.A. Rosid, A.S. Fitrani, I.R.I. Astutik, N.I. Mulloh, H.A. Gozali, Improving text preprocessing for student complaint document classification using sastrawi, in: IOP Conference Series: Materials Science and Engineering, Vol. 874, IOP Publishing, 2020, 012017.

[135] K. Smelyakov, D. Karachevtsev, D. Kulemza, Y. Samoilenko, O. Patlan, A. Chupryna, Effectiveness of preprocessing algorithms for natural language processing applications, in: 2020 IEEE International Conference on Problems of Infocommunications. Science and Technology, PIC S&T, IEEE, 2020, pp. 187–191.

[136] M. Toman, R. Tesar, K. Jezek, Influence of word normalization on text classification, Proc. InSciT 4 (2006) 354–358.

[137] C. Zong, R. Xia, J. Zhang, Data annotation and preprocessing, in: Text Data Mining, Springer Singapore, Singapore, 2021, pp. 15–31, http://dx.doi.org/10.1007/978-981-16-0100-2_2.

[138] J. Bevendorff, B. Chulvi, E. Fersini, A. Heini, M. Kestemont, K. Kredens, M. Mayerl, R. Ortega-Bueno, P. Pezik, M. Potthast, et al., Overview of PAN 2022: Authorship verification, profiling irony and stereotype spreaders, and style change detection, in: Experimental IR Meets Multilinguality, Multimodality, and Interaction: 13th International Conference of the CLEF Association, CLEF 2022, Bologna, Italy, September 5–8, 2022, Proceedings, Springer, 2022, pp. 382–394.