# THE TEXT CLASSIFICATION PIPELINE
# STARTING SHALLOW, GOING DEEPER

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF PALERMO UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Marco Siino
December 2023

# Abstract

An increasingly relevant and crucial subfield of Natural Language Processing (NLP), tackled in this PhD thesis from a computer science and engineering perspective, is the Text Classification (TC). Also in this field, the exceptional success of deep learning has sparked a boom over the past ten years. Text retrieval and categorization, information extraction and summarization all rely heavily on TC. The literature has presented numerous datasets, models, and evaluation criteria. Even if languages as Arabic, Chinese, Hindi and others are employed in several works, from a computer science perspective the most used and referred language in the literature concerning TC is English. This is also the language mainly referenced in the rest of this PhD thesis. Even if numerous machine learning techniques have shown outstanding results, the classifier effectiveness depends on the capability to comprehend intricate relations and non-linear correlations in texts. In order to achieve this level of understanding, it is necessary to pay attention not only to the architecture of a model but also to other stages of the TC pipeline. In an NLP framework, a range of text representation techniques and model designs have emerged, including the *large language models*. These models are capable of turning massive amounts of text into useful vector representations that effectively capture semantically significant information. The fact that this field has been investigated by numerous communities, including data mining, linguistics, and information retrieval, is an aspect of crucial interest. These communities frequently have some overlap, but are mostly separate and do their research on their own. Bringing researchers from other groups together to improve the multidisciplinary comprehension of this field is one of the objectives of this dissertation. Additionally, this dissertation makes an effort to examine text mining from both a traditional and modern perspective.

This thesis covers the whole TC pipeline in detail. However, the main contribution is to investigate the impact of every element in the TC pipeline to evaluate the impact on the final performance of a TC model. It is discussed the TC pipeline, including the traditional and the most recent deep learning-based models. This pipeline consists of *State-Of-The-Art* (SOTA) datasets used in the literature as benchmark, text preprocessing, text representation, machine learning models for TC, evaluation metrics and current SOTA results. In each chapter of this dissertation, I go over each of these steps, covering both the technical advancements and my most significant and recent findings while performing experiments and introducing novel models. The advantages and disadvantages of

various options are also listed, along with a thorough comparison of the various approaches. At the end of each chapter, there are my contributions with experimental evaluations and discussions on the results that I have obtained during my three years PhD course. The experiments and the analysis related to each chapter (i.e., each element of the TC pipeline) are the main contributions that I provide, extending the basic knowledge of a regular survey on the matter of TC.

# Acknowledgments

Questo percorso di dottorato non sarebbe stato possibile senza il contributo delle persone che, in modi e tempi diversi, hanno supportato e incoraggiato il lavoro svolto. È a loro che dedico queste righe di ringraziamento.

Ringrazio il Professore Giovanni Perrone e la Dottoressa Letizia La Barbera. Entrambi hanno autorizzato e incoraggiato l'avvio e la prosecuzione del mio dottorato. Con stima e fiducia hanno sostenuto i miei studi, consapevoli dei risultati che avrei conseguito nel medio-lungo termine e ottenuti in parte anche grazie alla loro scelta gestionale. Allo stesso modo ringrazio tutti i colleghi tecnici e amministrativi del Dipartimento di Ingegneria. Non hanno mai perso occasione per dimostrare il loro affetto e interessarsi dei miei studi.

Ringrazio tutti i ricercatori senior del Laboratorio di Telecomunicazioni. Più specificamente, Stefano Mangione, Daniele Croce, Domenico Garlisi e Fabrizio Giuliano. Da loro sono stato trattato non già come uno studente o un ricercatore junior, ma come un fratello minore stimato e incoraggiato. Da loro ho ricevuto consigli e pareri. Sono stati un aiuto fondamentale nella redazione di presentazioni e articoli. Mi hanno fatto sentire parte di una squadra. Grazie a loro e ai momenti trascorsi assieme non ho mai sentito l'esigenza di ambire a situazioni e contesti lavorativi differenti. Allo stesso modo ringrazio ragazze e ragazzi che hanno lavorato nel laboratorio e che ho avuto modo di conoscere in questi ultimi tre anni.

Ringrazio il Professore Giovanni Garbo. Per avermi dato la possibilità di condurre parte dei miei studi e dei miei esperimenti nel laboratorio di Teoria dell'Informazione. Ancora oggi, durante le conferenze e i meeting di lavoro in Italia e all'estero, incontro tanti suoi ex-studenti che, a distanza di anni dall'esame di Teoria dei Segnali, sovente continuano a rimarcare la qualità e la superiorità dei suoi insegnamenti. Anche - ma non solo - nella trattazione della sua materia. In questi anni ho imparato che il Professore Garbo potrebbe essere in grado di tradire i propri pensieri e le proprie idee, ma non sarebbe mai in grado di tradire i suoi studenti.

Ringrazio il Professore Marco La Cascia, co-tutor del mio dottorato. Che con le sue azioni, le sue parole e il tempo dedicatomi, ha sempre rivelato stima e consapevolezza rispetto alle mie abilità e alle mie conoscenze. Mi ha insegnato che nella vita di un uomo c'è altro che può al contempo essere croce e delizia dell'esistenza: le barche a vela.

Ringrazio il Professore Paolo Rosso. Con i suoi innumerevoli lavori in materia di NLP e con gli shared tasks organizzati annualmente ha contribuito a ispirare e guidare gran parte dei miei studi. Ho avuto l'onore di collaborare con lui e trascorrere in sua piacevole compagnia alcuni giorni in giro per Palermo. La sua ironia (la cui detection è ancora abbastanza fuori portata anche per i modelli GPT più all'avanguardia), la sua allegria contagiosa e la sua preparazione hanno illuminato me e i miei colleghi per tutta la durata della sua visita. Ho prestato la massima attenzione a ognuno dei suoi insegnamenti e suggerimenti, molti dei quali avulsi dal dominio del machine learning. E pertanto, per certi aspetti e in diversi casi, molto più rilevanti e significativi.

Ringrazio mio papà Emanuele, tutti i miei familiari e ognuno dei miei più cari amici, con cui sento di dovermi scusare per le innumerevoli assenze, negli anni, da ascrivere a ragioni di studio e lavoro. In tempi e modi differenti, ognuno di loro mi ha fatto arrivare amore, affetto e una indiscussa ammirazione. Per questo non posso che ritenermi quotidianamente fortunato.

Ringrazio Micaela Di Carlo. Che mi è stata vicina in questi tre anni, auspicando per me la strada dell'insegnamento universitario. Abbiamo potuto condividere assieme passioni comuni per musica, film e libri, nei miei sempre più rari momenti di "tempo libero". Abbiamo trascorso assieme tante esperienze indimenticabili. Prima di tornare, qualche ora più tardi, ai miei studi e al mio lavoro.

Ringrazio la mia Maestra di ricerca e tutor del mio dottorato, Professoressa Ilenia Tinnirello. Per avermi insegnato che l'etica, la morale e il desiderio di rappresentare e onorare le Pubbliche Istituzioni non contano solo sotto i riflettori accesi. Ma anche e sopratutto dietro le quinte. Per avermi insegnato a provare imbarazzo personale di fronte a gravi mancanze istituzionali, invece di voltare la testa dall'altra parte con indifferenza e ignavia. Per aver ispirato me e i miei studi con indicazioni e insegnamenti di carattere generale e a indirizzo specifico. Per aver cercato di fare niente di meno del proprio massimo nel cercare di farmi sentire parte integrante della squadra di Ingegneria delle Telecomunicazioni - da lei ininterrottamente sostenuta e amata. Per aver caldeggiato il proseguimento delle mie attività di ricerca a Palermo, con ogni mezzo possibile. Per aver provato, con ottima approssimazione, la correttezza della mia scelta originaria: chiedere a lei di guidarmi in questo percorso di ricerca che, anche se indirettamente, ha cambiato per sempre la mia visione delle cose.

Durante il suo ultimo Gran Premio in Formula 1, nel casco del più vincente pilota di tutti i tempi erano incise le parole che sento di dover usare, ora, per ringraziare tutte le persone che mi sono state vicine in questi ultimi tre anni.

*"Life is about passion, thanks for sharing mine."* (M. Schumacher, Interlagos, 2012)

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Overview

In several Natural Language Processing (NLP) applications like news categorization, sentiment analysis, and subject labelling, Text Classification (TC) is a crucial and relevant task. The goal of TC is to tag or label textual components like sentences, questions, paragraphs, and documents. In this era of massive information dissemination, manually processing and categorizing huge amounts of text data takes a relevant quantity of effort and time. To name a few, text information can be found on social media, websites, chat rooms, emails, questions and answers from customer service representatives, insurance claims and user reviews. Furthermore, human factors such as skills and fatigue can have a relevant influence on the effectiveness of manual TC. It is preferable to automate the TC pipeline involving machine learning models to get objective outcomes. Furthermore, to reduce the problem of information overloading, the improvement of information retrieval effectiveness can help in finding the necessary information for a certain task. In Figure 1.1 is illustrated a flowchart of the steps involved in TC, under the light of traditional and most recent machine learning models. A critical first stage is the preprocessing of the text to provide as input to the model. Classical approaches usually employ AI methods to collect relevant features, which are then classified with machine learning techniques. Next, the text representation approach can severely impact the outcomes of the model. Involving a series of transformations used to directly map a source text to predicted labels, deep learning, as opposed to traditional models, incorporates feature engineering into the process of training of the model. Up until 2010, classical TC models were the most used and popular. Traditional approaches are still today among the most popular ones. Some of them are Logistic Regressor (LR), Naïve Bayes (NB), Support Vector Machine (SVM) and K-Nearest Neighbour (KNN). Such methods clearly outperform past rule-based techniques in consistence and accuracy. However, they still need feature engineering and they are time-consuming. Additionally, it is hard to understand the semantic of the words since they frequently neglect the context or natural

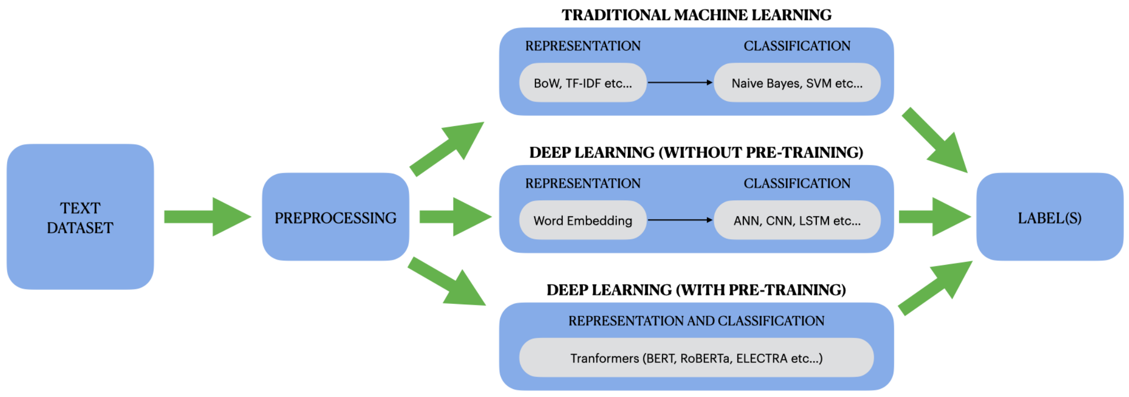

TRADITIONAL MACHINE LEARNING
REPRESENTATION
CLASSIFICATION
BoW, TF-IDF etc...
Naive Bayes, SVM etc...
DEEP LEARNING (WITHOUT PRE-TRAINING)
REPRESENTATION
CLASSIFICATION
Word Embedding
ANN, CNN, LSTM etc...
DEEP LEARNING (WITH PRE-TRAINING)
REPRESENTATION AND CLASSIFICATION
Tranformers (BERT, RoBERTa, ELECTRA etc...)
TEXT DATASET
PREPROCESSING
LABEL(S)

Figure 1.1: The TC pipeline, considering traditional and modern approaches.

sequential arrangement of textual material. In TC, deep learning algorithms gradually took the place of traditional techniques by the 2010s. Deep learning techniques for text mining automatically construct semantically pertinent representations without the need for humans to define rules and features. Consequently, the majority of TC activities are focused on deep neural networks. Fewer and fewer studies focus on using traditional models to overcome computational and data restrictions.

Most conventional machine learning models use a two-step procedure. First, the documents are stripped of a number of manually added features (or any other textual unit). In the following phase, a classifier receives these features so it can produce a prediction. The Bag of Words (BoW) feature and its extensions are frequently created by hand. Hidden Markov Models, NB, SVM, Random Forests (RF) and Gradient Boosting (GB), are some common classification algorithms employed in the second step. Numerous disadvantages exist with the two-step approach. For instance, using handcrafted features and expecting acceptable performance requires time-consuming feature engineering and analysis. Due to the strategy's heavy reliance on domain expertise for feature generation, it is also difficult to adapt it to new applications. Last but not least, because of the very specific features domain, these models cannot fully benefit from the vast volumes of training data available. To address the issues with the use of handcrafted features, the use of neural approaches has increased. The main component of these approaches is an embedding space, where text is encoded as a low-dimensional continuous features vector without the need for traditional features representation strategies. The Latent Semantic Analysis (LSA) proposed in [156] is one of the earliest studies on embedding models. The proposed architecture is trained on 200K words and has fewer than 1 million parameters. In [29], the first neural language model was proposed. It consisted of an artificial neural network trained on over 10 million words. When progressively larger embedding models were constructed with significantly more training data, a paradigm change occurred. A number of Word2Vec models that Google creates in 2013 [198] were trained using billions of words and quickly gained popularity for numerous NLP applications. As the basis for their contextual embedding model, the researchers from AI2[1] and the University of Washington created a Bidirectional-Long Short Term Memory (BiLSTM) network using 93 million hyperparameters and a training performed on a billion of words in 2017. A novel model named Embedding from Language Models (ELMo) [226] captures contextual information and performs significantly better than Word2Vec because. This subsequent development results in the construction of embedding models using Google's new neural architecture, Transformer [286]. Transformer is entirely attention-based, which significantly boosts the effectiveness of extensive model training on Tensor Processing Unit (TPU). In the same year, Google creates the Bidirectional Encoder Representations from Transformers (BERT) [73]. BERT has 340M parameters and was trained on 3.3 billion words. More training data and larger models are proposed in the literature every day. The most recent OpenAI GPT-3 model has 170 billion parameters [67] and it is based on Transformers. Some academics contend that despite the enormous models' remarkable performance on different NLP tasks, they do not truly grasp language and are insufficient for many domains that are mission-critical [120, 187]. Recently, there is a rise of interest toward neuro-symbolic hybrid models to solve significant flaws of neural models like interpretability, inability to use symbolic thinking and lack of grounding [251, 90].

Although there are many excellent reviews and textbooks on TC techniques and applications, this PhD thesis provides a thorough analysis of all the phases that go into creating a TC pipeline with several contributions, including novel and deep experiments to further investigate the impact on the performance of each stage of the pipeline. These contributions are usually reported at the end of each chapter. Even if specific languages are considered in the related works, from the standpoint of computer science, English is the language that is most frequently used and referred in the present literature regarding TC. Furthermore, most of the large language models and pre-trained word embeddings are originally developed focusing on English, partially or totally neglecting the other languages. The rest of this PhD thesis primarily uses the English as the reference language for many of the examples and cases presented and discussed.

Starting with a discussion on some of the more contemporary tasks — such as author profiling, topic classification, news classification, sentiment analysis — I then move on to the metrics used to evaluate performance on the SOTA datasets, and then I present SOTA models and most recent and relevant findings. I also cover the most recent deep neural network architectures, which are divided into a number of types based on their functioning, including Transformers, Convolutional Neural Networks (CNNs), Capsule Nets and Recurrent Neural Networks (RNNs). For each chapter presented here, I discuss experiments and findings related to the step in the TC pipeline.

The following are the main Research Questions (RQs) addressed in this PhD thesis. The RQs are usually addressed at the end of the chapters and supported by experiments, results, quantitative and qualitative analysis and discussions. Other minor findings are also presented and discussed in each chapter, even if they are not listed here as RQs.

[1]`https://allenai.org/allennlp/software/elmo`

- **RESEARCH QUESTIONS (RQs)**
  - **(RQ1) Chapter 2**: Does data augmentation help at improving the performance of TC models?
  - **(RQ2) Chapter 3**: Does text preprocessing have an impact also on the performance of modern classification models (e.g., Transformers)?
  - **(RQ3) Chapter 4 and 5**: After pre-training from scratch a word embedding-based model, what happens to the trained word vectors in the embedding space considered? Is it possible to explore and to understand the trained embedding space to drive the development of a simple deep learning model able to obtain SOTA results?

My thesis is organized as follows: Chapter 2 presents the most common datasets used and available in the literature and the most used metrics. Then I propose and discuss a data augmentation strategy to improve the performance of a classifier. In Chapter 3, the preprocessing technique to prepare raw text are presented and discussed. There, I further investigate and evaluate the impact of the most common techniques on SOTA models and datasets. In Chapter 4 the methods to represent text in a numerical way are reported. In this chapter I also propose a strategy based on PCA, to visualize and analyze a word embedding space trained from scratch. In Chapter 5, traditional and modern classifiers commonly employed for TC are discussed, including some of my findings and results concerning a signal analysis through the layers of a shallow CNN. In Chapter 6 some real-world applications from my previous works are presented along with the results and the relevant findings. In Chapter 7 the conclusions and the future perspectives are presented. The contributions and a summary for each chapter of this thesis are reported in what follows.

## 1.2 Outline and contributions

Several works have investigated TC techniques from a general standpoint. I specifically mention the work in [162], which offers a thorough analysis of model architectures, spanning from traditional to the modern deep learning-based ones. The survey by [148] offers a great examination of preprocessing procedures, including feature extraction and dimensionality reduction. However, despite including quantitative outcomes of conventional approaches, [200] focuses on deep learning models. By providing a view of each stage required to design a TC model, this thesis seeks to enhance the landscape of TC from a general point of view. As a result, I give a thorough explanation of the key data preparation procedures used along with TC models. Although this part of the pipeline is frequently disregarded, developing a useful framework for this task requires an understanding of how they are used and the reasons behind the decisions taken. I provide model descriptions from traditional ones to deep learning-based ones from more recent years, in contrast to prior TC evaluations. The design of the classifier and feature extraction are highlighted for the traditional models.

Once the text possesses well-designed properties, the classifier may be trained to converge quickly.

A specific summary for each chapter of this thesis is reported to conclude this section. Along with the background on the pipeline stage involved, the last part of each chapter is dedicated to the main contributions I have provided, supported by new experiments, models and/or methods proposed, quantitative and qualitative analysis.

## Summary of Chapter 2: Challenges, datasets and dataset pre-analysis in TC

In the early history of machine learning, information retrieval systems primarily used TC algorithms. But as technology has developed over time, TC and document categorization have become widely employed in several fields, including law, engineering, social sciences, healthcare, psychology, and medicine. I highlight some domains that use TC algorithms in this section. Some TC tasks are introduced in this chapter, including three new datasets related to emerging author profiling tasks. The datasets available in the literature and related to these tasks and usually employed as benchmark, are also reported and presented in this chapter. Then the evaluation metrics usually employed to assess a model performance on these datasets are presented.

### Contribution

The contributions for this section are two. With the first, I present and discuss a strategy for a preliminary linguistic analysis of a dataset. Such an analysis can eventually drive subsequent choices in the development of the steps involved in the TC pipeline. With the second contribution, I introduce and discuss a novel data augmentation technique based on backtranslation. Thanks to this data augmentation strategy, I found in some of my recent studies, that the TC model performance can be improved on several tasks.

## Summary of Chapter 3: Text preprocessing

In this chapter I collect, report and discuss the text preprocessing techniques found in the literature and their possible and most recent variants, proposing a nomenclature standard based on acronyms. I also provide the reader with useful information for self-study and in-depth study of the techniques presented along with advices on how to operate educated choices to select the preprocessing technique (or combination of techniques) given a specific task, model, and dataset.

### Contribution

My contributions are reported in the last section and concern several experiments. Specifically, I select the three most common techniques used in the literature to evaluate the impact of each of these techniques (alone or in combination) on the classification results of nine SOTA models (pre-trained deep, deep and non-deep) and on real world datasets. Then I evaluate how text preprocessing can affect the performance of modern pre-trained architectures based on attention (i.e., Transformers) compared to traditional ones. Finally, I determine if simple classifiers' performance are comparable to the ones obtained by Transformer-based models when text preprocessing is performed in accordance with the specific model and dataset used.

## Summary of Chapter 4: Text representation

Before moving to the classification stage, it is necessary to convert unstructured data, especially free-running text data, into organized numerical data. To do this, a document representation model must be used to employ a subsequent classification system following the text preprocessing stage. Text representation models convert text data into a numerical vector space, which has a substantial impact on how well subsequent learning tasks can perform. In the history of NLP, word representation has always been a topic of interest. It is crucial to properly represent such text data, since it contains a wealth of information and may be applied broadly across a variety of applications. This chapter examines the expressive potential of several word representation models, ranging from the traditional to the contemporary SOTA word representation provided by large language models.

The chapter discusses numerous representation models that are frequently employed in the literature. In the past, several researchers explored various theories to solve the issue of words losing their syntactic and semantic links with the chosen representation. The literature review, which demonstrates how numerous strategies have been used for a plethora of NLP-related tasks, is offered along with these techniques. Before discussing well-known representation learning and pre-trained language models, I first discuss various statistical models.

### Contribution

In the last section of this chapter is reported my contribution about the analysis of a trained word embedding for a specific TC task. Thanks to a Principal Component Analysis (PCA) tool, it is shown and analyzed the effect of a CNN training on a 3D visualization of a word embedding space. This way I am able to understand some implicit choices operated during the training of the model, to assign specific word vectors to certain keywords belonging to one of the two class labels used for the task.

## Summary of Chapter 5: Text classification methods

In Chapter 5 are reported both the traditional methods for TC and the most modern ones based on deep learning. Models discussed in this chapter belong to three different groups. The non-deep learning deterministic models, the not pre-trained deep learning models and large pre-trained language models known as Transformers. The term "earlier approaches" refers to all techniques used before the advent of deep neural networks, when the prediction was based on manually created features. Neural networks with only a few hidden layers are also included in this category, and these are so-called "shallow" networks. These methods replace several rule-based ones, which they outperformed in terms of accuracy. The most recent deep learning models, which have an impact on all artificial intelligence domains, including TC, are also discussed. These techniques have become popular because they can simulate intricate features without requiring manual engineering, which reduces the need for subject expertise.

### Contribution

In the last section, I present and discuss real-world competitive models that I designed and developed to address some SOTA task about TC. Finally, I present some approaches I used to perform a post-hoc analysis on a SOTA deep model to explore the results of the predictions provided. I conclude the chapter with a signal analysis of the CNN layer's output to understand the behavior of the network, either during the training phase and during the inference phase. I propose a methodology to further investigate the behavior of a deep learning model, looking also at its predictions and at the outputs provided by the intermediate layers of the model. The analysis presented was conducted focusing on a fake news spreaders dataset to explore the behavior of a shallow CNN. To perform this exploration, I looked at the predictions provided after completing the training phase [256]. This further step can be employed in the TC pipeline to improve the model performance and for a deeper understanding of its behaviors.

## Summary of Chapter 6: Applications for competitive tasks

This chapter focuses on TC while facing real-world applications. After introducing all the stages of the TC pipeline in the previous chapters, this chapter presents some architectures that encompass all these stages. These architectures are evaluated on emerging tasks and SOTA benchmarks to correlate the architectures and functioning of the models with the results obtained in specific tasks. After an introduction concerning the most popular international shared tasks on TC and NLP in general, in the chapter I report all the architectures I developed to address several recent TC challenges, providing some background on some modern international challenges of interest for the NLP community.

### Contribution

I developed the architectures presented in this chapter for obtaining — and in some case for outperforming — SOTA results on the proposed tasks. These tasks involve the automatic detection of several types of content or discourse. Specifically, the proposed models address the detection of hate speech, fake news, irony and stereotypes, harmful tweets and patronizing and condescending language. At the beginning of any section, a short background on the specific task is provided. Along with the novel architectures proposed, in the chapter are also presented and discussed the results obtained on each task. The challenges are related either to binary or multi-label classification tasks. Thanks to these real-world applications, I was able to better understand and correlate the impact of each element in the TC pipeline, to the performance and to the results obtained during my PhD studies.

## Summary of Chapter 7: Conclusions and future perspectives

In the last chapter of this thesis, I report my final conclusions and future perspectives on the matter.

# Chapter 2

# Tasks and datasets

The practice of collecting texts (such as tweets, news articles, and customer reviews) into groups is known as TC. Topic classification, news categorization and sentiment analysis are examples of typical TC tasks. Recent studies demonstrate that by giving text classifiers the ability to receive as input pairs of texts, it is effective to cast various natural language understanding tasks — such as natural language inference and extractive question answering — as TC. However, they do not generally work on a finite and predefined set of labels, so they do not properly fit into the TC field. In the first section of this chapter, some popular TC tasks from the literature are introduced.

The accessibility of labelled dataset for TC has emerged as the primary impetus for this study area's rapid development. The dataset, presented in this chapter, are frequently used as benchmarks in the TC-related literature. In this introduction part, I list the domain-specific properties of these datasets and provide an overview in the Table 2.1 that shows the task description, the overall sample count, the number of target classes, and articles presenting the corresponding dataset.

The TC tasks presented here are:

- Author profiling
- Topic classification
- News classification
- Sentiment analysis

In the last section of this chapter is presented a methodology to analyze and evaluate the dataset considered from a linguistic point of view. Such a pre-analysis can drive the next steps in the classification pipeline. Furthermore, I propose a data augmentation strategy based on backtranslation to make explicit, automatically, some latent semantic available in a text.

Table 2.1: Dataset characterization and stats.

| Dataset | Task | #Total documents | #Number of classes | Reference |
|---|---|---|---|---|
| FNS | Author profiling | 500 | 2 | [216] |
| HSS | Author profiling | 600 | 2 | [239] |
| ISS | Author profiling | 600 | 2 | [32] |
| MR | Sentiment analysis | 10,662 | 2 | [215] |
| SST1 | Sentiment analysis | 11,855 | 5 | [265] |
| SST2 | Sentiment analysis | 9,613 | 2 | [265] |
| MPQA | Sentiment analysis | 10,606 | 2 | [71] |
| IMDB | Sentiment analysis | 50,000 | 2 | [181] |
| Yelp2 | Sentiment analysis | 290,000 | 2 | [311] |
| Yelp5 | Sentiment analysis | 700,000 | 5 | [311] |
| Amazon2 | Sentiment analysis | 4,000,000 | 2 | [311] |
| Amazon5 | Sentiment analysis | 3,650,000 | 5 | [311] |
| Google News | News classification | 190,000 | 2 | [68] |
| Reuters news | News classification | 10,788 | 90 | URL[1] |
| 20NG | News classification | 376,420 | 20 | URL[2] |
| AG News | News classification | 127,600 | 4 | URL[3] |
| Sogou | News classification | 2,909,551 | 5 | URL[4] |
| PCL | Topic classification | 10,637 | 2 | [225] |
| DBpedia | Topic classification | 630,000 | 14 | [159] |
| Ohsumed | Topic classification | 7,400 | 23 | URL[5] |
| ISTO | Topic classification | 44,898 | 2 | URL[6] |
| EUR-Lex | Topic classification | 19,314 | 3,956 | [179] |
| Yahoo! | Topic classification | 1,460,000 | 10 | [311] |
| WOS | Topic classification | 46,985 | 134 | [147] |

## 2.1 Research areas

### 2.1.1 Author profiling

One of the three main areas of Automatic Authorship Identification (AAI), together with authorship attribution and authorship identification, is author profiling.The development of AAI started took shape at the turn of the 20th century. The approach was initially used on the writings of Francis Bacon, William Shakespeare, and Christopher Marlowe by American self-taught physicist and meteorologist Thomas Corwin Mendenhall. Mendenhall compared the word lengths of these three historical leaders in an effort to identify any quantitative stylistic variations.

Author profiling consists in the analysis of a corpus of texts in an effort to determine the author's identity or to identify distinct traits of the author based on stylistic and content-based factors. Age and gender are frequently analyzed factors, but more recent researches have also looked at additional aspects like personality traits and career [296]. Author profiling is useful in many sectors where it is necessary to identify particular traits of a text's author, with a rising focus on forensics and marketing. The task of author profiling might vary depending on its application in terms of the traits to be identified, the number of authors researched, and the quantity of texts available for analysis. Although generally restricted to written writings, such as literary works, this has expanded to include online texts as computers and the Internet have developed.

Despite significant advancements in the twenty-first century, author profiling is still a challenging process that has not yet been fully resolved. Below are some well-known author profiling datasets that can be found in the recent literature.

- **Fake News Spreaders (FNS)**.The FNS dataset is presented and discussed in [216] and available under request[7]. The dataset was used for the international shared task at PAN[8]. 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 dataset consists of tweets in Spanish and English. 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 and one hundred authors per label in the train and in the test set, respectively. Resuming, the dataset samples (i.e., 500 authors) provide a total number of 50,000 tweets. The results of the participants at the FNS task are available online[9].

- **Hate Speech Spreaders (HSS)**.The HSS dataset is presented and discussed in [239]. As a first step in preventing hate speech from spreading among online users, the task's organizers aim to identify potential Twitter users who spread hate speech.

[7]`https://zenodo.org/record/4039435`
[8]`https://pan.webis.de`
[9]`https://pan.webis.de/clef20/pan20-web/author-profiling.html`

The dataset consists of tweets in Spanish and English. In the dataset, there are two hundred tweets representing each author and the corresponding class label for the author (i.e., 1 if the author has shared hate speech in the past, and 0 otherwise). For each language, there are one hundred authors and fifty authors per class in the train and in the test set, respectively. Resuming, the dataset samples (i.e., 600 authors) provide a total number of 120,000 tweets. The results of the participants at the HSS task are available online[10].

- **Irony and Stereotype Spreaders (ISS)**.The ISS dataset is presented and discussed in [45, 32] and available under request[11]. The dataset was used for the international shared task at PAN[12]. The task's organizers want to focus on irony. Especially when words are used subtly and figuratively to indicate the opposite of what is literally expressed. A more violent version of irony, sarcasm aims to mock or ridicule a target without necessarily restricting the possibilities of hurting them. The objective is profiling users whose tweets can be labelled as sarcastic.

  A group of 600 Twitter authors make up the dataset that the PAN organizers have created. Two hundred tweets are provided for each author. Each author is represented by a unique XML file with 200 tweets. Four hundred and twenty authors made up the organizers' labelled train set. In the test set, there are 180 further ones. The train set's authors are identified by the letters "I" (ISS) or "NI" (nISS). The results of the participants at the task are available online[13].

### 2.1.2 Topic classification

Topic categorization, commonly referred also as *topic analysis*, seeks to pinpoint a text's main theme or themes (for instance, whether a product review is related to "easy of use" or "customer assistance"). The complex text theme is defined in the topic analysis in an effort to determine the text's meaning. The assignment of themes to documents, known as topic labelling, is a significant component of the technique aimed at facilitating the process of topic analysis. Here I list several SOTA datasets.

- **Patronizing and Condescending Language (PCL)**. Described in detail in [225], the dataset for (PCL) is from the detecting PCL task hosted at SemEval-2022. Such a task is an emerging one about detecting PCL [224]. 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

[10]`https://pan.webis.de/clef21/pan21-web/author-profiling.html`
[11]`https://zenodo.org/record/6514916`
[12]`https://pan.webis.de`
[13]`https://pan.webis.de/clef22/pan22-web/author-profiling.html`

good intentions. In order to complete the assignment, a classifier must determine whether PCL is present in a given text. The dataset is available on GitHub[14].

- **DBpedia**. Wikipedia's most frequently used info boxes were used to create the DBpedia [159], a sizable multilingual knowledge library. Every month, it releases a new edition of DBpedia, adding or removing classes and attributes. The most widely used version of DBpedia comprises 14 classes, 560,000 and 70,000 records, for training and for testing respectively.

- **Ohsumed**. The Ohsumed[15] has a MEDLINE database affiliation. There are 23 categories for cardiovascular diseases and 7,400 texts overall. All texts are classified into one or more classes and are abstracts of medical information.

- **ISTO Fake News dataset**. The dataset[16] contains two types of articles: fake and real news. This dataset was collected from real world sources; the truthful articles were obtained by crawling articles from Reuters.com (News website). As for the fake news articles, they were collected from different sources. The fake news articles were collected from unreliable websites that were flagged by Politifact (a fact-checking organization in the USA) and Wikipedia. The dataset contains different types of articles on different topics, however, the majority of articles focus on political and World news topics.

- **EUR-Lex**. The EUR-Lex dataset [179] consists of several document categories that are indexed in accordance with a number of orthogonal categorization systems to enable a variety of search functions. With 19,314 documents and 3,956 categories, the most widely used variant of the dataset is based on various parts of EU laws.

- **Yahoo! Answer**. The Yahoo! Answer[17] dataset [311] is about topic labelling with 10 different classes. Per class, there are 6,000 and 140,000 samples to test and train respectively. Three components, referred to as question titles, question contexts, and best responses, are included in every sentence.

- **Web Of Science (WOS)**. The WOS dataset [147] is a set of information and meta-information about articles that is available via Web of Science, the most reputable global citation database, regardless of the publisher. There are three variants of WOS: WOS-46985, WOS-11967, and WOS-5736. The full dataset name is WOS-46985. WOS-46985 has two subsets: WOS-11967 and WOS-5736.

[14]`https://github.com/Perez-AlmendrosC/dontpatronizeme`
[15]`https://davis.wpi.edu/xmdv/DSs/ohsumed.html`
[16]`https://www.uvic.ca/ecs/ece/isot/DSs/fake-news/index.php`
[17]`https://www.kaggle.com/DSs/soumikrakshit/yahoo-answers-DS`

### 2.1.3 News classification

News classification is the process of automatically categorizing news information according to predetermined tags, whose accuracy is derived from the training news records and is based on their content. Business, entertainment, politics, sports, technology, and other fields can be used to categorize news items. Users can find news articles they are interested in this way with the use of a news classification system, saving time and removing news overhead.

One of the most significant information source is news content. Users can obtain essential knowledge instantly thanks to the news classification system. The duty of categorizing news items according to topics or user interests is of utmost importance. By depending on user preferences, spotting new news topics or recommending material of interest, a news classification model aids people in obtaining real-time information. Here, I go into the detail of a few common datasets.

- **Google News**. The Google News dataset presented in [68] is made up by two datasets. The first one consists of a subset of clicks received on the Google News website over a certain time period, from the top 5000 users (top as sorted by the number of clicks). There are about 40,000 unique items that are part of this dataset and about 370,000 clicks. The second dataset is similar to the previous one (in fact a superset), and just contains more records: 500,000 users, 190,000 unique items and about 10,000,000 clicks. In order to have uniformity in comparisons, authors binarize the first dataset as follows: if the rating for an item, by a user, is larger than the average rating by this user (average computed over her set of ratings) they assign it a binary rating of 1, 0 otherwise.

- **Reuters news**. The Reuters-21578 dataset[18] is often used for text categorization. It was gathered by the Reuters economic press release service in 1987. A version of Reuters-21578 with multiple classes containing 10,788 documents is called ApteMod. 90 lessons, 7,769 training samples, and 3,019 test samples are included. R8, R52, RCV1, and RCV1-v2 are additional datasets generated from a portion of the Reuters dataset.

- **20 Newsgroup (20NG)**. The 20NG dataset[19] consists of newsgroup documents that were posted on 20 various themes. For text categorization, text clustering, and other tasks, different variations of this dataset are employed. One of the most often used versions has 18,821 papers, evenly distributed among all topics.

- **AG News**. The AG News dataset[20] consists of news articles compiled by academic news search engine ComeToMyHead from more than 2,000 news sources. It makes advantage of each news story's title and description fields. A total of 120,000 training texts and 7,600 test texts are included in AG. Each sample consists of a brief sentence that has a four-class label.

[18]`https://martin-thoma.com/nlp-reuters`
[19]`http://qwone.com/~jason/20Newsgroups/`
[20]`http://groups.di.unipi.it/~gulli/AG_corpus_of_news_articles.html`

- **Sogou**. The SogouCS and SogouCA news sets are included in the Sogou[21] dataset, which combines both of them. The name of the domains within the URL serve as the labels for each text. So, as the classification labels for the news, the domain names in their URLs are used. For illustration, the news at `http://sports.sohu.com` is classed under the sports category.

### 2.1.4 Sentiment analysis

Sentiment analysis, commonly referred to as *opinion mining* or *emotion AI*, is the systematic identification, extraction, quantification, and study of affective states and subjective data using natural language processing, text analysis, computational linguistics, and biometrics. Sentiment analysis is frequently used in marketing, customer service, and clinical medical applications. It is used to voice of the customer materials including reviews and survey replies, internet and social media, and healthcare materials.

This category of tasks entails identifying the polarity and viewpoint of user's ideas in text (such as tweets, movies, or product reviews). In contrast to the conventional TC, which examines the text's objective content, it is essential to learn whether the text supports a particular point of view. Understanding emotional states and subjective information in a text, which is frequently categorized in terms of evoked emotions, can also be part of the work. Either a binary problem or a multi-labels task can be used to model the work. Sentiment analysis, when focused on binary TC tasks, splits texts into negative and positive classes, as opposed to multi-class sentiment analysis, which groups texts into multiple labels. Here, I present the detail of some of the most common datasets used in the literature as benchmarks.

- **Movie Review (MR)**. The MR dataset[215] is a set of film reviews that was created with the goal of identifying the sentiment attached to each user review and deciding whether it is positive or negative. There is a sentence for each review. There are 5,331 positive samples and 5,331 negative samples in the corpus.

- **Stanford Sentiment Treebank (SST)**. The SST dataset [265] extends MR. It has two categories: one with binary labels and the other with fine-grained (five-class) labels. Namely, SST-1 and SST-2, respectively. There are 8,544/1,101/2,210 samples, in train/dev/test set respectively for a total of 11,855 movie reviews in SST-1. SST-2 is divided into train, dev and test sets, with respective sizes of 6,920, 872, and 1,821.

- **Multi-Perspective Question Answering (MPQA)**. The MPQA is an opinion dataset [71]. It also has two class labels and an MPQA dataset of opinion polarity detecting sub-tasks. In total, 10,606 phrases from news stories from various news sources are included in MPQA. It should be noticed that there are 7,293 negative texts and 3,311 positive texts, all without text labels.

[21]`https://huggingface.co/DSs/sogou_news/blob/main/README.md`

- **Internet Movie Database (IMDB)**. A dataset for binary sentiment classification was first described in [181] as the IMDB dataset. 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. 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 positives and 25,000 are negatives). For unsupervised learning, an additional 50,000 unlabeled documents are included. The IMDB dataset is available online[22].
- **Yelp**. The Yelp reviews dataset [311] comes from the 2015 Yelp dataset Challenge. 1,569,264 of the samples in this dataset include review texts. From this dataset, two classification tasks are created: one predicts the total amount of stars that a buyer has provided, and the other predicts whether a star's polarity is positive or negative. The first dataset has 650,000 and 50,000 samples for train and test respectively, and 280,000 training samples and 10,000 test samples for each polarity in the polarity dataset.
- **Amazon**. A well-known corpus known as the Amazon dataset was created by gathering product reviews from the Amazon website [311]. There are two categories in this dataset. There are 3,600,000 and 400,000 samples in the train and in the test sets in the Amazon-2 with two labels. For training and testing purposes, Amazon-5, which has five classes, has 3,000,000 and 650,000 comments.

## 2.2 Metrics

The *F1 score* and *accuracy* are two metrics often employed to gauge the effectiveness of TC models. Later, the assessment metrics are improved due to the complexity of the classification tasks or the existence of some specific activities. Single-label TC separates samples in one of the categories that are most likely to be used in NLP tasks. It is possible to ignore the relationships between labels in single-label TC because each text only belongs to one category. Multi-label TC, as opposed to single-label TC, breaks the corpus up into various category labels which depends on the task. These metrics were created for single label TC and are therefore inappropriate for multi-label jobs. Therefore, some metrics have been created for multi-label TC. Before introducing the metrics reported in the literature, below I provide the definitions of the terms used in the following equations.

- **True Positive (TP)**. A single prediction provided by a classifier is referred to as a TP when the model *correctly* predicts a positive class.
- **True Negative (TN)**. A single prediction provided by a classifier is referred to as a TN when the model *correctly* predicts a negative class.

[22]https://ai.stanford.edu/~amaas/data/sentiment/

| | | **Actual** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted** | Positive | TP | FN |
| | Negative | FP | TN |

Figure 2.1: A confusion matrix.

- **False Positive (FP)**. A single prediction provided by a classifier is referred to as an FP when the model *incorrectly* predicts a positive class.
- **False Negative (FN)**. A single prediction provided by a classifier is referred to as an FN when the model *incorrectly* predicts a negative class.

In Figure 2.1 is shown a *confusion matrix* [268]. A confusion matrix, also known as an *error matrix*, is a table structure which allows visualizing the performance of an algorithm, often a supervised learning one, in machine learning and, more specifically, the problem of statistical classification — in unsupervised learning it is usually called a matching matrix. Both variations of the matrix, where each column represents instances in the class predicted, and each row represents the actual class instances, are documented in the literature. The name was chosen since it is simple to determine whether the system is conflating two classes (i.e., commonly mislabelling one as another). It is a unique type of contingency table with two dimensions (actual and expected), identical sets of "classes" and two dimensions (each combination of dimension and class is a variable in the contingency table).

Given the above definitions, the following are the common metrics used in literature for several TC tasks.

**Accuracy**. Accuracy is the ratio of correct predictions on the total observations and is given by the Equation 2.1. Accuracy is one way to measure what percentage of predictions are right.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.1}$$

**Error rate**. Closely related to Accuracy is the *Error rate*. The definition is given by the Equation 2.2. The error rate expresses what percentage of predictions are wrong.

$$ErrorRate = 1 - Accuracy = \frac{FP + FN}{TP + TN + FP + FN} \tag{2.2}$$

Depending on how genuine positives and negatives are defined in a multilabel scenario, the definition of this metric may differ. A prediction is deemed accurate (referred to as "subset accuracy") when the projected labels exactly match the actual labels. Alternately, before the accuracy calculation, predictions can be flattened and condensed to a single-label task.

**Precision**. Equation 2.3 defines *precision* or *sensitivity* as the ratio of true positive (TP) observations to all-around positive predicted values (TP+FP). Precision is the proportion of correctly predicted events among all positively predicted events.

$$Precision = \frac{TP}{TP + FP} \tag{2.3}$$

**Recall**. Equation 2.4 gives *recall* or *specificity* as the ratio of true positive (TP) observations to all-around actual positive values (TP+FN). Recall is the ratio of right predictions made over all positive predictions that should have been made.

$$Recall = \frac{TP}{TP + FN} \tag{2.4}$$

For scenarios involving multi-class classification, it is possible to compute the precision and recall for each class label.

**F1 score**. Equation 2.5 illustrates the F1 score, which is the harmonic mean of recall and precision. The maximum precision and recall value of an F1 score is 1, while the lowest value is 0.

$$F1 = 2 \times \frac{Recall \times Precision}{Recall + Precision} \tag{2.5}$$

**Matthews Correlation Coefficient (MCC)**. The effectiveness of binary classification techniques is also measured by the Matthews Correlation Coefficient (MCC) [188], which collects all the data in a confusion matrix. MCC can be used to address issues with unequal class sizes and is still regarded as a balanced approach. The MCC scales from -1 to 0. (i.e., the classification is always wrong and always true, respectively). Equation 2.6 provides the formula for MCC.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.6}$$

Finally, some specific metrics related to multilabel tasks are Micro and Macro-F1 [185], and Precision@k and Normalized Discounted Cumulated Gains [169].

## 2.3 Dataset analysis

The example of analysis presented in this section was originally conducted in [256] and is based on the FNS dataset. The discussion about the tools and the methods described in this section can help for a better development of the next stages in the classification pipeline.

The Profiling Fake News Spreaders Task (PFNSoT) dataset is a multilingual dataset made of Spanish and English tweets. For each language, the data collected is made of 100 tweets per author and 150 authors per class (i.e., FNS and nFNS) in the training set, and 100 authors per class in the test set. I decided to use the PFNSoT dataset for two main reasons: PAN has a long tradition in organizing shared tasks; my extensive tests on several SOTA models are in this way comparable with the other task participants' results.

Although task organizers encouraged the submission of multilingual models, submissions of models dealing only with one language were also allowed. As reported in the task overview, participant results were lower for the English language in terms of binary accuracy. To further explore the dataset, I quantitatively and qualitatively investigated it using established corpus linguistics methods, implemented in the online well-known corpus linguistics tool, Sketch Engine[23] [138].

### Compare Corpora

In this subsection, I report a quantitative description of the Spanish and English datasets. Since I used corpus linguistics tools to carry out the analysis, in this section I use the term *corpus* (plural, *corpora*) to refer to each dataset. Table 2.3 provides a high-level description of the Spanish and English corpora, in terms of number of tokens identified in the tweets written by the same typology of authors. The corpora are also divided into subcorpora, class-wisely grouping tweets released as training and test data. In the table, each corpus is labelled by specifying the language, the class, and the partitioning criterion of the corpus. For example, the corpus es_train_0 collects the Spanish tweets (**es**_train_0) contained in the training set (es_**train**_0) written by nFNS authors (es_train_**0**), while es_1 to the totality (training and test sets) of tweets in Spanish written by FNS. While in the Spanish corpus there is a relevant difference in size between corpus 0 and corpus 1—and this difference in size is kept also in the training and test data—it is not the case in the English DS, in which the two classes have almost the same number of tokens both in train and test data. However, the difference in size in the Spanish dataset is not as big as to prevent corpus comparison in terms of common tokens (i.e., similar linguistic register used by the authors). For this comparison, I applied a chi-square ($X^2$) test [136] by using the built-in function of Sketch Engine, *Compare Corpora.* Thus, I compared train_0, train_1, test_0, and test_1 of both languages. In this way, I obtained two confusion matrices, reported in Figure 2.2, showing values greater of or equal to 1, with 1 indicating identity. The higher the value, the larger the difference between the two compared subcorpora.

**Spanish Corpus Matrix.** I assumed 1.74 as the reference measure for all the other comparisons, since it indicates the difference between train_0 and train_1, i.e., the data that models use for training. As reported in this matrix, the similarity measure between test_0 and train_0 is 1.36, which is 0.38 points smaller than the reference measure. The same applies to test_1 and train_1: their similarity measure is 1.41, which is 0.33 points smaller than the reference measure. The fact that the difference between the reference measure and the class-wise train and test similarity measure is a bit higher in nFNS might indicate that FNS are slightly more difficult to identify. In addition, it is worth noticing that, since the similarity measure between train_0 and test_1 (i.e., 1.57) is smaller than

[23]`https://www.sketchengine.eu`

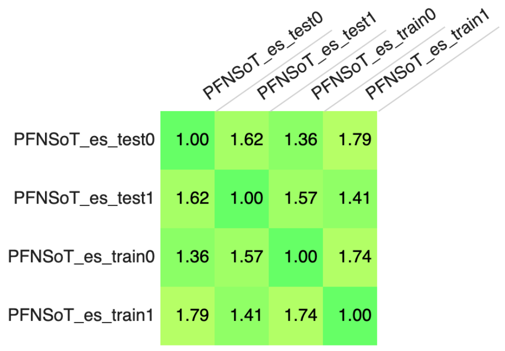
Table 2.2: dataset summary.

| Subcorpus Name | # Tokens | Percentage | Total |
|---|---|---|---|
| es_0 | 832,755 | 53.71% | 1,550,505 |
| es_1 | 717,750 | 46.29% | |
| en_0 | 669,519 | 50.57% | 1,323,982 |
| en_1 | 654,463 | 49.43% | |
| es_train_0 | 500,003 | 54.04% | 925,152 |
| es_train_1 | 425,149 | 45.96% | |
| en_train_0 | 402,788 | 50.92% | 791,024 |
| en_train_1 | 388,236 | 49.08% | |
| es_test_0 | 332,752 | 53.21% | 625,353 |
| es_test_1 | 292,601 | 46.79% | |
| en_test_0 | 266,731 | 50.04% | 532,958 |
| en_test_1 | 266,227 | 49.96% | |

the reference measure I assumed, this also might support the idea that FNS authors will be more difficult to identify than nFNS authors (in contrast, train_1 and test_0 similarity measure is 1.79, which is bigger than the reference measure, 1.74).

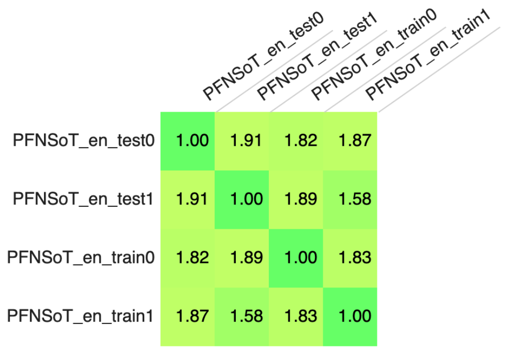
**English Corpus Matrix.** In this matrix, the reference measure given by the difference between train_0 and train_1 is 1.83. While the difference between train_1 and test_1 is below this value (i.e., $1.58 < 1.83$)—although with a smaller gap than the same difference in the Spanish dataset (Spanish: 0.33, English: 0.25)—the similarity measure between train_0 and test_0 differs from the reference measure by just 0.01—in the Spanish dataset is 0.38. This might suggest that systems may have more troubles in identifying nFNS. However, if I look at the difference between train_0 and test_1 and train_1 and test_0, I have similarity measures of 1.89 and 1.87, respectively, which are both slightly higher than the reference measure.

**Comparing what emerged** from these matrices and the error analysis carried out in [216], I noticed that my hypotheses are consistent with the aggregated task participant results. In the Spanish corpus, according to their confusion matrix, nFNS were predicted correctly 80% of the time, while FNS only 65% of the time, confirming *de facto* that FNS were harder to identify than nFNS in this corpus as indicated in the matrix (Figure 2.2a). In the English corpus, they reported a higher confusion from nFNS towards FNS, with nFNS correctly predicted 64% of the time and FNS 70%, confirming again what emerged from the matrix in Figure 2.2b. In addition, the fact that systems performed better on the Spanish corpus could be explained by a similarity measure nearer to 1 (i.e., indicating a higher similarity between the training set of that class and the correspondent test set) than that of the English corpus. These matrices obtained comparing corpora on Sketch Engine, then, might be useful to predict system errors in various corpora. However, looking only



| | PFNSoT_es_test0 | PFNSoT_es_test1 | PFNSoT_es_train0 | PFNSoT_es_train1 |
|---|---|---|---|---|
| PFNSoT_es_test0 | 1.00 | 1.62 | 1.36 | 1.79 |
| PFNSoT_es_test1 | 1.62 | 1.00 | 1.57 | 1.41 |
| PFNSoT_es_train0 | 1.36 | 1.57 | 1.00 | 1.74 |
| PFNSoT_es_train1 | 1.79 | 1.41 | 1.74 | 1.00 |

(**a**)



| | PFNSoT_en_test0 | PFNSoT_en_test1 | PFNSoT_en_train0 | PFNSoT_en_train1 |
|---|---|---|---|---|
| PFNSoT_en_test0 | 1.00 | 1.91 | 1.82 | 1.87 |
| PFNSoT_en_test1 | 1.91 | 1.00 | 1.89 | 1.58 |
| PFNSoT_en_train0 | 1.82 | 1.89 | 1.00 | 1.83 |
| PFNSoT_en_train1 | 1.87 | 1.58 | 1.83 | 1.00 |

(**b**)

Figure 2.2: Comparing English and Spanish corpora: confusion matrices obtained with the chi-square test. The value 1.00 indicates identity between the compared subcorpora. The greater the value, the more different the subcorpora. (**a**) Spanish DS. (**b**) English DS.

at these matrices, it is not possible to state *what* differs between the corpora. Then, I used other Sketch Engine facilities to gain insight into what actually differs between them.

### Keywords

Despite the term *keyword* is widely used outside corpus linguistics, in this field it is used to quantitatively highlight trends in the corpora. Specifically, through keyword analysis, it is possible to retrieve tokens that are statistically characteristic of a (sub)corpus when comparing it with another (sub)corpus (see [70] for a comprehensive exposition of the subject). For both language corpora, I used Keywords to identify what distinguishes the two classes. To do so, I used once FNS as focus corpus and nFNS data as reference corpus, and *vice versa*. In this way, I pointed out focus corpus *keywords* as compared to the reference corpus. Keywords in Sketch Engine are sorted according to their Keyness score, which is calculated as shown in Equation 2.3. In the expression, $fpm_{focus}$ stands for normalized per million frequency of the word in the focus corpus, $fpm_{ref}$ stands for normalized per million frequency of the word in the reference corpus, and N indicates the simplemath parameter, which is used to handle words that only occur in the focus corpus and not in the reference corpus (avoiding the problem of dividing by zero), and to decide whether to give importance to more frequent words or to less frequent words. In fact, different values of simplemath can be used to sort the keywords in the list differently. Generally, higher values of simplemath rank higher more common words; lower values of simplemath rank higher more rare words [137]. I decided to focus on core-vocabulary words, neither so rare nor so common, setting the simplemath parameter to 100. In Table 2.3 I report the first 50 keywords of both corpora.

$$Keynessscore = \frac{fpm_{focus} + N}{fpm_{ref} + N} \tag{2.7}$$

**Spanish Corpus Keywords.** Focusing on the authors labelled as nFNS (corpus 0 as focus) and FNS (corpus 1 as focus), I extract keywords which are used differently by the two groups of users (it is possible also that some tokens do not occur in both subcorpora). Based on these keywords and inspecting the linguistic context (i.e., co-text) in which they occur (using the Sketch Engine Concordance facility), I observed that nFNS (corpus 0) share information about technology (4, 9, 10, 15, 17, 19 'mobile' but also referred only to mobile phones, 29 'screen', 35 'users'), FN (14), toponyms (13, 18, 24–43), politics (20, 32), warnings (8, 11). Conversely, FNS (corpus 1) share information about mostly Latin American artists, music and related (5 'premiere', 8–4, 9, 11–13, 15, 17, 29–39, 41, 45, 46, 47, 48–20, 50), videos (2, 3, 10, 19), shocking or last minute news (5, 7–6, 18–22, 35–28, 37), and also galvanize users to get involved (1 'join us', 14 'download', 23, 31 '2ps-forget', 34 'share it').

In addition, it is worth noting the way in which the two groups use capitalization. While focusing on nFNS, keywords are well-written and capitalization is used in a standard manner (with some exceptions specific to the medium of communication, i.e., Twitter), when I look at FNS keywords, I notice misspellings (missing accents in 1, 4, 7, 18, 35), Latin American spelling (2, 3) and much more capitalized words.

**English Corpus Keywords.** Based on the keywords reported in Table 2.3, and looking at their co-text, I observed that nFNS talk about TV shows and related (2, 3, 4, 7, 11, 28, 29, 36, 43), fashion and related (12, 17, 20, 24, 25, 27), and invite to action (6, 18, 31). FNS, conversely, write about politics (2, 3, 4, 5, 6, 7, 13, 21, 23, 24, 30, 37, 40), famous people and gossip (1, 9, 12, 14, 17, 27, 28, 31, 35, 39), entertainment (19-28, 29), but also warnings about FN (8, 11, 15).

Differently from what emerged from keyword analysis in the Spanish corpus, in the English corpus it is not predictable to which class the first 50 keywords belong. In addition, tweets about FN alerts should not be in FNS data.

### Word Sketch Difference

One of the original features of Sketch Engine is the possibility of outlining the behavior of a word in a corpus using the Word Sketch facility. With its extension, called Word Sketch Difference, it is possible to compare two words observing differences in use or to compare how the same word is used in two different corpora. I used Word Sketch Difference to see how the same word is used by the two groups (i.e., FNS and nFNS) in the two corpora (i.e., English and Spanish datasets). I looked at the modifiers of the word *accident*, *accidente* in Spanish, because it is a word occurring in the two corpora and in the two classes, and because I expected a different use by the two groups which should not be due just to frequency. In Table 2.3 I report all the modifiers associated to *accidente* and *accident*, taken as lemma. In Figure 2.3, I show the distribution of their modifiers in the Spanish

Table 2.3: Spanish and English corpora—Keywords.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Spanish corpus first 50 keywords of nFNS – corpus 0 as focus and corpus 1 as reference** | | | | | | | | | |
| 1 | T | 11 | PRECAUCIÓN | 21 | qué | 31 | seguridad | 41 | información |
| 2 | HASHTAG | 12 | tuit | 22 | to | 32 | PodemosCMadrid | 42 | esa |
| 3 | Buenos | 13 | Albacete | 23 | added | 33 | hemos | 43 | Mancha |
| 4 | Android | 14 | bulos | 24 | Castilla-La | 34 | han | 44 | sociales |
| 5 | h | 15 | Google | 25 | Pues | 35 | usuarios | 45 | Os |
| 6 | he | 16 | artículo | 26 | sí | 36 | servicio | 46 | cómo |
| 7 | sentido | 17 | Xiaomi | 27 | Albedo | 37 | RT | 47 | Nuevos |
| 8 | RECOMENDACIONES | 18 | León | 28 | algo | 38 | datos | 48 | pruebas |
| 9 | Samsung | 19 | móvil | 29 | pantalla | 39 | os | 49 | Gracias |
| 10 | Galaxy | 20 | Cs_Madrid | 30 | disponible | 40 | playlist | 50 | creo |
| **Spanish corpus first 50 keywords of FNS – corpus 1 as focus and corpus 0 as reference** | | | | | | | | | |
| 1 | Unete | 11 | Lapiz | 21 | Dominicana | 31 | OLVIDES | 41 | Concierto |
| 2 | VIDEO | 12 | Vida | 22 | Fuertes | 32 | Joven | 42 | Acaba |
| 3 | Video | 13 | Conciente | 23 | Follow | 33 | Años | 43 | Muere |
| 4 | Clasico | 14 | DESCARGAR | 24 | DE | 34 | COMPÁRTELO | 44 | Hombre |
| 5 | ESTRENO | 15 | Mozart | 25 | Su | 35 | IMAGENES | 45 | Secreto |
| 6 | MINUTO | 16 | De | 26 | Descargar | 36 | Le | 46 | ft |
| 7 | ULTIMO | 17 | Ft | 27 | añadido | 37 | IMPACTANTE | 47 | Preview |
| 8 | Mayor | 18 | Imagenes | 28 | FUERTES | 38 | Accidente | 48 | lista |
| 9 | Alfa | 19 | Official | 29 | Don | 39 | Miguelo | 49 | Republica |
| 10 | Oficial | 20 | reproducción | 30 | Del | 40 | Remedios | 50 | Omega |
| **English corpus first 50 keywords of nFNS – corpus 0 as focus and corpus 1 as reference** | | | | | | | | | |
| 1 | Via | 11 | Synopsis | 21 | Tie | 31 | Check | 41 | isabelle |
| 2 | Promo | 12 | Styles | 22 | qua | 32 | Academy | 42 | AAPL |
| 3 | Review | 13 | Lane | 23 | Bayelsa | 33 | Ankara | 43 | fashion |
| 4 | Episode | 14 | GQMagazine | 24 | du | 34 | rabolas | 44 | Date |
| 5 | PHOTOS | 15 | Mariska | 25 | Robe | 35 | PhD | 45 | esme |
| 6 | Read | 16 | Hargitay | 26 | NYFA | 36 | Spoilers | 46 | isla |
| 7 | Actor | 17 | Nigerian | 27 | Tendance | 37 | DE | 47 | Marketing |
| 8 | TrackBot | 18 | READ | 28 | Supernatural | 38 | story | 48 | Link |
| 9 | RCN | 19 | br | 29 | Film | 39 | Draw | 49 | prinny |
| 10 | AU | 20 | beauty | 30 | Bilson | 40 | University | 50 | your |
| **English corpus first 50 keywords of FNS – corpus 1 as focus and corpus 0 as reference** | | | | | | | | | |
| 1 | Jordyn | 11 | ALERT | 21 | Schiff | 31 | tai | 41 | Price |
| 2 | realDonaldTrump | 12 | Grande | 22 | InStyle | 32 | Him | 42 | Says |
| 3 | Trump | 13 | Biden | 23 | Democrats | 33 | Her | 43 | post |
| 4 | Donald | 14 | Meghan | 24 | Trump's | 34 | Twitter | 44 | About |
| 5 | Hillary | 15 | NEWS | 25 | His | 35 | Markle | 45 | rally |
| 6 | Obama | 16 | published | 26 | After | 36 | Jonas | 46 | BUY |
| 7 | Clinton | 17 | Ariana | 27 | Reveals | 37 | border | 47 | Bernie |
| 8 | FAKE | 18 | Webtalk | 28 | Snoop | 38 | Khloe | 48 | Tristan |
| 9 | Woods | 19 | Viral | 29 | Thrones | 39 | Scandal | 49 | tweet |
| 10 | RelNews | 20 | added | 30 | Border | 40 | Pelosi | 50 | FBI |

Table 2.4: Modifiers of *ACCIDENTE* and *ACCIDENT* in the corpora.

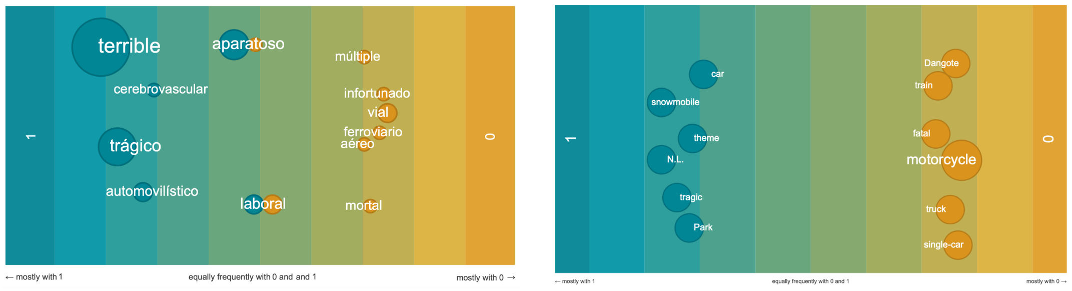
| **Spanish Corpus** | | | **English Corpus** | | |
|---|---|---|---|---|---|
| **Modifiers** | **nFNS** | **FNS** | **Modifiers** | **nFNS** | **FNS** |
| vial | 2 | 0 | single-car | 1 | 0 |
| infortunado | 1 | 0 | Dangote | 1 | 0 |
| ferroviario | 1 | 0 | motorcycle | 2 | 0 |
| mortal | 1 | 0 | truck | 1 | 0 |
| aéreo | 1 | 0 | train | 1 | 0 |
| múltiple | 1 | 0 | fatal | 1 | 0 |
| grave | 1 | 0 | car | 0 | 1 |
| laboral | 2 | 2 | theme | 0 | 1 |
| aparatoso | 1 | 5 | Park | 0 | 1 |
| propio | 0 | 2 | tragic | 0 | 1 |
| cerebrovascular | 0 | 1 | snowmobile | 0 | 1 |
| automovilístico | 0 | 2 | N.L. | 0 | 1 |
| trágico | 0 | 8 | | | |
| terrible | 0 | 19 | | | |

(Figure 2.3a) and English (Figure 2.3b) corpora. In both figures, on the left, the image shows the modifiers which are mostly associated with the selected lemma in FNS tweets; on the right, those associated to the lemma in nFNS tweets; in the middle, those employed by both groups (empty in the English corpus). The bigger the circle, the higher the frequency. In the Spanish corpus, even though in FNS tweets *accidente* occurs more, it is associated mostly with two connotative modifiers (*terrible* and *trágico*[24]). It is interesting to notice a correlation between the modifiers of *accident* in the English corpus with those used in the Spanish one: the use of *tragic* (in Spanish *trágico*) occurring in FNS subcorpus, while *fatal* (in Spanish *mortal*), and vehicles defining the type of accident, occurring in nFNS subcorpus. The presence of these modifiers might indicate that more subjective language is used in FNS data—as *trágico*, *terrible* and *tragic* suggest—while, in nFNS, the news about the accident seems to be reported in a more objective way.

## 2.4 Data augmentation using backtranslation

Social media's ascent, which nowadays dominates the information and entertainment arena all around the world, has revolutionized the way people communicate online [126, 269]. However,

[24]In English: *tragic*.



(**a**) (**b**)

Figure 2.3: Visualization of modifiers of *accidente* and *accident* in the Spanish and English corpora, respectively. (**a**) Spanish corpus. (**b**) English corpus.

it is possible that the latent information in this form of communication is not always explicit in the text, which could hinder the performance of NLP classification models. Data Augmentation (DA) is a technique that can generate an alternative representation of the input and eventually improve model performance. Therefore, uncovering this latent information could lead to better results in author profiling tasks [184]. In this section, I integrate and explore the concept of backtranslation [41, 261, 260, 177] to propose a novel module, to highlight and uncover latent information available in an author's text to improve TC performance.

Studies presented in [212, 255, 158] have shown that backtranslation can be employed as a powerful tool for expanding samples in NLP-related tasks. *Round-trip* (or *Back-and-forth* or simply *back*) translation entails converting spoken or written samples from one language into another and then back again. Moreover, to increase the size of a dataset for machine learning and NLP tasks, DA is a widely utilized approach [103]. This method has been shown to be particularly effective in leveraging the semantic differences between languages and improving the representation of the input [26, 35]. In this study, I specifically focus on a novel strategy of DA. In fact, thanks to the backtranslation module in the framework, I am able to augment each sample while maintaining the same number of dataset samples.

In the proposed setting, each sample is a user's corpus of texts (a Twitter feed), and I hypothesize that semantically enriching the user's text corpus using the proposed modules can improve performance. By augmenting each sample with one or multiple translations, I aim to increase the diversity and informativeness of the data to improve the representation of the input, ultimately leading to better classification performance of different NLP models. In this section, focusing on author profiling tasks, I investigate the effectiveness of backtranslation for expanding samples using English as the original source language and Italian, German, Japanese, and Turkish as the target languages. In one of my previous work [184] I investigated only Italian as a target language and only on a single dataset. The domain was related only to irony and stereotype detection, and there were highlighted promising performance compared to the not augmented version of the framework

[184]. German was adopted as one of the backtranslation languages by the winner of the *Toxic Comment Classification Challenge*[25] while Japanese and Turkish were chosen for their belonging to different linguistic families. The proposed framework is evaluated through a three-stage empirical experiment. First, a baseline of author profiling models is established using datasets without the augmentation modules. The second stage involves generating augmented data using backtranslation from English to target languages with one or multiple augmentations and then back to English. The backtranslated sample is then concatenated to the original one. In the final stage, a machine learning model is trained using the enriched data, and its performance is compared with and without the backtranslation module. I evaluate the framework on three different author profiling datasets (regarding, namely, fake news, hate speech, and irony and stereotypes spreaders). The results outperform the not augmented baseline, showing that the expansion of samples with multiple languages using backtranslation leads to improved performances in author profiling tasks. All the code used for the experiments in this section is available on GitHub[26].

## 2.4.1 The proposed framework

The main components of my proposed framework are three, and they are shown in Figure 2.4 and discussed in this section. It is worth mentioning that the original input sample passes through the same framework during both the training and test phases. While each component is further discussed in the following subsection, here I introduce all the steps performed as shown in the Figure. The input sample is provided as input to the backtranslation module. The backtranslation can be performed using one or more target languages. Then the backtranslated sample is merged with the original one using the expansion module. Finally, the newly expanded sample is provided to the classifier, which provides the final prediction. As already stated, each input sample passes through the pipeline of the framework for both the training and the inference stages.

#### Backtranslation module

The proposed augmentation module has been designed to enhance and eventually highlight content relevant to the classification task. Text data is translated into a different language, and then it is translated back into the original language as part of the backtranslation augmentation. Instead, than necessarily retaining the original context and meaning, this technique creates new textual data with distinct phrases from the original text. To perform the backtranslation in this study, I used the Google Translate API[27].

The augmentation module is also composed of several subcomponents to pre-process each sample (i.e., all the authors-tag and open-close document tags have been removed). This ensures that any

[25] `https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge`
[26] `https://github.com/marco-siino/DA-BT/tree/main/code`
[27] `https://pypi.org/project/googletrans/`

irrelevant or noisy text is removed from the sample. Next, the sample is translated using the translator, which converts the sample into a different language. The backtranslation process is then carried out, which involves translating the text back to its original language (in this case English), with the aim of enriching the semantic content of the text. It is worth noting that the backtranslation could be performed in more than one language. As shown in Figure 2.4, a sample can be eventually backtranslated using different target languages in parallel. In this case, all the backtranslated versions of the sample are provided to the following expansion module.

#### Expansion module

Inside the expansion module, the backstranslated samples are concatenated with the original one, and the augmented sample is generated. Again, it is worth repeating that also this process applies in both the training and test phase. While in previous works as [26] only a single backtranslation (i.e., with just a single target language) is used, in the proposed framework I allow several parallel backtranslation layers to perform the translation toward one or more target languages. In this case, the expansion module merges the text from the original sample with all the backtranslated versions. Considering the case where four target languages are used after the expansion module, the length of the expanded sample is around five times more than the original one.

#### Classifier

After the expansion module, the augmented sample is used to train a classifier and also to test its performance. Several SOTA classifiers can be employed in the framework. To evaluate and assess the performance of the two previous modules, I employ four different classifiers. They are, namely: **RoBERTa**, **GPT-2**, an **SVM** and a **CNN**. The results are reported in Section 2.4.3, and a comparison is made between training on the original and on the augmented datasets in each of the four selected languages, and using all of them. As the datasets are balanced between classes' sample sizes, accuracy was chosen as the evaluation metric. As also discussed in [256, 259, 307], a CNN-based architecture was the top-performing model over the three different datasets.

### 2.4.2 Experimental evaluation

#### Backtranslation languages

The proposed framework's performance is assessed considering different languages chosen accordingly to previous works available in the literature. In fact, this study is an extension of my previous work presented in [184]. In that work, I used only Italian as a target language for backtranslation and I did not consider other languages. Here I evaluate also other languages and more datasets to further investigate the performance of the framework and to conduct a qualitative analysis. Here I also use

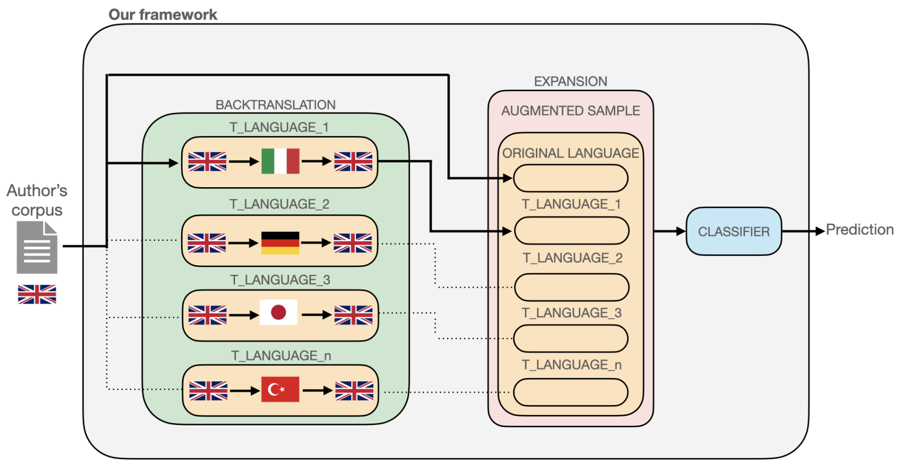


Figure 2.4: The proposed framework.

Italian as the target language, but I also used it in parallel with other languages. German is the second language I use for this study. This is due because German has been employed as a target language for the backtranslation in several other works [80, 109, 27, 26]. Furthermore, I want to investigate the performance using two additional languages with *subject-object-verb* words order and very distinctive characteristics in contrast with the structure of Italian and German. They are, namely, the Turkish and the Japanese. Turkish language characteristics include vowel harmony and significant agglutination. Turkish's usual word order is subject-object-verb. Noun classes or grammatical gender do not exist in Turkish. The usage of honorifics in the language makes a clear contrast between levels of courtesy, social distance, age, and familiarity with the addressee. Out of respect, the second-person pronoun and verb forms that relate to one individual are plural. Japanese is a mora-timed an agglutinative language with pure vowels, a phonemic vowel and consonant system, and a pitch-accent that has lexical significance. Normal sentence structure is topic-comment and subject-object-verb, with particles designating the grammatical function of words is the typical word order. The use of sentence-final particles can create inquiries or add emotive or dramatic emphasis. There is no article, no grammatical gender, and no number for nouns. Verbs are conjugated, but not for person, but rather for tense and voice. Adjectives in Japanese can be conjugated as well. Japanese has a sophisticated honorific system that uses verb forms and vocabulary to denote the speaker, listener, and other people's relative position.

## Classifiers

To test the effectiveness of backtranslation, together with the top-performing CNN, I tested other models, belonging to the set of deterministic and pre-trained models respectively. A brief discussion of all the models and their configuration is provided in the rest of this section.

- **CNN** or Convolutional Neural Networks classify data using a single convolutional layer, a max-pooling layer, and a linear layer. The architecture is the same as discussed in [259]. A single 1D-convolution layer is used in this model. There are sixty-four filters of size thirty-six in this layer. The layer then applies convolution to 36-ngram windows with a stride value of 1, which shifts the convolutional filter by one word embedding tensor for each convolution. The activation function used on the output is ReLu and no padding is used. The final output of the linear layer that follows the global average pooling is a single float value. A zero-threshold value determines the samples' label to compute the accuracy of the model.
- **RoBERTa** uses a bidirectional encoder to produce contextualized word embeddings and belongs to the class of pre-trained Transformer models. The fine-tuning of the model is performed for the task, which in this case is author profiling. In this model, presented in [173], authors conducted a replication study on BERT pre-training and achieved better performance by making modifications to the pre-train phase of a BERT model. These modifications included training the model longer with larger batches, removing the next sentence prediction objective, training on longer sequences, and dynamically changing the masking pattern applied to the training data.
- **GPT-2**, or Generative Pre-trained Transformer 2 was developed by OpenAI [237] and is an open-source large language model. GPT-2 can provide replies to inquiries, translating text, summarizing sections, and producing text. Since it is a general-purpose learner, it was not specifically taught any of these tasks, and its ability to complete them is an extension of its general ability to accurately synthesize the next item in any given sequence.
- **SVM**, or Support Vector Machine, is a linear classifier that aims to find the best hyperplane to separate different classes in a high-dimensional space. Based on [52], I tested the sklearn SVC implementation[28]. I used a linear kernel type with a value of 0.5 as a regularization parameter.

Using different models to test the augmentation modules also allowed to assess the usability of the framework. Different models have different strengths and weaknesses, and they may perform differently on different datasets or tasks. By testing the augmentation module on multiple models, I can better understand its effectiveness and limitations in different scenarios. Moreover, the usage of multiple runs for each model can help to reduce the impact of random initialization and provide more robust evaluation results.

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

**Experimental setup**

On a Tesla T4 from Google Cloud and an NVIDIA GeForce RTX 2080 GPU on a local system, I ran the experiments using TensorFlow. I used the *Simple Transformers* [29] library to evaluate the large language models. The batch size for all models was equal to 1. Each Transformer used came from the library of Transformers provided in [299]. I used early stopping to fine-tune RoBERTa and GPT-2 for 10 epochs in accordance with the test set accuracy. According to the reference study, the best accuracy was typically attained prior to the tenth epoch of fine-tuning. For 20 epochs, the CNN was trained from scratch. Again, there were no advantages in training for more than 20 epochs in this study; on the test set, the top accuracies were consistently attained before epoch 20. To assess the effectiveness of each model, I adhered to the protocol outlined in [173]. I therefore carried out five random weight initializations. There is no need for multiple runs of the SVM because it uses the implementation covered in the previous section, and it is deterministic. References to the initial implementations of each model and the experimental setups for each architecture are already supplied. On request, all the datasets I used can be released.

### 2.4.3 Results and discussion

I tested the performance of the augmentation module on three datasets: FNS, HSS and ISS. For each augmentation combination, I trained all four models, evaluating their performance on the test set. The results for each combination of augmentation and model are presented in Tables 2.6 and 2.5.

Looking at the results, the augmentation module ensures that there is at least an augmentation strategy with a performance that is at least as good as the one of the original (RQ1) (so not augmented) across **all datasets and models**. It is worth noting that *augmented-it* stands for augmentation using Italian, and *augmented-mix* stands for augmentation using all the four languages. Overall, **HSS** and **FNS** are the datasets where most of the combinations perform better than without the augmentation module.

Roberta, CNN, and GPT-2 present significant p-values for the Italian and German augmentation when trained with the FNS dataset. I compute this by performing an unpaired *one-tailed t-test,* and in the tables the average of the runs is reported too. With RoBERTa, the augmentations with German, Turkish, and all languages mix always perform better than without the augmentation modules. CNN architecture performs significantly better on the HSS dataset while on the ISS one, the performance of the training with original data can be equalled but not surpassed. Surprisingly, CNN with all languages as expansion cannot outperform other augmentation strategies.

The SVM model seeks to maximize the distance between the decision boundary (hyperplane) and the closest data points from each class. In the experiments for HSS, all the augmentation performs

[29]`https://simpleTransformers.ai/about/`

Table 2.5: RoBERTa, CNN, and GPT-2 accuracy for each dataset and augmentation. In the first column, the best results are reported, while the second one reports the average of the 5 runs. The p-value column reports the output of the one-tailed t-test to check if there is a statistically significant difference between the not augmented accuracy and the alternatives' accuracies over the 5 runs. Bold values represent the best value of accuracy in each dataset.

| | **RoBERTa** | | | **CNN** | | | **GPT-2** | | |
|---|---|---|---|---|---|---|---|---|---|
| **FNS** | **Best Run** | **Average** | **p-vals** | ***Best Run*** | **Average** | **p-value** | ***Best Run*** | **Average** | **p-value** |
| not-augmented | 0,7100 | 0,6890 | 0,00 | 0,7300 | 0,7200 | 0,00 | **0,6300** | 0,6300 | 0,0000 |
| augmented-it | 0,7150 | 0,7080 | 0,0296 | 0,7200 | 0,7140 | 0,1140 | 0,6200 | 0,6120 | 0,0004 |
| augmented-de | **0,7200** | 0,6930 | 0,3549 | 0,7250 | 0,7160 | 0,1914 | 0,6050 | 0,6050 | 0,0000 |
| augmented-ja | 0,7100 | 0,6980 | 0,1548 | 0,7200 | 0,7170 | 0,2297 | 0,6050 | 0,6050 | 0,0000 |
| augmented-tr | 0,7100 | 0,6940 | 0,2906 | **0,7350** | 0,7190 | 0,4420 | **0,6300** | 0,6140 | 0,0081 |
| augmented-mix | **0,7200** | 0,6970 | 0,2207 | 0,7200 | 0,7140 | 0,1366 | **0,6300** | 0,6140 | 0,0081 |
| **HSS** | ***Best Run*** | **Average** | **p-vals** | ***Best Run*** | **Average** | **p-value** | ***Best Run*** | **Average** | **p-value** |
| not-augmented | 0,5900 | 0,5660 | 0,00 | 0,6500 | 0,6280 | 0,00 | 0,6500 | 0,6500 | 0,0000 |
| augmented-it | **0,6400** | 0,5700 | 0,4295 | 0,6800 | 0,6440 | 0,1312 | **0,6600** | 0,6480 | 0,3520 |
| augmented-de | 0,5900 | 0,5700 | 0,3912 | 0,6500 | 0,6400 | 0,1134 | 0,6500 | 0,6500 | 0,0889 |
| augmented-ja | 0,6200 | **0,5760** | 0,3386 | **0,7200** | **0,7000** | 0,0000 | 0,6400 | 0,6400 | 0,0000 |
| augmented-tr | 0,6200 | 0,5660 | 0,5000 | 0,6700 | 0,6300 | 0,4383 | 0,6000 | 0,5960 | 0,0001 |
| augmented-mix | 0,6100 | 0,5680 | 0,4668 | 0,6700 | 0,6560 | 0,0071 | 0,6100 | 0,6080 | 0,0000 |
| **ISS** | ***Best Run*** | **Average** | **p-vals** | ***Best Run*** | **Average** | **p-value** | ***Best Run*** | **Average** | **p-value** |
| not-augmented | 0,8222 | 0,7967 | 0,00 | **0,9611** | 0,9611 | 0,00 | **0,9400** | 0,9120 | 0,0000 |
| augmented-it | 0,8222 | 0,7900 | 0,3586 | **0,9611** | 0,9578 | 0,0352 | 0,7660 | 0,7660 | 0,0000 |
| augmented-de | **0,8333** | 0,7944 | 0,4412 | **0,9611** | 0,9578 | 0,1043 | 0,7660 | 0,7660 | 0,0000 |
| augmented-ja | **0,8333** | 0,8000 | 0,4179 | 0,9556 | 0,9534 | 0,0024 | 0,8700 | 0,8300 | 0,0001 |
| augmented-tr | 0,8111 | 0,8011 | 0,3515 | 0,9611 | 0,9545 | 0,0895 | 0,7660 | 0,7660 | 0,0000 |
| augmented-mix | **0,8333** | 0,8022 | 0,3470 | 0,9500 | 0,9500 | 0,0022 | 0,9222 | 0,9222 | 0,1094 |

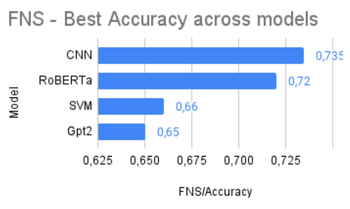
better than the original one; the all language (*mix*) augmentation is better both for HSS and FNS.

Table 2.6: The table reports the values of maximum accuracy reached by the SVM for each dataset and augmentation. Bold means maximum value by columns.

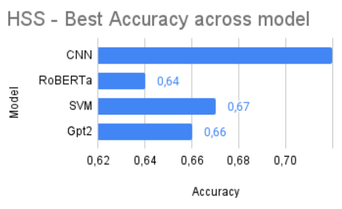
| | **SVM Accuracy** | | |
|---|---|---|---|
| | **FNS** | **HSS** | **ISS** |
| not-augmented | 0.6300 | 0.5900 | 0.9278 |
| augmented-it | 0.6450 | 0.6100 | 0.9222 |
| augmented-de | **0.6600** | 0.6000 | 0.9167 |
| augmented-ja | 0.6300 | 0.6400 | 0.9278 |
| augmented-tr | 0.6200 | 0.6200 | **0.9333** |
| augmented-mix | 0.6350 | **0.6700** | 0.9167 |

The CNN is the model that overall reaches the most accurate results, especially on the HSS dataset, and this can be seen in Figure 2.6. Figure 2.5 confirms that CNN is the best-performing model. CNN outperforms RoBERTa and SVM, which is the second-best on 2 out of 3 datasets. The official results for English[30], indicate that for HSS the best accuracy is equal to 0.7300, 0.005 less than the best run [259], the winner of FNS challenge reaches an accuracy of 0.7500, and finally the best accuracy for ISS is equal 0.9944.

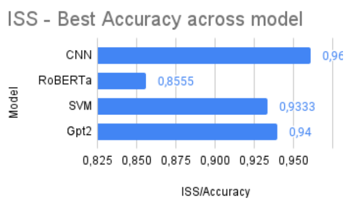
[30]`https://pan.webis.de/`



(a) FNS



(b) HSS



(c) ISS

Figure 2.5: Best accuracies for each model across datasets.

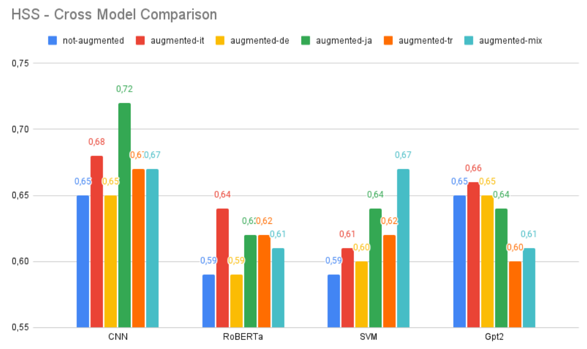


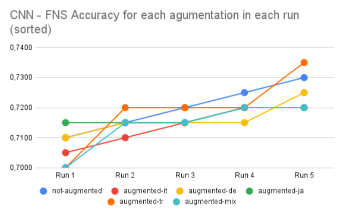
Figure 2.6: Cross-model and cross-augmentation results for the HSS dataset.

Figures 2.7, 2.8, 2.9 report the (sorted) accuracy (on the validation data, $y - axis$) for each of the 5 runs ($x - axis$) of each model trained with each dataset.

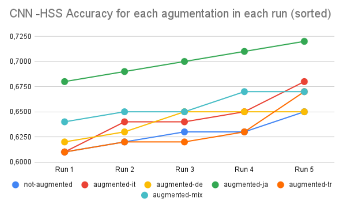
The Japanese augmentation in the HSS dataset always outperforms the languages or combinations used for augmentation. On the other hand, in the ISS dataset, CNN reaches a higher level of accuracy (around +10%) w.r.t. RoBERTa. It is worth noting that GPT-2 is not able to perform as well as the other tested models. In this context I use it as a binary classifier on the three datasets and, eventually, this could motivate the lack of performance. It is interesting that GPT-2 on FNS and on ISS is able to reach the highest accuracy without the proposed framework. However, the results are always lower, for all the three datasets, when compared to a CNN. The main reason could be due to the specificity of the task. When performing author profiling, it is not a single and small piece of text to be classified but a feed of texts from the same author. This observation and the results are consistent with the main findings reported in [256].

## 2.4.4 Qualitative analysis

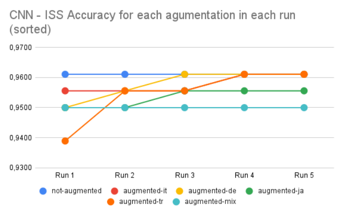
In this section, I perform some qualitative analyses on the samples augmented after the backtranslation by comparing them with the non-augmented versions. First, I compare augmented versions with



(a) CNN-FNS

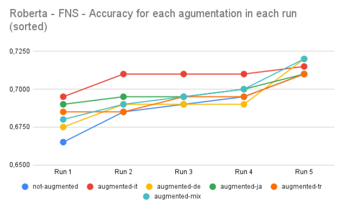


(b) CNN-HSS

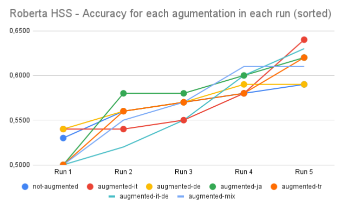


(c) CNN-ISS

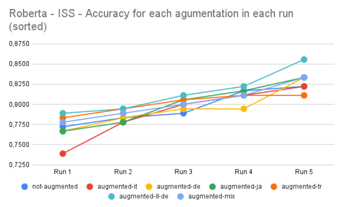
Figure 2.7: CNN accuracies across different datasets and augmentations.



(a) RoBERTa - FNS



(b) RoBERTa - HSS



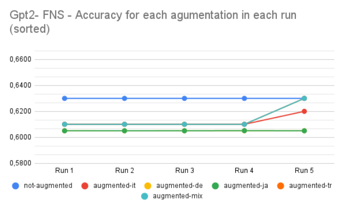
(c) RoBERTa-ISS

Figure 2.8: RoBERTa accuracies across different datasets and augmentations.

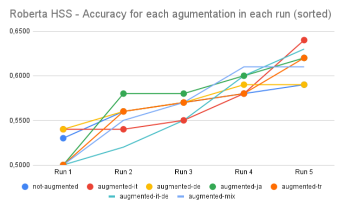
significantly improved performance, and then the augmented ones that have not. Specifically, with respect to the HSS dataset, I qualitatively analyze the augmented version in Japanese which allowed a significant increment in performance using a CNN. Then the qualitative analysis is conducted on ISS where the augmentation has not produced significant increases with any language. Later on, I compare some augmented samples in the case of German with the original versions.

### Japanese on HSS

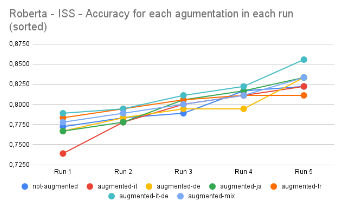
In Table 2.7 are shown some samples from the HSS dataset. The samples are backtranslated using Japanese and highlighting the changes. In the case 1*b* only the word *Queen* is replaced with the word *Mistress*. This is a case of word substitution, where the semantic of the word *Mistress* is more specific and contextualized than the word *Queen*. In fact, the word *Queen* represents a case of polysemy in which the word can refer to both a queen of a kingdom, the popular rock band, a chess piece and, by extension, the concept of Mistress. Thus, a classifier previously trained with other meanings of the word *Queen* may not fully understand the actual meaning. In contrast, the word *Mistress* has a specific meaning about a woman in a position of authority or control, often in sexual contexts. Also in the cases 2 and 3 some words are replaced (*confident* with *believe* and *man-meat* with *human flesh*). But in the case 3 the referent of the discourse is also changed. In the case 3*b*) *torture it* becomes *torture you*. Also in the case 4) a substitution of words could have made the



(a) GPT-2 - FNS



(b) GPT-2 - HSS



(c) GPT-2-ISS

Figure 2.9: GPT-2 accuracies across different datasets and augmentations.

latent semantics clearer for the classifier. In fact, the words *Sophia is on her usual* are replaced with *Sofia shows her usual*. A single word (i.e., shows) replaces "is on her" and this, in the case of a CNN with single-word embedding, allows the expressed concept to be enclosed in a single term. Also in the case 5) an interesting substitution (i.e., *to please* in place of *for the amusement*) makes explicit and shortens a concept on a single verb. Furthermore, the plural *races* are replaced by the singular race. It is interesting to note that 7 words present in 5$b$) were not present in 5$a$). In the case 6$b$) *hitting* replaces the word *beatings*. And also in this case the two concepts are similar but not equals. In the case 7$a$) the plural is replaced with the singular. Therefore, the author's comment loses the generic reference to a set of people and is addressed exclusively to a single subject. Finally, in the case 8), the translator corrects a typing error and, therefore, the word in the augmented sample can be eventually traced back, easily, to an already learned embedding space.

### German on ISS

With regard to the ISS dataset, as shown by the results, the German-augmented and non-augmented performances with CNN are equivalent. As the examples in Table 2.8 show, the translation is almost identical. This produces essentially similar classification performances.

In case 1$b$), even if some words have been replaced, the semantics are essentially the same. Furthermore, in the case of the short form *I'd* it is not even appropriate to speak of substitution as it has only been expanded with *I would*. Also in the second case, although the sentence contains many words, only a few of those present in 2$b$) are not present in 2$a$). *Also* achieves the same meaning as *too*, *Lawyer* has simply been replaced with the first capital letter, and *seem* and *look* are generally used interchangeably. Finally, in the case 3) only four words are changed and in one case, as before, *also* in place of *too* is added.

This great similarity between the augmented and non-augmented versions of the samples is in fact confirmed by the similarity of the results obtained from the models on the ISS dataset.

Table 2.7: Examples of original tweets from HSS backtranslated using Japanese. Changes in the backtranslated samples are highlighted in yellow.

| ORIGINAL | BACKTRANSLATED (JAPANESE) |
|---|---|
| 1a) And the Queen will cage your cock and balls! #URL# #URL# | 1b) And the Mistress puts your cock and balls in a cage! #URL# #URL# |
| 2a) RT #USER#: I'm confident that all men are inferior to Women. | 2b) RT #USER#: I believe that all men are inferior to women. |
| 3a) RT #USER#: Use your man-meat for something meaningful. Let Femocracy Women torture it. Bow &amp; Serve. #URL# | 3b) RT #USER#: Use human flesh for something meaningful. Let Femocracy Women torture you. Bow & serve. #URL# |
| 4a) RT #USER#: Sophia is on her usual fine and sadistic form in the new clip at #URL# #HASHTAG# | 4b) RT #USER#: Sofia shows her usual feisty sadistic look in new clip on #URL# #HASHTAG# |
| 5a) RT #USER#: A day at the races... nude males competing for the amusement of their female owners. #URL# | 5b) RT #USER#: A day in the race... Naked men compete to please their female owners. #URL# |
| 6a) #USER# Ball beatings is one of the most effective methods in order to keep in line the males of the family. | 6b) #USER# Ball-hitting is one of the most effective ways to keep the men in your family in line. |
| 7a) Bitches be in relationships and don't even like they bf | 7b) Bitch is in a relationship and doesn't like it |
| 8a) Tried to give a bih the world but she wanted the streets | 8b) Tried to give the world to a bitch, but she wanted the streets. |

### 2.4.5 Conclusion and future works

In conclusion, for all three datasets examined, the proposed framework improves the performance if compared to a simplified version of the considered architecture without the augmentation modules (i.e., backtranslation and expansion). The technique consists of an augmentation model that makes use of **backtranslation** before expanding each sample by concatenating it with the original data. The findings imply that a user's text corpus can be semantically enriched as a useful method to enhance the performance of an author profiling model. The CNN model fared well with the HSS dataset, whereas the RoBERTa model consistently improved with the inclusion of backtranslation and expansion, despite the fact that each model's performance varied among datasets and augmentation combinations.

Differences in performance between the augmented and original models were also tested for statistical significance with a one-tailed t-test. Still, the p-value is under the threshold of 0.05 only for a few combinations. It is crucial to notice that the low sample size ($N = 5$) can affect the outcome of the test.

I found that enriching samples with their respective backtranslations can lead to performance

Table 2.8: Examples of original tweets from ISS backtranslated using German. Changes in the backtranslated samples are highlighted in yellow.

| ORIGINAL | BACKTRANSLATED (GERMAN) |
|---|---|
| 1a) #USER# #USER# While Pierre's education may not be as elitist as Freeland's, I'd prefer as finance minister someone with his "commerce" education over a Slavic degree. | 1b) #USER# #USER# While Pierre's education may not be as elite as Freeland's, as Treasury Secretary I would prefer someone with his "business" background to a Slavic degree. |
| 2a) #USER# #USER# If #USER# wins #HASHTAG# she should consider "coaching" too. She's articulate but needs to shed the "lawyer" blandness. Can't look too meek when debating or Trudeau and media will eat her for breakfast | 2b) #USER# #USER# If #USER# #HASHTAG# wins, she should also consider "coaching". She's articulate, but needs to drop the "Lawyer" fade. Can't seem too meek when debating or Trudeau and the media will eat her for breakfast |
| 3a) #USER# #USER# #USER# Counter argument: Back in the '70's, Biden was racist too (different times, let's move on). Other accusations later were "hearsay". They say multi-blackface Trudeau isn't racist either. I think individual perception applies here. | 3b) #USER# #USER# #USER# Counter argument: In the 70's Biden was also racist (other times, let's move on). Other allegations later were "hearsay". They say multi-blackface Trudeau isn't racist either. I think individual perception counts here. |

improvements. The greater the diversity of the backtranslated versions, the more likely can be obtained a performance boost.

In addition, thanks to a qualitative analysis, I have found that backtranslation automatically allows the increasing of information content of a text without feature engineering. One of the most important things that emerged is that the backtranslation using Japanese allows a significant increase in performance. This is most likely due to a better explanation of the information that can express hatred on social media after backtranslating the samples.

In future works, it would be interesting to investigate this aspect also on other datasets and not only for author profiling tasks. Furthermore, it could also be of interest to evaluate the impact of other languages used in the backtranslation module, although, as emerged from this study, the inclusion of a larger number of languages does not necessarily lead to an increase in the performance of the classification models employed.

# Chapter 3

# Preprocessing

Tasks related to NLP usually consist of lexical tokenization, preprocessing, probabilistic tokenization and classification stages. The preprocessing step involves operations like lowercasing, stemming, lemmatization, stop words removal and others techniques presented in this chapter. Here, I use the term *preprocessing* to refer to any changes made to the input text after performing lexical tokenization and before proceeding with probabilistic tokenization. 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, lemmatization, casing conversion of characters, expanding abbreviations, correcting misspellings), and enhancing the quantity of semantic information available (e.g., the Part of Speech tagging, the use of strategies to manage negation words). This implies that preprocessing can potentially delete important data (such as deleting stop words when they are pertinent to a researcher's study issue). Furthermore, several errors can be introduced into the task's pipeline. For instance, when semantically distinct words conflate using stemming and changing the outcomes of a classification model. In this chapter, preprocessing involves text transformation before identifying which text units to use as tokens during the probabilistic tokenization stage.

Despite its importance, the text preprocessing stage is often underestimated in several text mining studies found in the literature. However, there is a substantial quantity of noise in unstructured texts available on the internet. In some cases, the amount of noise in a dataset can be so high that it could mislead any machine learning algorithm. The presence of noise could be caused by users who frequently utilize slangs and acronyms, as well as making spelling and grammar mistakes. To emphasize their emotions, users may also abuse punctuation marks. For example, typing multiple exclamation marks instead of a single one. In this context, my definition of noise is related to any useless information for any text-based task to be performed after preprocessing a dataset. As discussed here, an incorrect choice when preprocessing text can lead to a difference of over the 25%

in terms of accuracy in 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 provided as input 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. The crucial role that preprocessing has before and throughout the feature selection process, from my point of view, is of prominent importance and to be widely noted. Unfortunately, past researches have 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 PhD thesis I report and discuss these techniques and subsequently, for the most commonly used, I evaluate the results obtained by their combinations with respect to the model and to the dataset considered. This section aims at improving text preparation stage and resolving inconsistencies in preprocessing advices, to offer guidelines and ideas for future studies. I aim at improving the comprehension of the theoretical and empirical factors that should influence preprocessing choices. Here, I want to investigate how performance is affected by preprocessing choices while making use of deep (pre-trained or not) and non-deep learning models. Eventually, a preprocessing stage cannot only remove noise and/or highlights important features, but can also reduce the time required for training and testing a model. I can finally state that it is important to make an educated and context-dependent choice about which preprocessing method (or combinations of methods) to employ and in what order.

In this chapter I collect, report and discuss the text preprocessing techniques found in the literature and their possible and most recent variants, proposing a uniform nomenclature standard based on acronyms. I also provide the reader with useful information for self-study and in-depth study of the techniques presented here and presenting useful advice on how to operate educated choices to select the preprocessing technique (or combination of techniques) given a specific task, model, and DS. Furthermore, in the last section, I select the three most common techniques used in the literature to evaluate the impact of each of these techniques (alone or in combination) on the classification results of nine SOTA models (pre-trained deep, deep and non-deep) and on real world datasets. Then I discuss how text preprocessing can affect the performance of modern pre-trained architectures based on attention (i.e., Transformers). Finally, I determine if the performance of simple classifiers are comparable to the performance of Transformer-based models when text preprocessing is performed in accordance with the specific model and/or dataset used.

This chapter dedicated to text preprocessing is structured as follows. The gaps in the literature and the related work on the impact of preprocessing techniques are discussed in the next two sections. In Section 3.3 a complete discussion of the preprocessing techniques collected is presented. In the last section, I discuss the procedure for the experimental evaluation and the outcomes of my experiments on three different datasets using the three most common text preprocessing techniques on nine SOTA models.

## 3.1 Gaps in the literature

In this subsection, I 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 the section dedicated to the related work. I conclude this subsection, highlighting the gaps found in the literature.

In [263] authors analyze the preprocessing impacts 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 texts. Involving four conventional classifiers and a neural network in their experiments, the authors in [270] investigate how each preprocessing technique affects the performance of the models, using solely TF-IDF (unigram) to represent words. 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 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 products or movie reviews. In [210] authors analyze twelve different preprocessing techniques on three datasets. The datasets are built from Twitter and focus on hate speech detection. Authors observe the impact of the preprocessing techniques on the classification tasks they support. However, they do not fully explore all the possible combinations 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 next, 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 modern deep learning architectures are missing.
- There is a lack of experimental evaluations on models that can truly achieve valuable SOTA 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 this chapter, I hope to better investigate the matter without neglecting any aspects or point of view reported above.

## 3.2 Literature review

I report in this section the results of some of the most relevant and recent studies employing text preprocessing techniques to evaluate their effect. 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 the Section 3.3.

Recently, the authors in [153] have used 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 the 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 [101] 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, 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 [23] on a Twitter dataset for sentiment classification (i.e., the Stanford Twitter Sentiment Dataset). Authors explore some combinations of the preprocessing techniques proposed. 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 [91], authors employ traditional models like NB, 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 [15] demonstrate that the correctness of classifier predictions 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 traditional models (namely, NB, Decision Tree and SVM). Four traditional classifiers (i.e., NB, Logistic Regression, SVM and Random Forest) are also employed in [119], 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, is enhanced. The Transformers are used in [65] where, before applying TF-IDF, 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 accuracy 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 [318] 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* is mapped into *yee*). While the authors employ BERT as a classification model, they only use the preprocessing strategy discussed above. Also authors in [64] introduce and apply a new preprocessing strategy based on three new steps (i.e., lowering dimensionality, rising sparseness and reducing the 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

Table 3.1: Acronyms for 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 and Pseudonimization | *”@Obama 0x10FFFF 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 be go to shopping”* | *”I am 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”* |

crucial than the classification algorithm itself. Mainly, for obtaining the best performance at the lowest possible cost (trade-off among effectiveness-efficiency).

## 3.3 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 [270], the authors present an evaluation of several text preprocessing techniques on two datasets built to perform sentiment analysis on Twitter. The article was used as the foundation for my work because — as shown in Table 3.2 — it proved *a posteriori* to contain the largest number of techniques available. 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 here. Techniques not discussed in [270] were added as columns to Table 3.2 and also included and discussed. Studies with less than three techniques are not shown in Table 3.2 but, if targeting some specific technique with a novel or deeper point of view, they have been briefly discussed in the Section 3.3. At this point, for each of the study added from time to time to the reference list, the papers cited or citing each work in the Table 3.2 were included, as long as they discussed at least three different preprocessing techniques. Thanks to this approach, I can state that, to the best of my knowledge, the preprocessing techniques most frequently found in the literature have been included in this chapter.

All the preprocessing techniques reported here represent the very first stage for any task related to TC after lexical tokenization. As defined in [130] the tokenization is the task of separating a running text into words. To these words one or more preprocessing techniques can be applied. The next step, after text preprocessing, consists in splitting text into n-grams (i.e., probabilistic tokenization). Before providing the preprocessed text to a model, it is necessary to tokenize it into a

Table 3.2: 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)[6] | | | | | | | | X | | | | X | | X | | |
| Albalawai (2021)[7] | X | | | | X | X | | X | | | X | X | X | | | |
| Alzahrani (2021)[9] | X | | | | X | X | | X | | | | | | | | |
| Anandarajan (2019)[11] | X | | | | X | X | X | X | | X | X | X | | | | |
| Angiani (2016)[12] | X | | | | | | X | X | X | | | X | | | X | |
| Araslanov (2020)[14] | X | | | | | X | X | X | | | X | X | | | | |
| Babanejad (2020)[17] | X | | | | X | X | X | X | X | X | | X | | | X | |
| Bao (2014)[23] | X | | | | | | | | | | X | X | X | | X | |
| Denny (2018)[72] | | | | | X | X | X | X | | | | X | | | | |
| Duong (2021)[78] | | X | | | X | X | X | X | X | X | | | X | X | X | |
| Hacohen (2020)[99] | | | | | X | | X | X | X | | | | X | | | |
| Haddi (2013)[100] | | X | | | | | | X | | | | X | | | X | |
| Hickman (2022)[105] | | X | X | | | | X | X | X | | X | X | | X | X | |
| Jianqiang (2017)[119] | X | X | X | | | X | | X | | | | | X | | | |
| Kadhim (2018)[131] | X | | | | | | | X | | | | X | | | | |
| Kathuria (2021)[132] | X | X | | | X | X | X | X | | X | X | X | X | | X | |
| Koopman (2020)[145] | | | | | X | | X | X | | | X | X | | | | |
| Kowsari (2019)[148] | X | X | | | | | X | X | | | X | X | | | | |
| Kumar (2019)[150] | X | X | | | X | | | X | X | | X | X | | X | | |
| Kunilovskaya (2021)[152] | | | | | X | | X | X | | | X | | X | | | |
| Lison (2017)[168] | | | | | X | | | X | | X | X | | | | | |
| Mohammad (2018)[202] | X | | X | | | | X | X | | | X | X | | | | |
| Naseem (2021)[210] | X | X | X | | X | X | X | X | X | | X | | X | X | | X |
| Petrovic (2019)[228] | | | | | X | | X | X | | | | X | | | | |
| Pradha (2019)[233] | X | | | | X | | X | X | X | | X | X | | | | |
| Rosid (2020)[246] | | | | | | | X | X | | | | X | | | | |
| Singh (2016)[263] | X | X | | | X | | | X | X | | | | | | | |
| Smelyakov (2020)[264] | | | | | | | | X | | | X | X | | | | |
| Symeonidis (2018)[270] | X | X | X | X | X | X | X | X | X | X | X | X | X | | X | |
| Toman (2006)[282] | | | | | | X | X | X | | | X | X | | | | |
| Uysal (2014)[283] | | | | | | | X | X | | | | X | | | | |
| Zong (2021)[317] | X | | | | | | | X | | X | X | X | | | | |

numerical form which can be managed by a computer. In some studies, tokenization (either lexical or probabilistic) is presented as a preprocessing technique. However, I do not include the tokenization among the techniques presented here. The techniques presented in this chapter are characterized by their ability to alter the syntactic and semantic content of a text after the lexical tokenization. At the same time, tokenization (either lexical or probabilistic) is the necessary procedure to fragment a text to be able to supply it to subsequent stages. Nevertheless, considering that tokenization is often referred to as part of the preprocessing, I introduce and discuss tokenization in the rest of this section.

Lexical tokenization is discussed in [102, 192, 290, 1] and usually split text into words. Probabilistic tokenization can split texts into pieces called *tokens*. Although common forms of tokenization are performed at word-level, several sub-word tokenization strategies are discussed in the literature [253, 149, 252]. Regardless the size of the tokenization window used, tokenization generically 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 probabilistic 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 into word) or computer science (where it is called probabilistic tokenization and is used to map a token into a number) 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. In the case of languages like Chinese, the terms *word segmentation* apply.

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 a 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 [17], it is worth notice that it is not necessary to perform preprocessing on both the training and the test sets.

Finally, given the methodology reported in the previous section as well as in the rest of this chapter, the histogram in Figure 3.1 displays a list of the preprocessing techniques that have been documented in the literature. The histogram also shows the number of times that the reported techniques have been used in the related work.

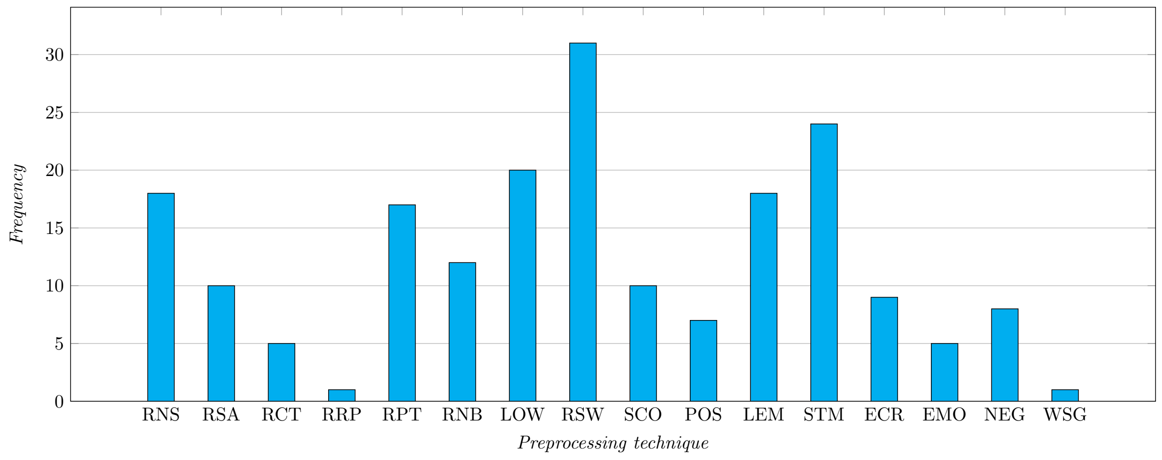


Figure 3.1: Number of times that the techniques discussed in this article are found in related work. In Table 1 are reported the expanded acronyms under the bars. The works related to the Figure are the ones listed in the Table 3.2. Each bar in the Figure actually shows the counts of the X in the table for each column.

### 3.3.1 Replace noise and pseudonimization

The definition of noise varies significantly according to the literature, with regard to removing and/or replacing noise. Usually noise replacement consists in replacing or removing unwanted strings and Unicode characters, which are regarded as crawling by-products, that can add further noise to the data. For this reason, some authors employ regular expressions to eliminate Unicode strings and non-English words. The authors in [17] 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.

Especially in the context of sentiment analysis, another form of noise replacement is *pseudonimization*. 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, depending on the task, can be treated as noise replacing them with specific tags. In the literature are described a number of methods to deal with this additional data supplied by users. In [2], authors replace all the URLs with a tag *U*, and replace user mentions (e.g. @*brucespringsteen*) with the tag *T*. The majority of academics believe that URLs don't reveal anything about the sentiment of a tweet [135, 115, 8, 242]. Other scholars expand URLs from Twitter into full URLs before tokenization [38, 30]. The tweet text is then refined by removing any URLs that match the tokens. In conclusion, no general rules apply in definition and managing of noise. Definition and operations can vary significantly from a study to another.

### 3.3.2 Replace slang and abbreviation

Considering the character count restrictions in social networks (e.g., Twitter), abbreviations, acronyms, informal writing styles, short words and slang are frequently used [277]. These words have to 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 [146] slangs and abbreviations are converted into word meanings that can be comprehended by utilizing conventional text analysis methods. In [270] 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.

### 3.3.3 Replace contraction

Contractions are short-form words that are used by users to reduce the number of characters in a tweet/post [248]. 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* be replaced by *cannot*).

Expanding contractions could or could not be a beneficial preprocessing technique before performing probabilistic 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 not be all that helpful. It is worth mention that, even if the main referenced language of this thesis is the English, some interesting considerations could be made concerning other languages. For example, French has a contraction phenomenon which consists of truncating many words (for example, *manif* for *manifestation*), and Italian often presents articles with an apostrophe (e.g., *L'arte della guerra*, 'The art of war'), which should likewise be managed when focusing with these languages.

### 3.3.4 Remove repeated punctuation

In [270], 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, authors substitute a representative tag in its place. For instance, "multiQuestion-Mark" is used in place of the token "???". This procedure is performed before deleting punctuation. However, in the not pre-trained models evaluated in this PhD thesis, 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 *???*). Of course, these different splitting strategies would lead to different behaviors of a subsequent classifier.

### 3.3.5 Remove punctuation

In written texts, punctuation can be used to express sentiment and emotion [281] (e.g., "*You 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 tasks. For this reason, punctuation removal is often applied in many preprocessing tasks for automated TC. However, punctuation symbols can also denote sentiment. In [20], authors detect punctuation signs like "*!!!*" and replace them with the label "*multiexclamation*". An application where punctuation is removed can also be found in [167]. In the study presented in [259], the authors do not remove punctuation during preprocessing. In fact, they consider as separate entries in the dictionary the words *up* and *up!*. In this way, the word embedding layer, trained from scratch in the study, at the end of the training phase is able to differentiate the meanings of the two entries in the dictionary assigning different word vectors in the embedding space. These behaviors could be, eventually, able to get the intended meaning of the version with the exclamation mark, to invoke someone for moving faster. Removing punctuation from the sentence and replacing it with a single space (i.e., "*You are late Hurry up*"), would result in the change of some latent information, maybe of interest for certain TC tasks (e.g., author profiling as in the study of [259].

### 3.3.6 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 [167, 11]. 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 [72, 259] 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 pronounced it. Such a 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.

### 3.3.7 Lowercasing

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

Lowercasing is discussed in [48] 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, authors in [283] change capital letters from uppercase to lowercase. According to authors, the classification's performance has improved. Lowercasing has been a common method 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 [76]. In the 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[1] instead of comparing the same sound to the one of the 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 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 this work, several datasets and models are tested, so it is discussed the general impact of the technique using modern classifiers on real world cases.

### 3.3.8 Remove stop words

The removal of stop words, according to this study, is the most often employed preprocessing method found in the literature. Stop words are typically frequent terms in a language and are assumed to be the least informative [94] (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

[1]... and in this case, maybe, you should look for a new drummer.

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, there are usually more than 400 stop words in the English language [77, 87].

The first study considering stop words is conducted in [180]. 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 [249], 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. Authors compare results between static stop word removal techniques (e.g., based on pre-compiled lists) versus dynamic stop word removal techniques [183] (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 those uncommon words appearing rarely in the dataset. Researchers have found that a word's relevance can be inferred from its frequency in a data collection. This discovery led to the exploration of various well-liked stop word removal techniques in the literature. While some approaches consider both the top and the 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). To conclude this section, four different stop word removal techniques are now described.

- *The traditional approach.* The traditional approach [285] 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 [60, 183]. 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 [62]. A lower mutual information means that the word has a weak ability for helping in discrimination, hence it needs to be dropped.
- *Random sampling of data chunks.* It was initially suggested in [175] 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 [129] metric to order the terms in each chunk according to how informative they are.

### 3.3.9 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. These errors might make classification tasks more problematic. 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 [206] 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 step generally improves performance.

Eventually, an interesting way to perform spell-checking is presented in [291] where a spell checker is employed to improve stemming, while synonyms of related tokens are combined.

### 3.3.10 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 [186]. 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[2]. 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 [270] 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 [24], in order to tag opinion statements with sentiments, the authors employ POS tags as pointers. In the literature, exist dozens of different tag sets, defined in the context of different theoretical frameworks and also designed to represent morphologically different languages. The above-mentioned tag set is the one related to a popular project of the last century for the construction of a treebank of English language (i.e., the *Penn Treebank*). The tag set is still used today, but has been superseded by others more suited to represent not only the English language. One of the most relevant is the tag set project of Universal Dependencies[3].

Some popular libraries and tools that use rule-based approaches to perform POS tagging are the

[2] An example from Twitter is the case of a retweet replaced by the tag *RT*
[3] `https://universaldependencies.org/`

NLTK library's *pos_tag()*[4] and the *TextBlob*[5] Python library. Other libraries based on statistical models are the *spaCy library's POS tagger*[6] that is trained on the OntoNotes 5 corpus and the *Averaged Perceptron Tagger in NLTK*[7] that is based on the above-mentioned tag set project of the Universal Dependencies.

Specially in deep learning-based models, this process of assigning POS 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 [36], while others found POS tagging useful [11].

### 3.3.11 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 [97], lemmatization reduces various word forms to the same lemma to enhance user sentiment extraction effectiveness. Lemmatization is discussed in [48] and, in the context of an SVM model, in [160]. In [154] authors address the issue of ambiguity after lemmatization. Authors use lemmatization in combination with POS disambiguation to alleviate the problem.

Lemmatization has long been a common preprocessing step for traditional models. Since deep learning models 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 [104]. 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.

### 3.3.12 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

[4]`https://www.nltk.org/api/nltk.tag.pos_tag.html`
[5]`https://textblob.readthedocs.io/en/dev/quickstart.html`
[6]`https://spacy.io/api/tagger`
[7]`https://www.nltk.org/api/nltk.tag.perceptron.html`

after stemming, since derived words share semantic similarities with their root forms.

The first known stemming algorithm was presented in 1968 and discussed in [178]. Going forward, the algorithm for stemming introduced in [230] 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 [267] and also discussed in [289]. The goal of stemming in both studies is to find, for any derived word, its corresponding stem. As discussed in [92], the stemming algorithm depends on the language considered (i.e., Turkish in this case). The library commonly used for Turkish language is discussed in [4]. For the same language, the fixed-prefix approach described in [50] 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 authors in [96]. Although advanced algorithm employ morphological understanding 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 the Indian language, as well as popular stemming strategies.

Truncating approaches, statistical methods, and mixed methods are typically used to apply 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 [203].

- *Truncating techniques* involve removing a word's prefixes or suffixes, referred to as affixes. Truncating a word at the n-th 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 developed in 1980 [230]. On the fundamental algorithm, numerous alterations, improvements, and suggestions have been proposed. The original algorithm is based on the fact 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 stage 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. A stemming framework named *Snowball* was created by Porter. The primary goal of the framework is to give developers the freedom to create custom stemmers for different languages or character sets.

- *Lovins stemmer* was proposed in 1968 [178]. The Lovins stemmer eliminates a word's longest suffix. Each word is altered, checking a different table that performs numerous alterations to turn these 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 [213] 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 issues 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.

### 3.3.13 Removing elongation

A character that is repeated once or more times can be found in elongated words (e.g. *coooooool*, *greeeeeeat*, *goooood* etc.). Tweets and other social media posts frequently contain words with repeated letters that can be managed to better mining sentiment [19]. 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.

### 3.3.14 Emoticon and emoji handling

On the internet and in social networks, emotional icons are frequently used to denote users' sentiment [110]. Users use different 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 [293] authors study and evaluate the impact of emoticons on sentiments of tweets. Authors demonstrate the value of emotional icons in conveying messages on social media. In [218], the usefulness of processing emoticons on user-generated content is highlighted by the 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 *sad* in place of *:-(*. 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 [2] 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.

### 3.3.15 Negation handling

As stated in [17], 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 [199].

In [17] authors handle negation performing the following steps. At first, they compile an antonym dictionary using the WordNet dataset. In their work, 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 considered. 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.

Negation handling, mentioned here, usually solves the problem considering the presence of particles or adverbs of denial. Indeed, to treat negations effectively also on a larger portion of text (instead of single words), parsing strategies apply.

### 3.3.16 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 a hashtag within the shared post. In this case, word segmentation would change the desired usage of the author, as reported in [210]. Nevertheless, segmenting merged words has proved to be helpful in understanding and better classifying contents of tweets and posts[214][301].

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).

## 3.4 Experiments on text preprocessing

To assess the impact of the three most common techniques (i.e., lowercasing, removing stop words and stemming) I performed several experiments. I evaluated the impact of the single techniques but also the impact of all the possible combinations of them. In Figure 3.2 is shown the process I applied for the experiments. As can be seen from the figure, the application order of each technique is relevant. For this reason, I evaluated the preprocessing techniques in order. Even if in Figure 3.2 I only show a running example using one, two or three techniques, in the result section I present and discuss the effect of using all the possible combinations of two and three techniques. The libraries I used to apply the techniques were already presented and referenced in the Section 3.3.

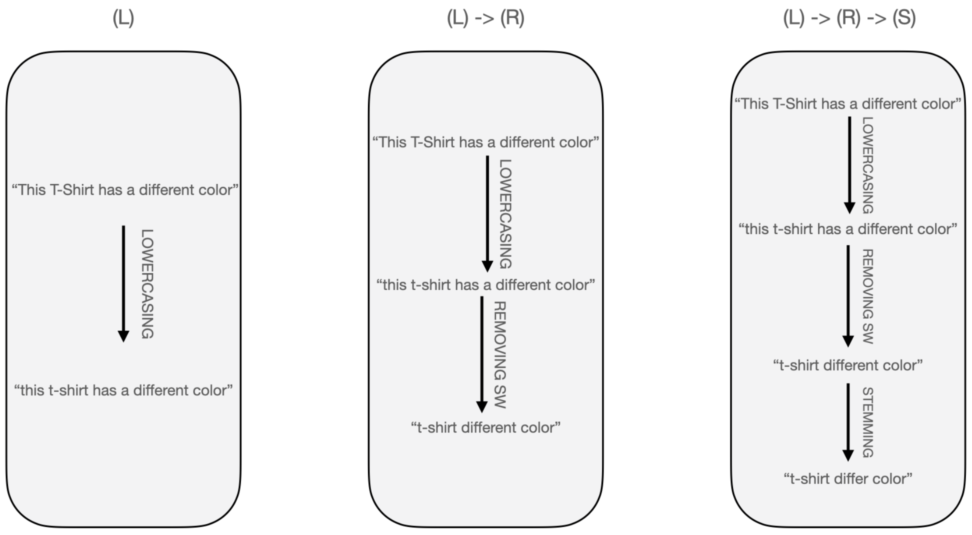

(L)
“This T-Shirt has a different color”
LOWERCASING
“this t-shirt has a different color”
(L) -> (R)
“This T-Shirt has a different color”
LOWERCASING
“this t-shirt has a different color”
REMOVING SW
“t-shirt different color”
(L) -> (R) -> (S)
“This T-Shirt has a different color”
LOWERCASING
“this t-shirt has a different color”
REMOVING SW
“t-shirt different color”
STEMMING
“t-shirt differ color”

Figure 3.2: From the left to the right 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 the experiments, I evaluated the combinations of the three most common techniques.

### 3.4.1 Evaluated models and datasets

I introduce the models and the datasets evaluated in the experiments in this section. For the traditional models I provide the reference to the libraries used and in the Section 3.4.2 I further discuss parameters setup for the tested configuration. For the deep learning models, I 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 traditional and to deep models. For the traditional models the vector of ints is translated into a bag-of-words representation[8], while for the deep models the vector of ints is used as-is by the following word embedding layer. In the case of the deep learning 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.

- **Logistic Regression (LR)**. LR is commonly employed in TC for several tasks [254]. 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 [262]. For the experiments, I used the *sklearn* Logistic Regression implementation[9]. I used an L2 penalty, a C value equal to 1.0 and the *lbfgs* solver as discussed in [47].
- **Naïve Bayes (NB)**. As reported in [189] and experimentally demonstrated over time by outcomes from various TC tasks [240], NB is one of the most effective model to employ for classification. I evaluated a multinomial NB classifier from the *sklearn* MultinomialNB implementation[10]. Data are commonly expressed as word vector counts.

[8]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

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

[10]`https://scikit-learn.org/stable/modules/generated/sklearn.Nave_bayes.MultinomialNB.html`

- **SVM**. As reported in [58] and in [174], classifiers based on SVM are well-established methods for TC tasks. SVM are also employed in ensemble-based text classifier, as reported in [63]. Thanks to SVM models, classification results compared to other classification methods have been greatly improved. Based on [52], I tested the *sklearn* SVC implementation[11]. As a regularization parameter, I 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 [191] along with the development of the perceptron [245], nowadays, ANN are widely implemented on a wide range of tasks. TC is no exception. The architecture I implemented for the 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 Figure 3.3.
- **Convolutional Neural Network (CNN)**. The CNN evaluated here is the one presented in [259] and also used in [184]. In this case I do not report further details or the image of the network architecture which can be found in the above-mentioned papers. Such a CNN consists essentially of a single convolutional layer. As demonstrated by its results, this CNN outperforms Transformers and others proposed models as stated in [239] on a classification task similar to the ones proposed in this PhD thesis.
- **Bidirectional LSTM (BiLSTM)**. In place of feed-forward networks, recurrent neural networks are widely utilized to categorize text data. In [211], authors discuss how to perform TC using LSTM network and their variants like BiLSTM and GRU. For the work, I developed a simplified version of the BiLSTM discussed in [258]. Also, in this case I do not report further details or the image of the network architecture which is presented in [258]. The model consists of two bidirectional LSTM layers. I did not employ any activation functions for any of the dense layers. I used a binary cross-entropy loss and the optimization algorithm by Adam [141] to train the model.
- **RoBERTa**. Authors in [173] — 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 I used is presented in [173].
- **ELECTRA**. According to what stated in [57], ELECTRA suggests replacing certain tokens with possible replacements taken from a small generator network, instead of masking the input

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

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 [176]. In the experiments, the original version of ELECTRA, presented in [57], was used.

- **XLNet**. A generalized autoregressive pretraining strategy is the one suggested in [304]. 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 this study I used the pre-trained XLNet using zero-shot cross lingual transfer discussed in [53].

A recent rise in the application of classification techniques based on graphs is noteworthy. A recent study can be found in [176] for TC but, recently, graph-based methods are also used for traffic prediction [166], computer vision [234] and social networking [257]. However, most of these methods are not yet able to outperform models evaluated and discussed here.

The four datasets evaluated in this study come from different domain, and they have been already presented in the Chapter 2. They are, namely: the **FNS**, the **PCL**, the **IMDB** and the **20N** datasets. I describe their structure, content and their respective size in the above-mentioned chapter. As already stated, all the four datasets used are publicly available and used in recent literature for TC tasks. I used datasets with varying sizes and distinct classification objectives to examine how each preprocessing strategy affects different classification tasks.

### 3.4.2 Experimental setup

The experiments were performed using TensorFlow on an NVIDIA GeForce RTX 2080 GPU on a local machine and on a Tesla T4 from Google Cloud. For the Transformers, I used the *Simple Transformers*[12] library. Each of the Transformers used came from the library of Transformers provided in [299]. The batch size for all models was 1. For 10 epochs, I 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. I followed the protocol used in [173] to evaluate the performance of each deep model tested. So, I initialize each model with random weights, and then I run the training and the evaluation phases for five times (with early stopping

[12]`https://simpleTransformers.ai/about/`

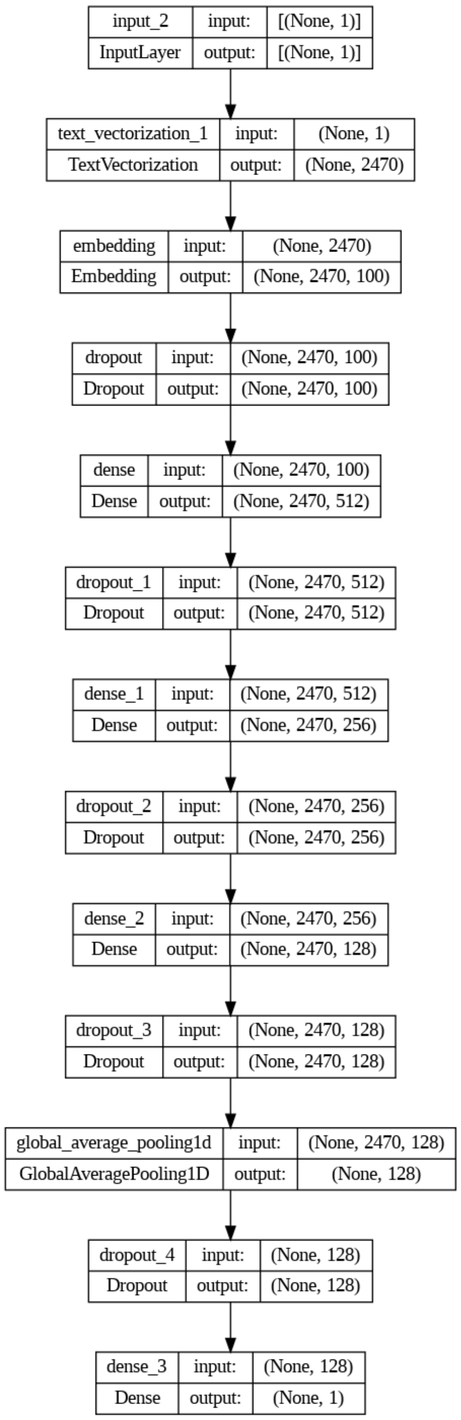

input_2 | input: | [(None, 1)]
InputLayer | output: | [(None, 1)]
text_vectorization_1 | input: | (None, 1)
TextVectorization | output: | (None, 2470)
embedding | input: | (None, 2470)
Embedding | output: | (None, 2470, 100)
dropout | input: | (None, 2470, 100)
Dropout | output: | (None, 2470, 100)
dense | input: | (None, 2470, 100)
Dense | output: | (None, 2470, 512)
dropout_1 | input: | (None, 2470, 512)
Dropout | output: | (None, 2470, 512)
dense_1 | input: | (None, 2470, 512)
Dense | output: | (None, 2470, 256)
dropout_2 | input: | (None, 2470, 256)
Dropout | output: | (None, 2470, 256)
dense_2 | input: | (None, 2470, 256)
Dense | output: | (None, 2470, 128)
dropout_3 | input: | (None, 2470, 128)
Dropout | output: | (None, 2470, 128)
global_average_pooling1d | input: | (None, 2470, 128)
GlobalAveragePooling1D | output: | (None, 128)
dropout_4 | input: | (None, 128)
Dropout | output: | (None, 128)
dense_3 | input: | (None, 128)
Dense | output: | (None, 1)

Figure 3.3: The ANN architecture implemented for the experiments. Numbers in brackets indicate tensor dimensions. Layers as depicted on the Google Colab Notebook.

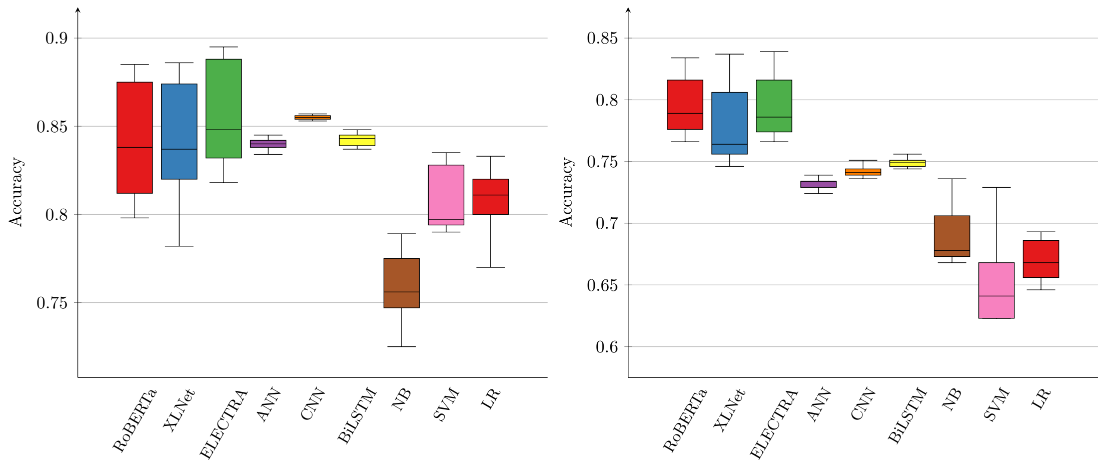


Figure 3.4: Box plot for the nine models evaluated on the IMDB (left) and on the PCL (right) dataset. The deep learning models are the less sensitive to the preprocessing strategy employed, while the Transformers are the most sensitive.

for each run), then I report the median accuracy along the five runs, as the representative result of each model. Furthermore, in the Section 3.4.3 I 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 the experiments. For the three traditional models, I use the implementations discussed in the Section 3.4.1 and because of their deterministic nature there is no need of performing multiple runs. I have previously provided references to each model's original implementation, along with each architecture's experimental setup. The datasets I used are described in the preceding section and are accessible upon request.

### 3.4.3 Results

I report my comments on the results in this section. The results reported in this section concern the impact of the three most common preprocessing techniques. The experiments investigated not only the effect of single techniques but also in combination. On the three dataset used, from Table 3.3 to Table 3.6 the results of the 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, I 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, I report the results obtained using the combination of the three technique.

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 Figure 3.4 and in Figure 3.5 I 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 3.3 to 3.6 and each box represents the result distribution for the model indicated on the *x-axis*.

### 3.4.4 IMDB

For the IMDB dataset, the results of the deep models are shown in the Table 3.3. The best performance is obtained by ELECTRA with lowercasing as preprocessing technique. The use of the same model 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 the 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 deep 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 the Figure 3.4. In fact, the size of the boxes related to the result distributions of the deep 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 traditional models on the same dataset are reported in Table 3.6. 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.

### 3.4.5 PCL

The results of the deep models for the PCL dataset are reported in Table 3.4. 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 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, deep models do not highlight 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 using 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 traditional models are reported in Table 3.6. 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.

### 3.4.6 FNS

The results of the deep models are reported in Table 3.5 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 Figure 3.5. 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 samples with respect to other datasets. In fact, any sample consists of the last 100 tweets of a Twitter user. As widely discussed in [256], traditional and deep models 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, deep models do not exhibit a great difference between the worst and the best results, while varying the combination of techniques applied. Considering the deep 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 traditional models on the same dataset are reported in Table 3.6. 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 logistic regressor and of the SVM, the worst performance is obtained performing no preprocessing at all.

### 3.4.7 20N

The results obtained by the three traditional models on the 20N dataset are reported in Table 3.6. It is worth repeating that this dataset entails a multi-class classification problem, 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 the NB model. In this case the gap between the best and the worst result is irrelevant, and I do not even highlight the best results obtained almost in every preprocessing combination. The LR shows the most variable behavior in terms of results.

Table 3.3: Median accuracy and maximum gap from the median accuracy of the three deep models on the IMDB dataset. In bold black and 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 |

Table 3.4: Median accuracy and maximum gap from the median accuracy of the three deep models on the PCL dataset. In bold black and 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** |

Table 3.5: Median accuracy and maximum gap from the median accuracy of the three deep models on the FNS dataset. In bold black and 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 3.6: Accuracies for the three non-deep models on the three test dataset used. In bold black and red are shown the best and the worst results, respectively, for each model. For NB on 20N, I 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 |

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, I do not show the table about the deep and 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, I do not show the box plot for all the models. In fact, for this dataset, I 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 traditional 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 deep 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 the experiments are available on GitHub.

### 3.4.8 Discussion

From a theoretical point of view, I have empirically shown that the text preprocessing strategy can affect the performance of any modern classifiers, including the most recent Transformers-based architectures (RQ2). Along with the different datasets used can be seen that preprocessing only marginally affects deep models, while the most significant impact is on the Transformers. It is likely that this result depends on the word embedding for the two classes of models. While the Transformers make use of a pretraining phase, the embedding trained from scratch in the case of the deep models could be the main cause of the less sensitivity to the preprocessing strategy applied. In Figure 3.4 similar results between the IMDB and the PCL dataset can be observed. Interestingly, the traditional 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 SOTA results, outperforming more complex and recent pre-trained architectures (i.e., the

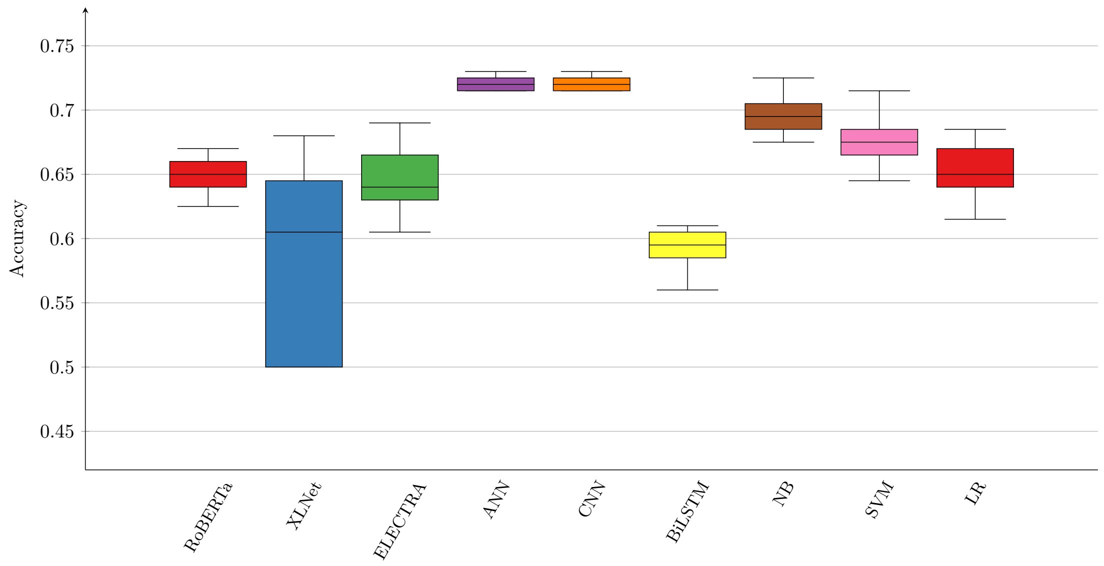


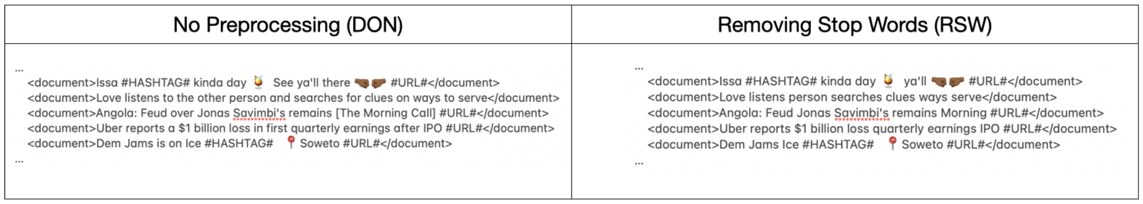
Figure 3.5: Box plot for the nine models evaluated on the FNS dataset. On this dataset, eight out of nine models show minimal sensitiveness to the preprocessing strategies.

Transformer-based ones). I 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 it appears irrelevant to focus on preprocessing when dealing with deep 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 a 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 Figure 3.4 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

Figure 3.6: Effect of no-preprocessing and of one of the preprocessing strategy on a small part of a single sample from the FNS dataset.



| No Preprocessing (DON) | Removing Stop Words (RSW) |
|---|---|
| ... | ... |
| <document>Issa #HASHTAG# kinda day 🍹 See ya'll there 🤜🤛 #URL#</document> | <document>Issa #HASHTAG# kinda day 🍹 ya'll 🤜🤛 #URL#</document> |
| <document>Love listens to the other person and searches for clues on ways to serve</document> | <document>Love listens person searches clues ways serve</document> |
| <document>Angola: Feud over Jonas Savimbi's remains [The Morning Call] #URL#</document> | <document>Angola: Feud Jonas Savimbi's remains Morning #URL#</document> |
| <document>Uber reports a $1 billion loss in first quarterly earnings after IPO #URL#</document> | <document>Uber reports $1 billion loss quarterly earnings IPO #URL#</document> |
| <document>Dem Jams is on Ice #HASHTAG# 📍 Soweto #URL#</document> | <document>Dem Jams Ice #HASHTAG# 📍 Soweto #URL#</document> |
| ... | ... |

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 such a case the use of stemming leads to the best result obtained by the CNN.

For the multi-class classification task regarding the 20N dataset, I 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.

### 3.4.9 Qualitative analysis

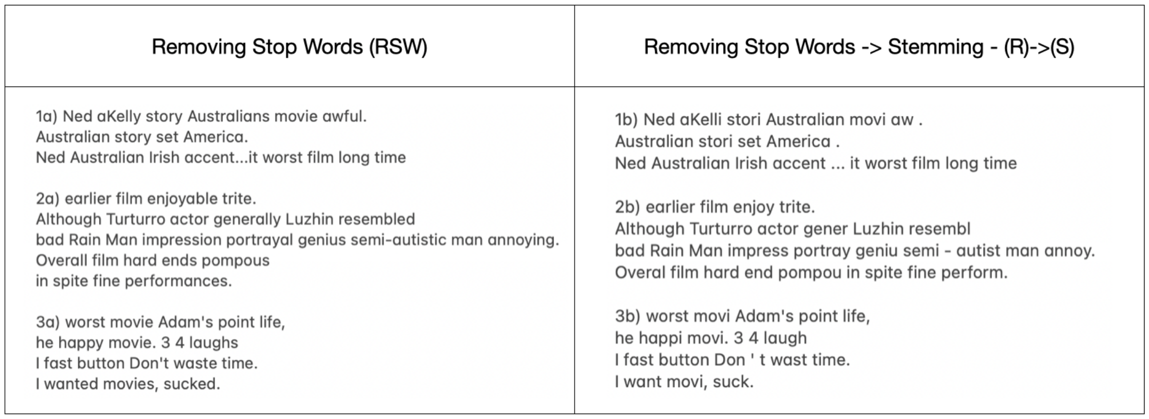
In this section, I conduct a brief qualitative analysis to show how the text preprocessing alters dataset samples and how the specific strategy affects the performance.

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 Figure 3.6. 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 available.

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 written, 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 the Figure 3.7 it is easy to understand the reason preprocessing is so

Figure 3.7: 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.



| Removing Stop Words (RSW) | Removing Stop Words -> Stemming - (R)->(S) |
|---|---|
| 1a) Ned aKelly story Australians movie awful. Australian story set America. Ned Australian Irish accent...it worst film long time | 1b) Ned aKelli stori Australian movi aw . Australian stori set America . Ned Australian Irish accent ... it worst film long time |
| 2a) earlier film enjoyable trite. Although Turturro actor generally Luzhin resembled bad Rain Man impression portrayal genius semi-autistic man annoying. Overall film hard ends pompous in spite fine performances. | 2b) earlier film enjoy trite. Although Turturro actor gener Luzhin resembl bad Rain Man impress portray geniu semi - autist man annoy. Overal film hard end pompou in spite fine perform. |
| 3a) worst movie Adam's point life, he happy movie. 3 4 laughs I fast button Don't waste time. I wanted movies, sucked. | 3b) worst movi Adam's point life, he happi movi. 3 4 laugh I fast button Don ' t wast time. I want movi, suck. |

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 much 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 the 18%. This result is easy to be understood, looking at the three samples from the dataset shown in the Figure 3.7.

### 3.4.10 Conclusion and future works

In this study, I have presented the most popular preprocessing techniques found in the literature. I have then evaluated and compared the effect of the three most common techniques 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 chapter also lists the worst- and best-performing strategies in terms of the dataset and the model, 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. The analysis emphasizes how crucial preparing data is to ensure consistency when comparing various learning models. Moreover, the research highlights that, depending on the preprocessing method selected, the results are highly variable, also using modern Transformers. The 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 this 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 chapter, I 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 and better understanding 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 I 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 combination 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 [216] [239] [32], concerning three different author profiling tasks, the best performance obtained by the traditional and the deep models in place of the Transformers should be further investigated. For all of these author profiling datasets, the impact of preprocessing could be investigated to further corroborate some findings reported in this study. Finally, different preprocessing methods could also be used to investigate and understand in greater depth behaviors of deep and Transformer models. The benefit could be to unveil some interesting mechanisms happening under the hood, with particular regard to the field of the deep learning.

# Chapter 4

# Representation

Before moving to the classification stage, it is necessary to convert unstructured data, especially free-running text data, into organized numerical data. To do this, a document representation model must be used to employ a subsequent classification system following the text preprocessing stage. Text representation models convert text data into a numerical vector space, which has a substantial impact on how well subsequent learning tasks can perform. In the history of NLP, word representation has always been a topic of interest. It is crucial to properly represent such text data, since it contains a wealth of information and may be applied broadly across a variety of applications. This chapter examines the expressive potential of several word representation models, ranging from the traditional to the contemporary SOTA word representation language models. Model designs, including language models, have been explored, and a range of text representation techniques have been examined. These models are capable of turning massive amounts of text into useful vector representations that effectively capture relevant semantic data. Furthermore, different machine learning models can make use of these representations for a range of NLP tasks. If it is able to effectively capture intrinsic data properties, better text representation will probably lead to superior performance. I also briefly discuss the drawbacks of the provided representation models in the sections that follow. In detail, after the preprocessing of a raw text, the next stage is to perform a probabilistic tokenization accordingly to a split strategy. Probabilistic tokenization consists in separating text units and converting it into a numerical representation. In automatic TC, a single word is one of the most common elements to use as the unit from a text. In this case a single n-gram is referred to a single word.

Even if it is not properly a text representation method, to represent a unit of text, the n-gram can be employed as a feature. A representation that makes use of single words (1-gram), regardless of the order, is called a BoW. This approach is fairly simple to implement. It represents text as a vector, typically with text that is manageable in size. The terms 2-gram and 3-gram are frequently used. When two or more *grams* are used in place of a single gram (i.e., word) the term *n-gram* can be used. An illustration of a 2-Gram is given in the following clause:

- *"Once upon a time you dressed so fine."*

In the proposed example, the tokens would be:

- {*“Once upon”, “upon a”, “a time”, “time you”, “you dressed” “dressed so”, “so fine”*}

An Example of 3-Gram:

- *"Once upon a time you dressed so fine."*

In the proposed example, the tokens would be:

- { *“Once upon a”, “upon a time”, “a time you”, “time you dressed”, “you dressed so”, “dressed so fine”*}

It is worth mention that also split strategies at character level have been reported in the literature, as in [311], where the authors show that a character-level CNN achieve SOTA performance. Comparisons are made between deep models like word-based ConvNets and RNN and more conventional models like BoW, n-grams, and their TF-IDF variations. In this case, considering a sentence like:

- *"Purple Haze"*

The tokens are as follows:

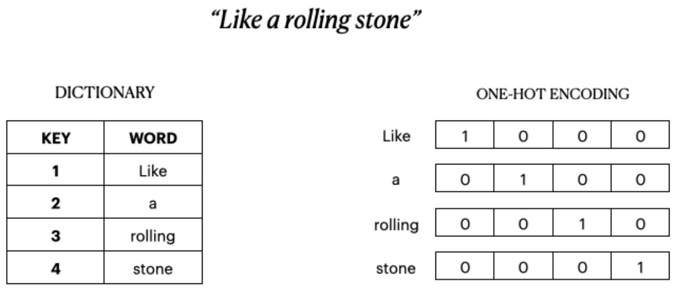
- {*"P", "u", "r", "p", "l", "e", "H", "a", "z", "e"*}

The remainder of this section discusses numerous representation models that are frequently employed. In the past, several researchers have put forth various theories to solve the issue of words losing their syntactic and semantic links with the chosen representation. These techniques are presented along with the literature review. First I present some statistical methods, then relevant representation learning and pre-trained language models are discussed too.

## 4.1 Text representation models

### 4.1.1 Statistical models

The earliest and simplest methods for representing textual data are statistical word representation techniques. Early classification models for computer vision, information retrieval and NLP made



*"Like a rolling stone"*

DICTIONARY

| KEY | WORD |
|---|---|
| 1 | Like |
| 2 | a |
| 3 | rolling |
| 4 | stone |

ONE-HOT ENCODING

| | | | | |
|---|---|---|---|---|
| Like | 1 | 0 | 0 | 0 |
| a | 0 | 1 | 0 | 0 |
| rolling | 0 | 0 | 1 | 0 |
| stone | 0 | 0 | 0 | 1 |

Figure 4.1: One-hot encoding example

heavy use of these word representation models. It is easy to design and apply this class of models in almost every existing task. Despite their simplicity, the following are a few drawbacks of such models: a) they do not take word order into account; b) they do not take word relationships into account; c) the input vector and the vocabulary are proportional in size, making them computationally expensive, which may lead to subpar performance.

These models, that were frequently employed in the past for TC, are presented in this section. This type of words representation approaches are based on word frequency. These techniques convert text into a vector form that includes a number that in some way quantifies a word's usage frequency inside a text. Common statistical techniques that are frequently employed in the literature are briefly described in the sections that follow.

### One-hot encoding

A basic way to represent text is the one-hot encoding. With the one-hot encoding, It is converted each categorical value into a new categorical column, and it is assigned a binary value of 1 or 0 to those columns. One hot encoding has a dimension equal to the number of vocabulary terms. Vocabulary terms are all represented as vectors of binary values (i.e., 0 or 1). After mapping each token into an integer value, to represent the integer value a binary vector is used. All the values are zero, and the vector index of the considered word is marked with a 1. Every distinct word has its own dimension, which is a single 1 in that dimension and 0s in the other dimensions. With one-hot encoding, all words in the dictionary are orthogonal to each other.

Considering the following sentence:

- *"Like a rolling stone"*

The one-hot encoding representation is depicted in Figure 4.1.

### Bag of Words (BoW)

Bag-of-Word (BoW) is another method to represent a document. BoW is used to form a vector representing a document using the frequency count of terms in the text. This method of representation is also referred to as a *vector space model*. By viewing text bodies as unordered groups of words, this method reduces complex texts. The consequence of this is that phrase semantic and structure connections of text pieces are ignored (they are "thrown" into a "bag" of words). Nevertheless, it has been demonstrated that, despite its robust assumptions, it can successfully complete a number of classification tasks.

The fundamental principle of BoW models is that each word is represented as a one-hot-encoded vector of size equal to the vocabulary. As a result, approaches based on BoW are frequently used along with feature extraction methods that take into account the diversity of words, enabling the preservation of a single vector per document as opposed to one for each word. The cardinality of the vocabulary alone could be in the millions, therefore it is immediately obvious that this could cause size problems.

The BoW method is employed in a number of fields, including machine learning for computer vision, Bayesian spam filters and document categorization. A body of text, such as a sentence or a document, is viewed in a BoW as a bag of words. The BoW procedure produces word lists. These words' semantic relationship is not taken into consideration in their gathering and construction, since the words in a matrix are not sentences that structure sentences and grammar. A sentence's meaning can often be inferred from its terms. The main topic of the corpora may later be ascertained by counting multiplicity rather than grammar or appearance order.

Unfortunately, the BoW representation scheme has its own limitations. Some of them are: high dimensionality of the representation, loss of correlation with adjacent words and loss of semantic relationship that exist among the terms in a document. Additionally, because there could be millions of words in a vocabulary, BoW models have trouble scaling up (for instance, "I love you" and "you love me" have the same vector representation). So, size is one of the main issue facing the community of computer scientists and data scientists by BoW.

A BoW representation example is depicted in Figure 4.2.

### Term Frequency-Inverse Document Frequency

Term Frequency (TF), often used with BoW, is another technique for text representation. The approach allocates the feature space to the number of token in each document. The simplest way for weighing words is called TF, and it maps a single word to a number that represents how many times it appears across the entire corpus.

Word frequency is frequently used as a boolean or a weighted with a logarithmically scaled scale in methods that expand the findings of TF. In all weight words techniques, the word frequencies in

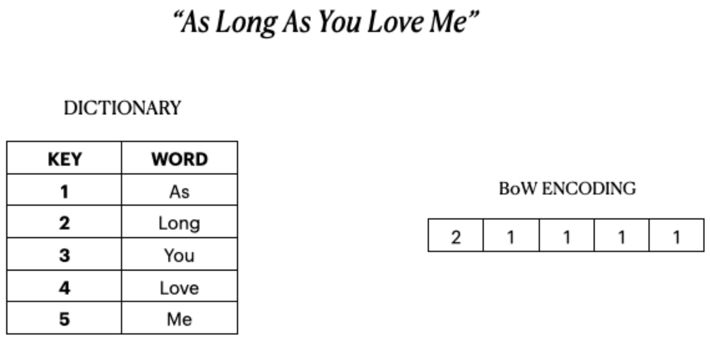


Figure 4.2: BoW encoding example

each document are converted into a vector. This method is obvious, but it has limitations because it may be dominated by words that are often used in the language.

When used for a corpus of texts, the relative frequency of a word in a single document in contrast to other documents is typically used instead of the explicit count. Another thing to note is that popular terms, in particular, are naturally worth less in very large corpora. Thus, TF is frequently weighted by Inverse Document Frequency (IDF). IDF penalizes their overall score in order to decrease the impact of popular terms (and boosting the one of rarer words). Term Frequency-Inverse Document Frequency is the term used to describe the combination of TF and IDF (TF-IDF). The mathematical representation of TF, IDF and TF-IDF is given in the Equations 4.1, 4.2 and 4.3.

$$tf_{ij} = \frac{n_{ij}}{|D_j|} \tag{4.1}$$

$$idf_i = \log_{10} \frac{|D|}{|d_i|} \tag{4.2}$$

$$tf - idf = tf_{ij} \times idf_i \tag{4.3}$$

Here $n_{ij}$ is the number of occurrences of the term $i$ in the document $j$. The number of terms in the document $D_j$ is $|D_j|$. Looking at Equation 4.2, $|D|$ is the total number of documents and $|d_i|$ is the number of documents containing the term $i$.

It happens that TF-IDF representations will always be significantly large depending on the extent of the vocabulary. It is possible to cap the number of characteristics that can be included in the vectors in order to reduce difficulties with memory usage and time complexity. A dimensionality reduction approach can also be applied to the full-sized representations as an alternative.

Even though TF-IDF attempts to address the issue of common terminology in the document, it still has certain descriptive shortcomings. Specifically, because each word is supplied as an index separately in the document, TF-IDF is unable to account for the similarity between the words. But more recent developments in complex models have given rise to new approaches, like word embedding, that can take into account ideas like word similarity and POS tagging.

### 4.1.2 Word embedding models (from scratch)

Statistical word representation methods are plagued by the large dimensionality of dictionaries and are unable to catch semantic and syntactic meaning of words. Due to these models' flaws, researchers learned how to disperse words in low-dimensional space. Statistically based approaches limit their use in developing an appropriate model in machine learning. Although prior methods primarily collected syntactic representations of words and little portions of syntactic relationships that connecting them in phrases, they essentially still fall short of being able to capture their semantic meaning. Word synonyms serve as a prime illustration of this problem; although being semantically equivalent, these models are unable to adequately reflect their resemblance. This results in representations that are orthogonal to one another when viewed in the context of the feature space, i.e., they are viewed as entirely distinct from one another. Despite the fact that representing words considering the syntax, does not imply that the approach accurately reflects the word semantics. BoW models, on the other hand, disregard the word's meaning. The terms "auto", "car", "automobile", for instance, are frequently employed in texts interchangeably. BoW model's vectors for these words, however, are orthogonal. Understanding sentences within the model is severely hampered by this problem. The phrase's BoW also has the issue of disrespecting the phrase's word order. This issue cannot be resolved by the n-gram, hence a similarity must be found between each word in the sentence. As a result, numerous models were put out in the past, each of which automatically finds the representations for subsequent tasks like classification. These techniques that automatically discover features are referred to as feature learning or representation learning. This topic is crucial, since machine learning models depend greatly on how they represent their input. Traditional feature learning approaches are being replaced by deep learning-based models, characterized by the self-learning crucial features. Both supervised and unsupervised learning techniques can be used to learn proper representation. Statistical text representation techniques have been supplanted in the field of NLP by unsupervised text representation techniques like word embeddings.

The objective of this method is to infer a mapping from each text component, which is often a word, to an n-dimensional vector of continuous values. In sight of this, an embedding is an array that can be processed by a computer and also conceals some true words semantic. These techniques rely on artificial neural networks, which create these mappings using a variety of learning strategies. They generally rest premising that the meaning of a word can be inferred from the ones that come before and that follow in a text.

Due to the word embeddings' prior understanding of many machine learning models, they became particularly effective representation techniques to enhance the performance of various downstream tasks. These neural network-based algorithms have superseded traditional feature learning techniques because of their strong representation learning capabilities. This new class of models has a major impact on how well downstream learning tasks work. The scientific community has widely used representation learning techniques to create effective models. Word2Vec, GloVe, and FastText are examples of continuous word representation methods that have significantly enhanced classification outcomes and eliminated categorical representations' drawbacks. The ability of these continuous word representations to capture more semantic and syntactic information of the textual data without sacrificing much information is discovered to have a greater impact than traditional linguistic features. Because they encode words out of context in their most fundamental form, earlier word embeddings are frequently technically classified as "static." Despite their effectiveness, there are still some challenges they are unable to resolve, such as the fact that they assign each word the same vector while ignoring its context, making them unable to manage polysemy problems. Therefore, practically speaking, they do not model polysemy (where an individual word can have different meanings). A word only has one embedding, regardless of how many meanings it may have; if a word token is extremely polysemous, it is likely that its embedding will combine its several senses. Consider the term *sound*, for instance. As a noun, it can be identified as something audible, yet as an adjective, it can describe something/someone that is in good shape. However, these are only two of the almost 50 possible meanings for this term; in light of this, it should be clear that no one depiction can effectively convey all of them at once. Furthermore, models like Word2Vec and GloVe assign a random vector to a word that they did not come across during training, making them incapable of handling out of vocabulary (OOV) terms that were resolved by FastText, which separates words into n-grams. The performance of TC is lowered by each of these restrictions. Additionally, none of the current SOTA algorithms work well with low-quality text.

Unigrams can be converted into intelligible input for machine learning algorithms using a variety of word embedding techniques. The remaining portions of this section introduce Word2Vec, GloVe, and FastText, three of the most popular approaches that have been productively applied to deep learning techniques. Then are introduced a few context-based representation techniques.

### Word2Vec

Authors in [197] proposed one of the earliest well-known families of word embedding designs. To produce a high-dimensional vector for each word, their method uses shallow neural networks. When Word2Vec was initially introduced, the Continuous Skip-gram and the Continuous-Bag-Of-Words (CBOW) were included. By attempting to guess a central word from its context-dependent surroundings, the CBOW technique learns word representations. On the other hand, a skip-gram model turns the task by attempting to predict a word's neighbors. These are undoubtedly challenging problems,

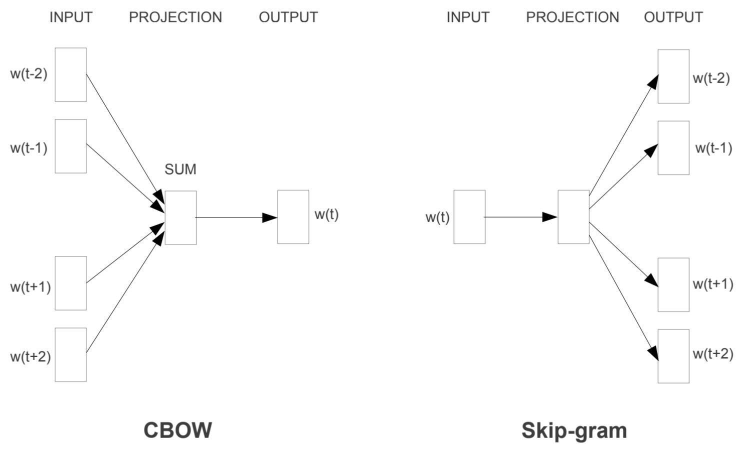


Figure 4.3: The original picture from the work on CBOW and Skip-gram models presented in [197].

and the architecture is not intended to know how to guess the words exactly; rather, the real goal is to generate meaningful mappings between words and embeddings rather than to accurately forecast the words.

The original image from [197] is displayed in Figure 4.3. A basic CBOW model is shown in the picture. This approach offers a highly potent tool for identifying linkages in corpora and/or word similarities. For instance, the embedding can take into account the proximity of two words in the vector space it gives them, such as "large" and "bigger."

**Continuous BoW Model**. For a specific objective of words, the continuous BoW model uses many words as representation. As context words for the target term "air-force," "airplane" and "military" come to mind. This entails $n$ replications of the input to hidden layer connections, where text in it is the quantity of context words. Making a vocabulary, or a list of all the original terms in the corpus, should come first. The task will be "predicting the term given its context," according to the shallow neural network's output. The quantity of words utilized is determined on the window size setting (common size is 4–5 words).

**Continuous Skip-Gram Model**. This architecture is very close to that of CBOW, but it seeks to maximize categorization of a word based on the preceding word in the same phrase rather than anticipating the next word based on context. For machine learning algorithms, the syntactic and semantic content of sentences is preserved using the continuous BoW and Skip-gram model.

### Global Vectors for Word Representation (GloVe)

Another word embedding method is GloVe [222], which stands for Global Vectors for Word Representations. The method is comparable to Word2Vec, but it varies fundamentally in that it uses a count-based model as opposed to Word2Vec's typical predictive architecture. In contrast to count-based models, which essentially determine semantic relatedness between words by explicitly exploring the underlying statistics of the corpus, such as word co-occurrence, predictive models define word vectors by reducing the loss between the target and prediction given context words and vector representations. Word2Vec just uses local information (the context of each word) to build word vectors, whereas GloVe embeddings are trained considering global co-occurrence data. The huge size of the matrix for word co-occurrence that GloVe model employs in its computations need to use a dimensionality reduction phase, which is another thing that should be taken into consideration. This technique is more suited for parallelization, which facilitates training on larger datasets. Although it is debatable that compressing representations makes them more robust, the fact that this strategy may be applied to larger datasets cancels out this benefit.

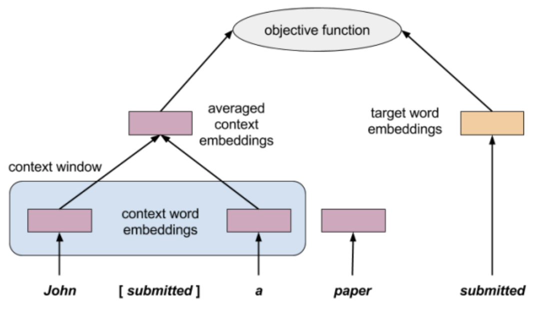
The embedding utilized in several publications is built with over four hundred thousand vocabularies learned across the corpora of Gigaword 5 and Wikipedia 2014, as well as 50 dimensions for word representation. Additional pre-trained embeddings with different dimensions (e.g., 100, 200, or 300) are also available from GloVe. These have been developed using training data from even larger corpora, such as Twitter content.

### FastText

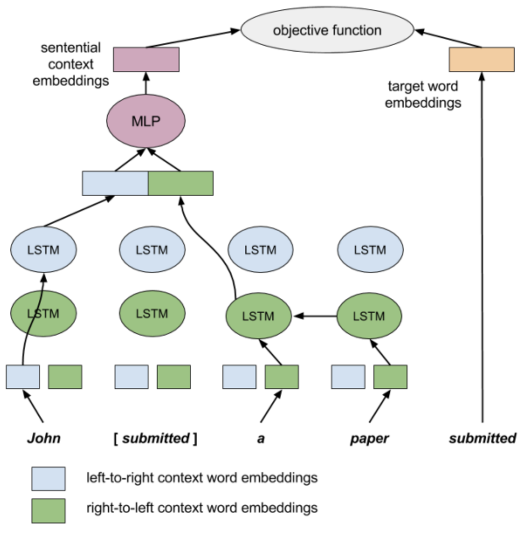
One of the best methods for static word embeddings is FastText, which was created by Bojanowski [37] at the Facebook AI Research lab. This method resolves the main problem that its forerunners ignore word morphology by assigning each word a distinct vector. Instead, an n-gram using bag-of-characters serves as FastText's representation of each word. For instance, the word "house" with n = 3 would be represented as "ho", "hou", "ous", "use" and "se" along with the entire word as a unique sequence. The skip-gram architecture is used to train FastText embeddings. Yet, due to the way words are encoded, the final vector for a word will be composed of the sum of its character n-grams. As a result, since their n-grams are shared by more common words, it can build efficient word embeddings for unusual words. The significant part is that it also implies that FastText can handle OOV words provided that it has seen the n-grams that make up such words during training. Instead, OOV words are a case that neither GloVe nor Word2Vec can handle. Facebook released word vectors that have already been trained using FastText on Wikipedia and are available in 294 different languages.

### Generic Context word representation (Context2Vec)

This representation technique is presented in [193], and the original image, as compared to Word2Vec, is shown in Figure 4.4. The BiLSTM neural network used in the model replaces the model's word representation within a given window with a superior and more potent one. Using a large text corpus,



(a) word2vec *CBOW*


objective function
sentential context embeddings
target word embeddings
MLP
LSTM
John
[ submitted ]
a
paper
submitted
left-to-right context word embeddings
right-to-left context word embeddings

(b) *context2vec*

Figure 4.4: The original picture from the work on Context2Vec presented in [193].

a neural network was trained to embed words and sentence context in the same low-dimensional space, subsequently refines the model to reflect how the target words and their entire sentential context interact with one another.

### Contextualized word representations Vectors (CoVe)

Based on context2Vec, the CoVe model, was introduced in [190]. CoVe was constructed via machine translation, as opposed to the methods employed by GloVe (Matrix factorization) or Word2Vec (skip-gram or CBOW). Starting with GloVe word vectors, the authors' basic strategy was to pre-train a two-layer BiLSTM for attention sequence to sequence translation. Then, they coupled it with GloVe vectors to create a CoVe, the output of the sequence encoder, and employed it in a downstream task-specific mode with transfer learning. On a range of typical tasks, the authors demonstrate that adding these context vectors (CoVe) enhances performance over using solely unsupervised word and character vectors (SQuAD).

**Embedding from Language Models (ELMo)**

The authors of [227] describe a brand-new kind of deeply word representation using context that simulates both the intricate aspects of word use (such as semantics and syntax) and the ways in which these uses change depending on the linguistic environment (i.e., to model polysemy). ELMo, which provides rich contextual word representations, was proposed by the authors. The flexible nature of word use in grammar and semantics, as well as how these uses should change as the linguistic environment changes, are difficulties that researchers agree should be considered in a word representation model. They consequently provide a deep contextualized word representation technique to overcome the two issues. From a bidirectional language model, the word embedding are learned forward and backward. In contrast to other contextual word representations, instead of only using the last layer representations, ELMo uses a linear concatenation of the representations learned from the bidirectional language model. For the same term, ELMo offers many embeddings in various phrases. Both forward and backward language models of bidirectional language models employ the log-likelihood of phrases during the training phase. After concatenating the hidden representations obtained from the forwarding language model, the final vector is calculated. With relative error reductions ranging from 6 to 20% over strong base models, adding ELMo alone creates a new SOTA outcome for every task taken into account.

### 4.1.3 Language models

Identifying the subsequent word in a sentence is the most basic kind of language modelling, which is the ability in estimating the probability of a word given a number of words that come before or after it in the context. Despite being far older than neural networks, language models have played a significant role in several modern deep learning-based breakthroughs. Some early language models were n-gram models, which function by tying probabilities to word sequences (i.e., sentences). It makes sense that a better-organized sentence will result in a higher score; nevertheless, the precise meaning of this probability value depends on the job (e.g., an improved translation). Although the task is to calculate the likelihood of a forthcoming word, the task is related to assigning probabilities to full sentences and is structured as such. The Markov assumption, which is typically used in these models, states that the likelihood of a forthcoming term depends solely on the $k$ terms that came before it. The next developments in this field will rely on the Transformer [286] architecture because it has been demonstrated to be quicker and more efficient for language modelling than LSTM or CNN. Although Transformer will also be covered in the parts that follow, they are just briefly described here and throughout the remainder of this section as language representation models.

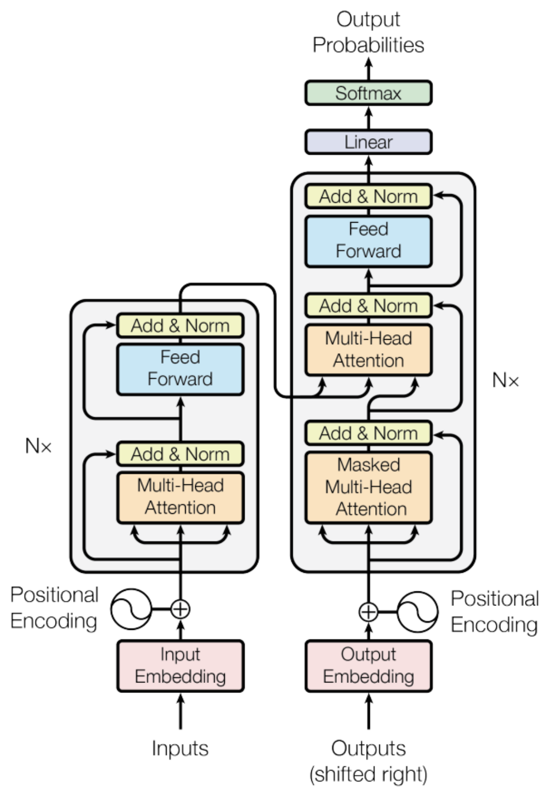


**Figure 1: The Transformer - model architecture.**

Figure 4.5: The original picture from [286].

Encoder-decoder structures are common in competitive neuronal sequence transduction models.The model is autoregressive at each phase, using the previous symbols as extra input to construct the next. Transformers' encoder converts an input series of symbol representations (x1,. . . , xn) into an equivalent sequence of continuous representations, z = (z1, . . . , zn). Then the decoder produces a sequence (y1,. . . , ym) of symbols, starting with z. In accordance with its general architecture, the Transformer uses layered self-attention and point-wise, entirely connected layers for the encoder and decoder. The general architecture of a Transformer is depicted in Figure 4.5 as presented in the original work in [286].

On downstream tasks, every Transformer-based architecture typically goes through the following steps: a) General language models pre-training; b) Target task language models fine-tuning; and c) Target task classifier fine-tuning. The language model's pre-training is unsupervised, and because there are many unlabeled text datasets, the pre-training can be as broad as possible. However, it continues to rely on models that are specific to a given task. The transformer-based models that are described below were chosen since it is currently challenging to create a better model architecture for each position, the improvement is therefore still incremental. These models can deal with context-related problems, but because they were developed using general domain corpora like Wikipedia, they can only be used for a limited number of tasks or domains. It has been hypothesized that domain-specific transformer-based models can enhance performance in subdomains. Some popular pre-trained language models are briefly introduced in the section that follows. Many NLP tasks use their pretrained embeddings as a first step toward a downstream job. The rest of this part provides a brief overview of Transformers before presenting the most common models also employed for TC challenges in Chapter 5.

#### RNN Encoder–Decoders

Sequence transduction methods have long been dominated by networks with RNN-like designs. Through the use of RNN-based encoder-decoder architectures and recurrent language models, which are advancements of traditional word embedding methods, researchers began pushing the limits of TC. To a better explanation of Transformers, it could be considered a translation task, where the sequence of input is a sentence in a source language and the output sequence is the translation of that sentence in another language. Each word in the input sequence is sent sequentially to the encoder, this has the effect of giving the model the new input word at time step $t$ and the hidden state at time step $t-1$. RNNs should theoretically be able to learn both long- and short-term associations between words because the input is used in steps and is dependent on the outcome of the previous step. The encoder's output is the "context" which is a compressed representation of the input sequence. After that, the decoder assesses the context and creates a brand-new set of words sequentially (for example, a translation into a different language), where each word is dependent on the results of the preceding time step. Contextually significant information (the context) is latently recorded while encoding and may afterward be utilized for tasks like TC. The primary drawback of this strategy is that the encoder must compress all pertinent data into a vector with a fixed-length. This was shown to be a problem, especially for longer phrases, and it was noted that as input sentence length increases, basic encoder-decoder performance rapidly degrades. Furthermore, because of their sequential nature, recurrent models have intrinsic restrictions. Parallelization is impossible due to sequentially, which results in more complex computations. As a result of the network's propensity to forget previous parts of the sequence, longer sentences are considered as the true bottleneck of RNNs and can cause memory problems (this is mainly due to the vanishing gradient issue). The attention mechanism was one approach used to overcome the drawbacks of recurrent architectures. One of the most significant turning points in the development of NLP was reached when this process eventually became a fundamental component of the Transformer architecture. In contrast to LSTM-based models, which showed little benefit from a significant increase in size, these designs' depth has actually been shown to be quite advantageous to their performance.

### The Attention Mechanism

Attention was designed to help the learning process focus on input phrases' more significant components by paying them "attention" when it was first utilized as an update to various architectures. As was indicated before, encoder-decoder designs based on RNNs have historically been used to solve seq2seq problems. These designs use stacked RNN layers for both the encoder and the decoder.

Bahdanau [18] presented the idea of attention to address this issue in neural machine translation tasks. The authors argued that the decoder can discriminate between input words and determine which of them are crucial for synthesizing the following target word by disseminating knowledge of the whole input sequence. The attention technique relies on the encoder hi's hidden state (also known as "annotation") improving the input context of each decoder unit, which contains data on the entire input sequence. "Additive attention" is the term used to explain the technique discussed here. Although there are numerous ways to incorporate the attention mechanism in seq2seq architectures, the goal is to create an alignment score that measures the relative importance of words in the input and output sequence. Even outside the realm of NLP, where attention first showed its worth, attentive artificial neural networks are now used in several applications. Hierarchical attention networks [195, 305] are novel examples of applications in the field of TC. These methods rely on paying attention at two levels: the word level when encoding document phrases and the sentence level when essentially encoding the significance of each sentence in respect to the intended sequence. But now, rather than being only an additional augmentation, attention is used as a solid foundation. This is the foundation of the Transformer design, which keeps a well-known encoder-decoder structure but does not employ recursion. Instead, dependencies between input and output are established only through the attention mechanism. Transformers have been demonstrated to produce superior outcomes while also gaining significantly more speed because they are highly parallelized.

### The Transformer Architecture

Vaswani [286] introduced the Transformer architecture, a cutting-edge encoder-decoder architecture that enables to handle all input tokens (such as words) simultaneously rather than sequentially. Transformers presents input sequences as a bag of tokens with no sense of order. The Transformer uses a mechanism known as "self-attention" to understand the relationships between tokens. Thanks to a particular encoding phase that is carried out before the first layer of the encoder, the embeddings for the same word that come in the phrase at a different location will also have a distinct representation. Positional encoding is the process that fills in the information that would otherwise be lost regarding the relative location of words. The self-attention layer, a crucial part of this architecture, naturally enables the encoder to scan other words in the input phrase as it processes one of its words. A multi-head attention layer is produced by stacking several layers of this kind. The head outputs are then concatenated, and the resulting output is then passed through a linear layer to combine the various outputs into a single matrix.

Multiple parallel iterations of these procedures are carried out by the Transformer multi-head self-attention layer. The goal of this is to broaden the range of representation sub-spaces which the model could concentrate on. The output of the attention heads is concatenated and routed via a linear layer to form the final representation, which condenses data from all the attention heads. This representation is then normalized, added to the residual input, and given to a feed-forward linear layer.

Transformers greatly improve TC and other NLP tasks by efficiently learning global semantic representation. It often uses unsupervised techniques to autonomously mine semantic knowledge, then builds pre-training targets to help machines understand semantics.

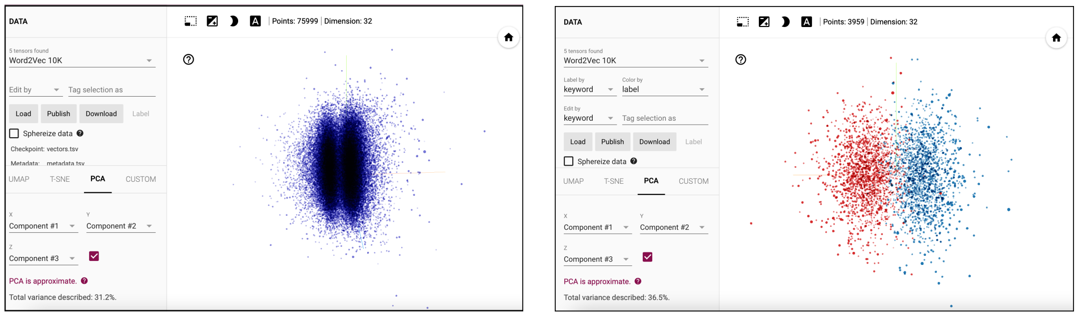
## 4.2 Analysis

In this section, I report the results of a case study to conduct an example of analysis of a word embedding trained from scratch. Thanks to the methodology proposed in this section, it is possible to better investigate the results and the behavior of a deep model trained on a specific dataset. The analysis presented here was conducted focusing on the FNS dataset to investigate the behavior of a simple CNN and its predictions on the test set after completing the training phase [256]. This further step can be employed in the TC pipeline to improve the performance of a model and for a better understanding of its behaviors.

In Chapter 2 I observe that keywords are good indicators to distinguish the two FNS and nFNS classes, as corroborated also by the results of the Bayesian model reported in Table 5.2. However, the CNN-based model must go beyond these frequency differences, as its results suggest. In this section, I provide a post-hoc analysis of the word embedding layer. Although hybrid approaches have been exploited to eXplainable AI [133], the CNN tested here can be defined as a shallow neural model. Thus, it can be analyzed mapping each layer outputs to its inputs.

### 4.2.1 A word embedding case study

After the training, I visualized in the embedding projector two clearly distinguishable clusters, as reported in Figure 4.2.1a (RQ3). To verify how these two clusters are related to the two classes, I labelled the words represented there. To do so, I extracted 3959 keywords using a Bayesian model—precisely, I extracted 1980 most frequent tokens in corpus 0 and 1979 most frequent tokens in corpus 1—and labelled them accordingly. Then, I visualized them in the embedding space of the trained CNN model, as shown in Figure 4.2.1b. Note that I used key tokens retrieved by the Bayesian model and not those obtained using Sketch Engine, because the former has the same tokenization of the CNN model. I excluded tokens occurring in both corpora 0 and 1. Figure 4.2.1b confirms


DATA

5 tensors found: Word2Vec 10K

Edit by | Tag selection as

Load | Publish | Download | Label

Sphereize data

Checkpoint: vectors.tsv

Metadata: metadata.tsv

UMAP | T-SNE | PCA | CUSTOM

X: Component #1 | Y: Component #2 | Z: Component #3

PCA is approximate.

Total variance described: 31.2%.

Points: 75999 | Dimension: 32

DATA

5 tensors found: Word2Vec 10K

Label by: keyword | Color by: label

Edit by: keyword | Tag selection as

Load | Publish | Download | Label

Sphereize data

UMAP | T-SNE | PCA | CUSTOM

X: Component #1 | Y: Component #2 | Z: Component #3

PCA is approximate.

Total variance described: 36.5%.

Points: 3959 | Dimension: 32

(**a**) (**b**)

Figure 4.6: Word embedding as visualized in a 3-dimensional space. (**a**) Unlabeled word embedding space (75,999 points). (**b**) Labelled word embedding space (3959 points).

that the two clouds are closely related to the two task classes. Red dots refer to FNS, blue dots to nFNS. Exploring these clouds, I can find some keywords also identified using Sketch Engine Keywords (Table 2.3). In Figure 4.2.1a,b, I highlighted *Unete*[1] as FNS keyword and *bulos*[2] as nFNS keyword. Apart from *Unete*, in Figure 4.2.1a, I can find other keywords individuated in the preliminary analysis conducted in Section 2.3. Of course, since Sketch Engine tokenization differs from that of the CNN model, there is not a one to one mapping. While, for example, following the standard tokenization in Sketch Engine I can distinguish cased and uncased letters, it is not the case with punctuation, which is always kept apart. In the embedding space, I can notice that the tokens with a higher keyness score are positioned farther than the other cluster (see, for example, *Unete* in Figure 4.2.1a). Thus, this could suggest that in the embedding space, tokens are located according to their keyness score.

## 4.2.2 Discussion

What emerges from this analysis, concerning the word embedding representation, is that the deep model involved (a shallow CNN) during the training phase is able to clearly subdivide the two vector spaces of the word vectors directly related with the two labels. It is worth noting that this deep model ability is highly dependent on the task. Indeed, when some authors are strongly characterized by a certain dictionary, the separability of the classes can already take place in the initial word embedding stage and not when performing the convolution in subsequent layers.

However, this separability is not always feasible when training from scratch a word embedding layer. As the task varies, the authors belonging to a class may not necessarily be characterized by certain keywords or, even worse, there may be an overlap between the point clouds present in the word embedding. Therefore, the methodology presented in this section could be useful to analyze the

[1]In English: *join up*.
[2]In English: *hoaxes*.

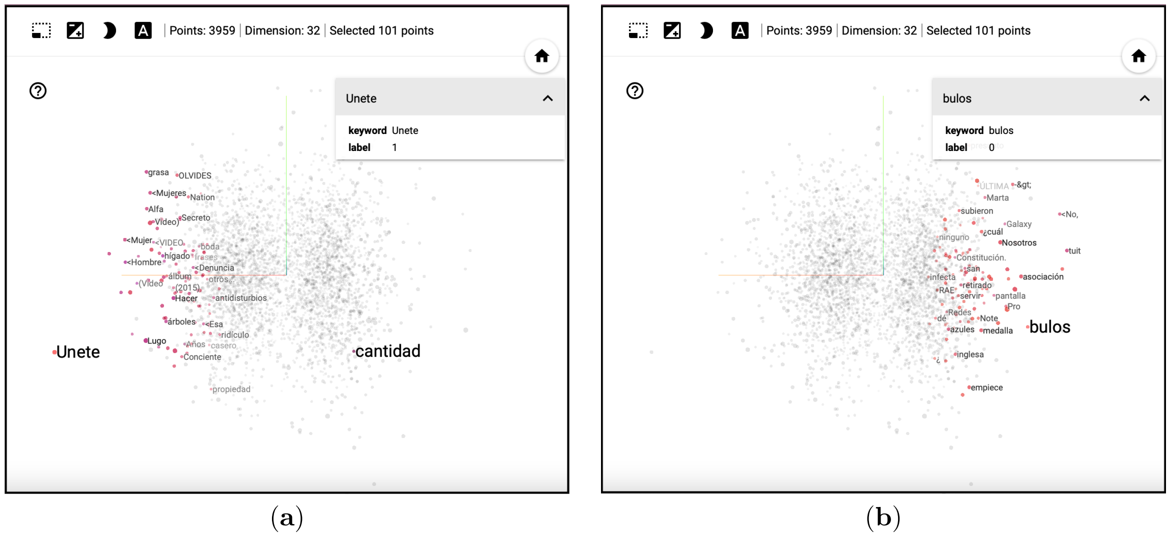


Figure 4.7: Visualization of FNS and nFNS keywords in the labelled embedding space. (**a**) Label 1. (**b**) Label 0.

embedding space after the model training and, based on the result, evaluate whether it is necessary to introduce further complexity into the model with successive layers to improve the classification performance.

# Chapter 5

# Classification

The TC is usually defined as the process of extracting features from the raw text data and categorizing the text data based on these features. Over the past few decades, a lot of TC models have been put forth. Models discussed in this chapter belong to three different classes. The first class includes Non-Deep Learning (NDL) deterministic models, the second class comprises not pre-trained Deep Learning (DL) models, the third class includes large pre-trained Language Models (LMs) known as Transformers.

The preferred method for TC up until recently was NDL models. These techniques usually use general-purpose classifiers that are not tailored to this situation when it comes to the actual classification algorithms. The steps in the TC pipeline (Figure 1.1) before extracting machine-interpretable characteristics and representations from texts are partially "offloaded" from the unique challenges posed by textual data (i.e., text interpretation). One of the original models used for TC tasks was NB. In the following section, general classification models are suggested. These models, which include KNN, SVM, Logistic Regression, and Random Forest (RF), are frequently used to classify texts. Recently, it is debatable if the Light Gradient Boosting Machine (LightGBM) and the Extreme Gradient Boosting (XGBoost) will deliver outstanding performance.

In order to address the TC issue for DL models, a Convolutional Neural Network (CNN) model has been introduced in [139]. I also take artificial neural networks, RNNs, and bidirectional LSTMs into consideration.

Although it wasn't created with TC tasks in mind, the Bidirectional Encoder Representation from Transformers (BERT) and other Transformer-based architectures have been extensively used when creating TC models due to their success on a variety of TC datasets. Also, others language models have been employed on several TC tasks as classifiers. I have already provided some background on Transformers in the Chapter 4 which is more related to the original scope of any Transformer architecture. Here, I present some of the most common architectures employed for TC.

Then I present relevant SOTA results of several models and the corresponding result on some datasets discussed in Chapter 2.

## 5.1 Classification models

### 5.1.1 NDL classifiers

Traditional NDL models accelerate TC without any initial pre-training, achieving relevant results on a plethora of TC tasks. In any NDL models, the first step involves to preprocess input text, with techniques like removing stop words, removing noise and unwanted characters/strings (see Chapter 3). Then a representation model is chosen to convert text data into a numerical representation as discussed in Chapter 4.

The NDL classification algorithms are briefly described in this section. These methods are based on generic classification approaches; as was mentioned, careful data pre-processing and feature engineering are stressed in order to obtain competitive results.

#### Logistic Regression

Logistic regression (LR)[93] is one of the earliest classification techniques worth mentioning. By attempting to determine which properties are most helpful to distinguish cases, the linear classifier LR attempts to forecast probabilities over classes. Its basic formulation is most effective for binary classification tasks, but it may be extended to the multinomial situation by using a formulation that typically incorporates the softmax function or by building an ensemble of several binary classifiers using a one-vs.-rest strategy.

Linear classifiers as LR are good for large and high-dimensional datasets. It has been proven to outperform traditional back-off smoothing, because the former has the ability to process unknown terms and also avoids over evaluating the conditional probability which is originally zero. Ridge logistic regression is a popular solution to TC problem, however its role in large scale documents is still questionable. To eliminate this difficulty, sparse solution is combined with ridge regression. The sparsification removes less important features, thereby solving the classical problem of ridge regressors[223].

LR is commonly employed in TC for several tasks [254]. 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 [262]. Using the *sklearn* is available online a Logistic Regression implementation[1]. A common solver is *lbfgs* and is discussed in [47].

### Naïve Bayes

Because of how straightforward their structure and calculation are, models like Naïve Bayes (NB) are particularly well-liked. The assumption of independence, which states that no feature affects any other features, is what gives this system its simplicity. The NB method's core idea is to use the prior probability of a class given the features—as seen in the training set—to determine its posterior probability.

NB classifiers are derived from Bayes theorem, which states that given the number of documents $n$ to be classified into $z$ classes where $z \in \{x_1, x_2, ...., x_z\}$ the predicted label out is $x \in X$. The Bayes theorem, which asserts that the predicted label out is $x \in X$, is the foundation for NB classifiers. Given the number of documents $n$ to be categorized into $z$ classes, where $z \in \{x_1, x_2, ...., x_z\}$, the expected label out is $x \ in \ X$. This is how the NB theorem is formulated:

$$P(x|y) = P(x)\frac{P(y|x)}{P(y)} \tag{5.1}$$

Where $y$ stands for a document and $x$ stands for the classes. The NB algorithm will, to put it simply, compute the likelihood that each word in the training data will be classified. Once each word's probability has been determined, the classifier is next instructed to categorize fresh data using the probabilities that had already been determined during the training phase.

The NB approach is straightforward, and the parameters are more minuscule and less vulnerable to missing data. The assumption is that features are independent of one another. The performance of NB declines when the number of features is high or when there is a strong connection between the features. The NB method makes an independent assumption that the conditions between texts are independent once the target value has been provided. To get the posterior probability, the NB method largely uses the prior probability. NB is widely used for TC task because of its straightforward nature. Even if it is occasionally incorrect to assume that the characteristics are independent, doing so greatly simplifies calculations and improves performance.

NB for TC has been utilized for document classification tasks on a large scale since the 1950s, according to [230]. Thomas Bayes developed the Bayes theorem, which serves as the theoretical foundation for the NB classifier approach. This method of information retrieval has received a lot of attention in recent studies [235]. This method of TC uses generative models, which are the most often used approach. The simplest form of NB simply counts the words in documents. The use of the NB classifier can also be considered as a modern TC application because it is employed in the identification of fake news [95] and sentiment analysis[204]. Three well-liked NB TC methods are Bernoulli NB, Gaussian NB, and Multinomial NB.

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

As reported in [189] and experimentally demonstrated over time by outcomes from various TC tasks [240], NB is one of the most effective model to employ for classification. A popular multinomial NB classifier from *sklearn* is the MultinomialNB implementation[2]. When dealing with multinomial distributed data, MultinomialNB implements the NB method. Data are commonly expressed as word vector counts.

### K-NN-Based Classification

By locating the k-most comparable labelled instances and, in its most basic iteration, assigning the most prevalent category to the unlabeled instance being classed, TC based on K-Nearest Neighbors (k-NN) algorithms[61] approaches the problem differently.

Instead of using a discriminating class domain to establish the category, KNN mostly relies on the nearby finite neighboring samples. So it is better suited than other approaches to split the dataset with greater crossing or overlap of the class domain. The KNN algorithm locates the k documents in the training set that are closest to a test document called x, and then ranks the category choices based on the k neighbors' classification. The score of the category of the neighbor documents may depend on how closely x resembles each neighboring document. Multiple KNN documents may fall under the same category; in this case, the similarity score of class k with regard to the test document x would be calculated by adding these scores. The candidate is assigned to the class with the highest score from the test document x after the score values have been sorted.

On the large-scale datasets, the KNN approach, however, requires an abnormally long time because of the positive association between model time/space complexity and the volume of data ([118]). Scholars in [266] suggest a KNN technique without feature weighting to reduce the amount of selected features. By employing a feature selection, it is able to identify pertinent features and create word interdependencies. KNN typically classifies samples with more data when the distribution of the data is extremely asymmetric. To enhance classification performance on the unbalanced corpora, the Neighbor-Weighted K-Nearest Neighbor (NWKNN) [278] is presented. It gives neighbors in a narrow class a large weight and neighbors in a wide class a little weight.

### Decision Tree

Decision Tree (DT) development and presentation were done in [236] and in [182] respectively. It is one of the oldest classification models for text and data mining, and it is successfully used for classification tasks in a variety of fields. This concept was primarily motivated by the desire to build tree-based attributes for data points, but the key question is which feature will be a child's level and which would be a parent feature. The DT classifier design has a root, decision, and leaf node that represent the dataset, execute computation, and carry out classification, respectively. The classifier

[2]`https://scikit-learn.org/stable/modules/generated/sklearn.Nave_bayes.MultinomialNB.html`

learns the judgment that must be made in order to divide labelled groups during the training phase. The data is processed through the tree in order to categorize the unclassified instance. A specific property of the incoming text is compared to a fixed that was previously learned during the training phase. The choice is based on whether the chosen feature is more prominent than or less prominent than the fixed feature, which divides the tree into two parts. This comparison is made at each decision node. The text will finally traverse these decision nodes and arrive at the leaf node that describes the class to which it has been allocated. The benefits of the DT classifier include the nearly non-existent number of hyperparameters that need tuning, its simplicity in description, and the ease with which its visualizations can be understood. On the other hand, the DT classifier has some significant drawbacks, including the risk of overfitting, sensitivity to small changes in the data, and difficulties with prediction outside of samples.

The DT method produces simple classification rules, and the pruning technique [241] can also assist lessen the impact of noise. Its fundamental weakness, however, is from its inability to effectively handle datasets with rapidly growing sizes. Information gain is explicitly used by the Iterative Dichotomiser 3 (ID3) algorithm [236] as the attribute selection criterion in the selection of each node. It's utilized to choose the attribute for each branch node before choosing the one with the greatest information gain value to serve as the discriminant attribute for the current node.

An DT-based symbolic rule system is proposed by the author in [124]. The approach converts each text into a vector based on the frequency of each word, and it then generates rules based on training data. The classification of the additional data, which resembles the training data, is done using the learning rules. Fast Decision-Tree (FDT) [287] also employs a two-pronged approach to lower the computational costs of DT algorithms: pre-selecting a feature set and training multiple DTs on various data subsets. To address the issues of imbalanced classes, the results from different DTs are integrated using a data-fusion technique.

An DT-based symbolic rule system is proposed by the author in [124]. The approach converts each text into a vector based on the frequency of each word, and it then generates rules based on training data. The classification of the additional data, which resembles the training data, is done using the learning rules. Fast Decision-Tree (FDT) [287] also employs a two-pronged approach to lower the computational costs of DT algorithms: pre-selecting a feature set and training multiple DTs on various data subsets. To address the issues of imbalanced classes, the results from different DTs are integrated using a data-fusion technique.

#### Random Forest

Random Forest (RF), also known as an ensemble learning methodology, focuses on ways to compare the outcomes of multiple trained models in order to provide a better classifier and performance than a single model. A proposed RF classifier that is easy to learn and produces better classification outcomes is described in [108]. A bootstrapped subset of the training text is used to train each tree in the number of DT classifiers that make up the RF classifier. At each decision node, a random subset of the characteristics is chosen, and the model only looks at a portion of these attributes. The main problem with using a single tree is that it has a lot of variety, which makes it susceptible to the effects of how the training data and feature arrangements are organized. Although this classifier is rapid to train on textual data, Bansal [22] found that it is slow to make predictions after training. It performs well with both categorical and continuous data, can handle missing values automatically, is robust to outliers, and is less impacted by noise than training numerous trees, which can be computationally expensive, take a long time to train, and use up a lot of memory.

**Support Vector Machines (SVMs)**

Authors in [59] suggest the Support Vector Machine (SVM) to deal with the binary classification of pattern recognition. For the first time, authors in [121] represent each text as a vector, employing the SVM algorithm for TC. The TC challenges are divided into numerous binary classification tasks using SVM-based methods. By maximizing the distance between the hyperplane and the two categories of training sets, SVM generates an ideal hyperplane in the one-dimensional input space or feature space in this situation, resulting in the best generalization ability. The objective is to maximize the distance along the category boundary that is perpendicular to the hyperplane. In other words, this will produce the lowest categorization error rate. To arrive at a globally optimal solution, the challenge of building an optimal hyperplane can be turned into a quadratic programming problem. To ensure that SVM can handle nonlinear problems and develop into a reliable nonlinear classifier, it is crucial to select the right kernel function [161, 275]. To further reduce the labelling effort based on the supervised learning algorithm SVM, active learning [164] and adaptive learning [221] methods are employed for TC. Joachims [123] suggests a theoretical learning model combining the statistical traits with the generalization performance of an SVM, analyzing the features and benefits using a quantitative approach. This analysis examines what the SVM algorithms learn and what tasks are suitable. A universal decision function that takes into account a particular test set is presented to be used with the Transductive Support Vector Machine (TSVM) [122] to reduce misclassifications of specific test collections. It establishes a better framework and studies more quickly by using existing knowledge.

SVMs extend to multidimensional, non-linear classification by projecting their inputs to a higher-dimensional space in order to potentially be better able to distinguish training categories. The kernel trick is the name of the process because the function that maps to this higher-dimensional space is known as a kernel function. The key to getting good performance is choosing the proper form and parameters.

As reported in [58] and in [174], classifiers based on SVM are well-established methods for TC tasks. SVM are also employed in ensemble-based text classifier, as reported in [63]. Thanks to SVM models, classification results compared to other classification methods have been greatly improved. Based on [52], is available online the *sklearn* SVC implementation[3].

### 5.1.2 DL classifiers

The Artificial Neural Networks (ANN) that make up the DL classifiers mimic the human brain to automatically learn high-level features from data, outperforming conventional models in speech recognition, picture processing, and text understanding. To categorize the data, input datasets like a single-label, multi-label, unsupervised, imbalanced dataset should be examined. The input word vectors are delivered into the ANN for training in accordance with the trait of the dataset up until the termination condition is met. The downstream tasks, such as sentiment categorization, question answering, and event prediction, provide as proof of the training model's effectiveness. In the recent decades, a large number of deep learning models for TC have been suggested. The first two deep learning methods for the TC task that outperform conventional models are the multilayer perceptron and the recursive neural network. Then, for text categorization, CNNs, RNNs, and attention processes are applied. Many researchers enhance CNN, RNN, and attention, or model fusion and multitask approaches, to improve TC performance for various tasks. Text categorization and other NLP methods have advanced significantly with the introduction of BERT, which can produce contextualized word vectors. It has been found that TC models based on BERT perform better than the models mentioned above in a variety of NLP tasks, including TC. Additionally, Graph Neural Network (GNN)-based TC technology is being studied by certain academics in order to collect structural information in the text that cannot be captured by alternative techniques. Except the attention-based models, I go into detail below about a few exemplary models. For a detailed discussion on attention-based models, please refer to Chapter 4.

#### Artificial neural network

The gap between shallow and deep methodologies is bridged by straightforward structures like Multilayer Perceptrons (MLPs) or Artificial Neural Networks (ANN). These neural network designs are among the most fundamental, but they serve as the cornerstone for the first word embedding methods and produce great results when used as standalone classifiers. These MLP models often approach input text as an unordered BoW, with each input word being represented by a different feature extraction method (like TF-IDF or word embeddings).

ANN see text as a collection of BoW. They first use an embedding model, such as Word2Vec [197] or Glove [222], to learn a vector representation for each word. They then use the vector sum or average of the embeddings as the representation of the text, pass it through one or more feed-forward layers known as Multi-Layer Perceptrons (MLPs), and perform classification on the representation

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

of the final layer using a classifier, such as The Deep Average Network (DAN)[116] that is one of these models.

DAN performs better than other more complex models that are intended to explicitly learn the compositionality of texts, despite their simplicity. On datasets with large syntactic variance, DAN, for instance, performs better than syntactic models. A straightforward and effective text classifier named fastText is proposed by the authors of [128]. FastText sees text as a collection of words, much like DAN. FastText, unlike DAN, uses a bag of n-grams as extra features to record local word order data. In practice, this proves to be quite effective, producing outcomes that are comparable to those obtained by methods that explicitly employ the words order [294].

Additionally, the authors of [157] propose doc2vec, which use an unsupervised approach to train fixed-length feature representations of variable-length textual units like sentences, paragraphs, and documents. Doc2vec's architecture resembles that of the CBOW model. The extra paragraph token that is via matrix converted to a paragraph vector is the only difference. To forecast the fourth word in doc2vec, this vector's concatenation or average with a context of three words is employed. The paragraph vector serves as a placeholder for context-missing data and can serve as a reminder of the paragraph's subject. After training, the paragraph vector is sent to a classifier for prediction and utilized as features for the paragraph (for example, in place of or in addition to BoW). When Doc2vec is released, it produces brand-new SOTA outcomes on a number of TC tasks.

### Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [232]—which are designed to get word relationships and text structures for TC—view text as a series of words. Pure RNN models, on the other hand, frequently perform worse than feed-forward neural networks. Long Short-Term Memory (LSTM) is the most often used RNN variation, since it is intended to better capture long-term dependency. By incorporating a memory cell to retain values over virtually any time period and three gates (input gate, output gate, forget gate) to control the flow of data into and out of the cell, LSTM solves the gradient disappearing or exploding issues that plagued vanilla RNNs. There have been efforts to make RNNs and LSTM models for TC better by capturing additional data, such as natural language tree structures, long-span word relations in text, document topics, and so forth. The authors of [211] describe how to conduct TC using LSTM networks and various variations, such as BiLSTM and GRU. Additionally, authors who employ a BiLSTM in [258] do so with noteworthy outcomes. Two bidirectional LSTM layers make up the model.

The authors of [274] develop a Tree-LSTM model, a generalization of LSTM to tree-structured network typologies, to learn complicated semantic representations. Because natural language possesses syntactic characteristics that would naturally join words to form phrases, the authors contend that Tree-LSTM is a more effective model for NLP tasks than the chain-structured LSTM. On the two tasks of sentiment classification and predicting the semantic similarity of two sentences, they validate the efficiency of Tree-LSTM. The chain-structured LSTM is also extended to tree structures by the authors of [315], using a memory cell to preserve the history of numerous child cells or numerous descendant cells in a recursive process. The new model, they contend, offers a systematic approach to thinking about long-distance communication over hierarchies, such as language or picture parse structures. The LSTM architecture is supplemented in [55] with a memory network in place of a single memory cell in order to model long-span word relations for machine reading. With brain attention, this permits adaptive memory use during recurrence and provides a method for weakly inducing relationships between tokens. In terms of language modelling, sentiment analysis, and NLI, this model yields encouraging results. By capturing important information with various timescales, the Multi-Timescale LSTM (MT-LSTM) neural network, which is described in [170], is also intended to model extended texts, such as sentences and papers. A typical LSTM model's hidden states are divided into many categories by MT-LSTM. At various times, each group is updated and activated. MT-LSTM can therefore model extremely long documents. On TC, MT-LSTM is said to perform better than a number of baselines, including models based on LSTM and RNN. RNNs have trouble remembering long-distance dependencies, but they do a decent job of capturing the local structure of a word sequence. Contrarily, word ordering is not taken into account by latent topic models, which can only represent the overall semantic structure of a document. The authors of [74] suggest a TopicRNN model to combine the advantages of latent topic models and RNNs. It uses latent topics to capture global (semantic) dependencies, while employing RNNs to capture local (syntactic) dependencies. According to reports, TopicRNN performs better in sentiment analysis than RNN baselines. Other intriguing RNN-based models exist. The authors of [171] train RNNs to utilize labelled training data from numerous related tasks by utilizing multitask learning. The authors of [125] investigate an LSTM-based text region embedding technique. Authors in [314] present a novel architecture that combines a BiLSTM model with two-dimensional max-pooling to capture text features. A bilateral multi-perspective matching model is put out in [295] inside the "matching-aggregation" framework. A BiLSTM model is used by the authors of [292] to investigate semantic matching utilizing various positional sentence representations. It is crucial to remember that RNNs are a subset of DNNs. A recursive neural network continually applies the same set of weights over a structural input to create a structured prediction or a vector representation over inputs of varying sizes. Recursive neural networks (RNNs) are recursive neural networks with a linear chain structure input, whereas recursive neural networks with a hierarchical structure input, such as parse trees of English language sentences ([265]), can operate on hierarchical structures by integrating child representations into parent representations. RNNs are the most popular recursive neural networks for TC because of their effectiveness and ease of use.

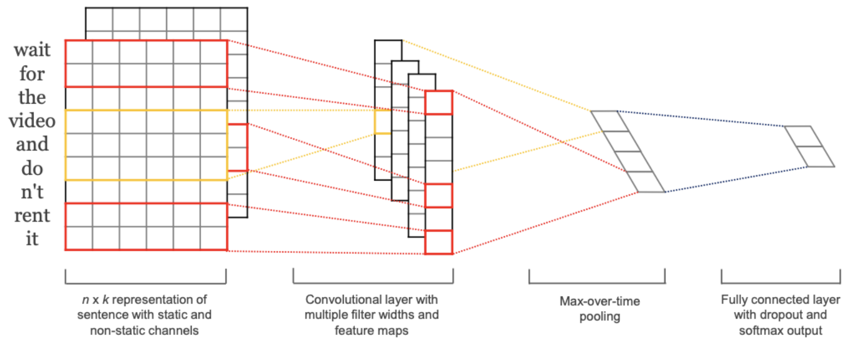

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

**Figure 1: Model architecture with two channels for an example sentence.**

Figure 5.1: The original image of the CNN architecture proposed in [139].

### Convolutional Neural Networks

Computer vision applications are frequently linked with Convolutional Neural Networks (CNNs). CNNs are employed for classifying images using convolving filters that can extract picture characteristics. However, they have also been used, especially in the context of NLP and TC. In [139], one of the earliest attempts to use a CNN for sentiment analysis is covered. Figure 5.1 shows the original network structure. The author describes a series of experiments using a CNN trained for sentence-level classification tasks on top of pre-trained word vectors. The author demonstrates that a straightforward CNN with little hyperparameter adjustment and static vectors performs admirably on a variety of benchmarks. Additional performance benefits can be obtained by learning task-specific vectors through fine-tuning. In order to support the use of both task-specific and static vectors, the author also suggests a straightforward change to the architecture. The CNN models mentioned here outperform the current state of the art on 4 of the 7 tasks, including sentiment analysis and question classification.

The CNN architecture used in [256] to identify FNS on Twitter is displayed in Figure 5.2. The input text's word vectors are first combined into a word embeddings matrix. The convolutional layer, which has multiple filters with various dimensions, feds the matrix. The output of the convolutional layers is then passed through the pooling layer and concatenated to create the final vector representation of the text for two additional pairs of conv-pool layers. The last vector predicts the category. To avoid overfitting, certain dropout layers are placed between layers.

Examining their input, which likewise uses word embeddings, is the simplest way to comprehend these methods. RNNs typically input a sentence's words in order, but CNNs provide sentences as a matrix, with each row representing an embedding of a word (therefore, the number of columns corresponds to the size of the embeddings). Contrary to RNN, CNN can apply convolutions defined by many kernels to numerous chunks of a sequence at once. In contrast, convolutional filters often

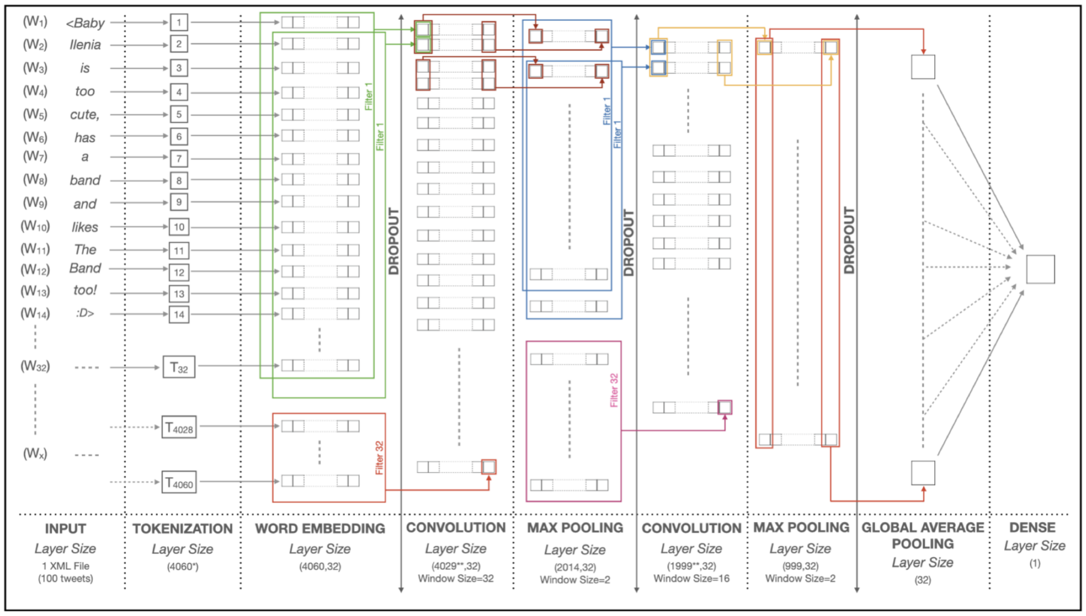


Figure 5.2: The architecture of the CNN used proposed in [256].

glide over local portions of an image in two directions in image-based tasks. Instead, filters in text-related tasks are typically made to be as wide as the embedding size, ensuring that this operation only proceeds in ways that make sense from a sentence-level perspective while always taking the full embedding for each word into account. In general, the speed and effectiveness of CNNs' latent representations are considered to be their key benefits. On the other hand, when analyzing text, other features that could be used while working with images, like location invariance and local compositionality, make little sense.

Other interesting applications based on CNN are discussed in [259] and also used in [184]. Such CNNs consist essentially of a single convolutional layer. As demonstrated by its results, these CNNs outperforms Transformers and others proposed models as stated in [239].

## Capsule Neural Networks

CNNs use pooling and successive layers of convolution to categorize images and words. Convolution procedures lose information on spatial relationships and are more likely to misclassify things based on their orientation or proportion, despite the fact that pooling operations identify important features and simplify computation. Hinton et al. suggest a novel strategy known as capsule networks (CapsNets)[106] to overcome the issues with pooling. A capsule is a collection of neurons that represent various properties of a certain kind of thing, such as an object or an object component, by their activity vector. The length of the vector denotes the likelihood that the entity exists, while the vector's orientation denotes the characteristics of the entity. In contrast to max-pooling of CNNs, which chooses some information and discards the rest, capsules use all the data present in the network up to the final layer for classification by "routing" each lower-layer capsule to its ideal parent capsule in the higher layer. Different methods, such as dynamic routing-by-agreement [247] or the EM algorithm [107] can be used to implement routing.

Capsule networks, which have recently been used in TC, modify capsules to represent a sentence or document as a vector. The authors of [302] suggest a TC model built on a CapsNets variation. An n-gram convolutional layer, a capsule layer, a convolutional capsule layer, and a fully linked capsule layer make up the model's four layers. The authors test three methods for stabilizing the dynamic routing process in order to reduce the disruption caused by noise capsules that contain background data such stop words or words that are unrelated to any document categories. Additionally, they investigate Capsule-A and Capsule-B, two capsule structures. Similar to the CapsNet in [247] is Capsule-A. To learn a more thorough text representation, Capsule-B employs three parallel networks with filters of various window sizes in the n-gram convolutional layer. In the tests, CapsNet-B performs better.

### Graph Neural Networks

TextRank [196] is one of the earliest graph-based models created for NLP. The authors suggest modelling a natural language document as a node-and-edge graph. Nodes can represent text units of various types, such as words or complete sentences, depending on the applications at hand. Similar to how lexical or semantic relationships, contextual overlap, etc. may all be represented using edges, so can other forms of relationships between any nodes. DL methods for graph data, like the text graphs utilized by TextRank, are extended to create modern Graph Neural Networks (GNNs). Over the past few years, DNN, including CNNs, RNNs, and autoencoders, have been adapted to handle the complexity of graph data. In order to perform graph convolutions, for instance, a 2D convolution of CNNs for image processing is generalized by taking the weighted average of a node's neighborhood information. Convolutional GNNs, such Graph Convolutional Networks (GCNs) [142] and its derivatives, are the most often used among the numerous types of GNNs because they perform well and are simple to combine with other neural networks, and they obtain SOTA results in many applications. An effective CNN variation for graphs is the GCN. To learn graph representations, GCNs stack layers of previously learned first-order spectrum filters and then apply a nonlinear activation function. TC is a common GNN application in NLP. GNNs use the relationships between words or documents to infer the labels of documents. Authors in [219] propose a graph-CNN based DL model that first transforms text into a graph of words before using graph convolution procedures to convolve the word graph. They demonstrate through tests that CNN models have the benefit of learning several levels of semantics, whereas the graph-of-words representation of texts has the advantage of capturing non-consecutive and long-distance semantics. Authors suggest a TC model in [220] that is built on hierarchical taxonomy-aware and attentional graph capsule CNNs. The usage of the hierarchical relationships among the class labels, which in earlier methods are deemed independent, is one of the model's distinctive characteristics. In particular, the authors create a novel weighted margin loss by taking into account the label representation similarity in order to exploit such relations. They also construct a hierarchical taxonomy embedding approach to train their representations. Similar Graph CNN (GCNN) model is used for TC in the method in [306]. They create a single text graph for a corpus based on word co-occurrence and document word relations, and then train a Text Graph Convolutional Network (Text GCN) for the corpus. The Text GCN learns the word and document embeddings jointly under the supervision of the known class labels for documents, after initializing with a one-hot representation of each.

### 5.1.3 Transformers

Also in TC, the attention-based techniques are successfully applied. The model can pay varying attention to different inputs thanks to the attention mechanism. It first groups necessary words into sentence vectors, and then groups necessary sentence vectors into text vectors. Through the two levels of attention, it can determine the relative contributions of each word and sentence to the classification judgment, which is useful for applications and analysis. The popularity of the attention mechanism stems from its potential to enhance TC performance with interpretability. The remainder of this section introduces a few of the most well-known Transformer architectures that are also employed for a number of TC applications.

#### Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a method for fine-tuning for individual tasks without creating bespoke network architectures by first training a large language model on free text. The contextualized word representation language model is presented in [73] and uses parallel attention layers rather than sequential recurrence in the transformer. BERT is trained with two tasks in place of the fundamental language task to promote bidirectional prediction and sentence-level comprehension. BERT is trained on two unsupervised objectives: (1) a Masked Language Model (MLM) task, in which 15% of the tokens are randomly masked (i.e., replaced with the "[MASK]" token), and the model is trained to predict the masked tokens; and (2) a Next Sentence Prediction (NSP) task, in which the model is given a pair of sentences and trained to determine when the second one follows the first. The purpose of this second assignment is to gather more practical or long-term data. English Wikipedia text passages and the dataset of Books Corpus are used in BERT training. The BERT-Base and BERT-Large pre-trained models are both available. BERT can be used to unannotated data as well as fine-tuned task-specific data directly from the trained model. Online resources include both the fine-tuning code and the publicly available pre-trained model.

### GPT2

In 2019, the OpenAI team published GPT2 [237], a scaled-up version of GPT. In terms of the location of layer normalization and residual relations, it adds a few minor enhancements over the previous version. There are actually four different GPT2 variants, the smallest of which is identical to GPT, the medium of which is comparable to BERT Large, and the xlarge of which was produced with 1.5B parameters, which is the actual GPT2 standard.

### RoBERTa

Authors in [173], 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.

### ALBERT

Despite its success, BERT has some drawbacks, such as its enormous amount of parameters, which leads to concerns with pre-training time degradation, memory management challenges and model degradation. These problems are extremely effectively addressed by ALBERT, which Lan proposed in [155] and updated based on the BERT architecture. ALBERT uses two-parameter reduction techniques to scale pre-trained models, removing the crucial obstacles. The large vocabulary embedding matrix is divided into two smaller matrices using factorized embedding parameterization, NSP loss is replaced with SOP loss, and cross-layer parameter sharing prevents the parameter from increasing with network depth. When compared to BERT, these techniques considerably reduce the amount of parameters utilized while having little to no impact on the model's performance, enhancing parameter efficiency. As BERT large has 18 times fewer parameters and can be trained roughly 1.7 times faster, an ALBERT configuration is the same as that. Despite having fewer parameters than BERT, ALBERT produces novel SOTA outcomes.

### DistilBERT

A lighter version of BERT based on a transformer (i.e., DistilBERT), requires a quicker model to train being a more compact general-purpose language representation model. DistilBERT shrinks the original BERT model by 40% while keeping 97% of its language understanding skills and increasing speed by 60%. If BERT can be seen as the instructor in the process of knowledge distillation, DistilBERT is the pupil. A little model that represents the student is trained to mimic the behavior of the larger model (i.e., the teacher). Such a compact model is trained with a linear combination of three losses: the *distillation loss* (i.e., $L_{ce}$), the *masked language modelling loss* (i.e., $L_{mlm}$), and the *cosine embedding loss* (i.e., $L_{cos}$). Because of the distilled nature of the model, training and fine-tuning on a specific dataset for a specific task is of prominent importance. Refer to [250] for a thorough description of DistilBERT.

#### XLNet

A generalized autoregressive pretraining strategy is the one suggested in [304]. By optimizing the predicted likelihood across all combinations of the factorization order, it enables learning bidirectional contexts. BERT is surpassed by XLNet, often with a relevant margin, on a number of tasks, including question answering, sentiment analysis, document ranking and NLI. A popular implementation is the pre-trained XLNet using zero-shot [53].

#### Text-to-Text Transfer Transformer (T5)

By converting the data to text-to-text format and using an encoder-decoder framework, unified NLU and generation is possible. The T5 pre-training corpus has been developed, and it also comprehensively contrasts previously presented methodologies, in terms of pre-training aims, architectures, pre-training datasets, and transfer mechanisms. T5 employs a pre-training for multitasking and a text infilling objective. T5 employs the decoder's token vocabulary as the prediction labels for fine-tuning.

#### ELECTRA

According to what stated in [57], 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 GNN, ELECTRA can also be employed as an embedding layer, as in [176].

### 5.1.4 Hybrid and others approaches

#### Hybrid approaches

To capture local and global aspects of sentences and documents, many hybrid models that incorporate LSTM and CNN architectures have been developed.

A CNN-RNN model that can capture both global and local textual semantics and, consequently, represent high-order label correlations while having a manageable computational complexity is used by Chen et al. [54] to perform multi-label TC.

A Convolutional LSTM (C-LSTM) network is suggested by Zhu [316]. In order to create the sentence representation, C-LSTM uses a CNN to extract a series of higher-level phrase (n-gram) representations. For document modelling, Zhang et al. [313] suggest using a Dependency Sensitive CNN (DSCNN). The sentence vectors learned by the LSTM in the hierarchical DSCNN model are then supplied to the convolution and max-pooling layers to produce the document representation.

Xiao et al.[300] recommend using character-based convolution and recurrent layers for document encoding, since they see a document as a series of characters rather than words. When compared to word-level models, my model produced equivalent results with a lot less parameters.

Kowsari et al. suggest a Hierarchical Deep Learning method for TC in [147]. At every level of the document hierarchy, HDLTex uses stacks of hybrid DL model architectures, such as MLP, RNN, and CNN, to give specialized knowledge.

A reliable Stochastic Answer Network (SAN) for multistep reasoning in machine reading comprehension is proposed by Liu [172]. Memory networks, Transformers, BiLSTM, attention networks, and CNN are just a few of the neural network types that are combined in SAN. The context representations for the questions and passages are obtained via the BiLSTM component. A passage representation that is question-aware is derived by its attention mechanism. A second LSTM is then employed to create a working memory for the section. A Gated Recurrent Unit (GRU) based answer module then generates predictions.

For language modelling, Kim et al. [140] use a highway network with CNN and LSTM over characters. A character embedding lookup is done in the first layer, followed by convolution and max-pooling operations to create a fixed-dimensional representation of the word that is then transferred to the highway network. The output of the highway network serves as the input for a multi-layer LSTM. To extract the distribution across the following word, an affine transformation and a softmax are then applied to the LSTM's hidden representation.

### Other approaches

The twin neural network is another name for the siamese neural network [56]. It works in tandem with two different input vectors and uses equal weights to produce equivalent output vectors. A siamese adaptation of the LSTM network made up of pairs of variable-length sequences is presented by Mueller [205]. The model, which outperforms ANN of higher complexity and painstakingly created features, is used to estimate the semantic similarity between texts. The model also encodes text using neural networks with word vectors as inputs that were separately learned from a sizable dataset.

Deep learning techniques call for numerous additional hyperparameters, which raises the computational difficulty. In semi-supervised tasks, Virtual Adversarial Training (VAT) [201] regularization based on local distributional smoothness can be employed. It simply needs a few hyperparameters and can be directly read as robust optimization. Miyato uses VAT to significantly enhance the model's robustness, generalizability, and word embedding performance.

By increasing the total number of rewards received, Reinforcement Learning (RL) learns the best course of action in a particular situation. Zhang [310] provide an RL strategy for creating organized sentence representations by teaching the structures relevant to tasks. The model includes representation models for Hierarchical Structured LSTM (HS-LSTM) and Information Distilled LSTM (ID-LSTM). The HS-LSTM is a two-level LSTM for modelling sentence representation, and the ID-LSTM learns the sentence representation by selecting keywords that are pertinent to tasks.

Memory networks[66] develop the capacity to integrate the long-term memory and inference components. Li [165], who uses two LSTMs with extended memories and neural memory operations to manage the extraction duties of aspects and opinions at once. Latent topic representations indicative of class labels are encoded using Topic Memory Networks (TMN) [309], an end-to-end model.

Common-sense acquired outside the country. Authors in [75] believe that the event extracted from the original text lacked common knowledge, such as the goal and emotion of the event participants, because there was not enough information about the event itself to identify it for the EP task. The model enhances the effectiveness of stock forecasting, EP, and other factors.

The words and their relationships to one another are represented in the quantum language model by fundamental quantum events. In order to learn both the semantic and the sentiment information of subjective writing, Zhang et al. [312] propose a sentiment representation approach that is quantum-inspired. The model performs better when density matrices are added to the embedding layer.

Notable mention should also be made of integration-based (or ensemble learning) methods, which combine the output of various algorithms to improve performance and interpretation. These contain a number of subcategories, with bagging and boosting being the most well-liked ones. Bagging[40] (also known as bootstrap aggregation methods) averages the results of many classifiers without strong dependencies by training each of them separately on a part of the training data (sampling with replacement). Random forests are the most prevalent example of such a method, which increases accuracy and stability.

## 5.2 SOTA Results

Some results on the author profiling datasets discussed in Chapter 2 are shown in Table 5.2. Regarding the first three best results on the FNS dataset, the first best approach [44] make use of an LR ensemble based on five sub-models (n-grams with Logistic Regression, n-grams with SVM, n-grams with Random Forest, n-grams with XGBoost and XGBoost), the second-best approach employs combinations of character and word n-grams and SVM [229]. The third one [144] used Logistic Regression and Support Vector Machines, depending on the language, with combinations of character and word n-grams.

In addition, in [256] other models are compared on the same dataset. In Figure 5.2 is reported the average accuracy of each model evaluated. AVG accuracy is calculated averaging the accuracy in each language (Spanish and English). In Table 5.2 detailed results of the experiments are reported. As already discussed, the binary accuracies reported are the median over five random initializations on the test set, except for SVM and NB, because their deterministic nature allow reporting accuracy in a single run. For the same reason, the standard deviation of these two models is not reported in the table. The standard deviation is calculated using the five accuracies over the five random initialization. This metric further provides information on the ability of each model to replicate constant results over several runs. As hypothesized during my preliminary linguistic analysis, performances over the English test set are worse compared to the Spanish test set for all the models evaluated. The results indicate the shallow CNN as the best performing model on both test sets (English and Spanish), and the one achieving the smallest standard deviation on the Spanish test set. The smallest standard deviation over the English test set is obtained by the Multi-CNN. Transformer-based models generally perform worst in terms of standard deviation. It is interesting that the linear SVM is able to outperform any Transformers on the Spanish test set, but ELECTRA. On the other hand, NB on the English test set, is able to perform better (or equal, compared to RoBERTa) of any Transformer evaluated. Given the size of each sample in the dataset, results are in line with those reported in other studies ([49]). As far as Longformer is concerned, I expected a better performance from it. It is worth bearing in mind that each sample within train and test sets contains a feed of the last 100 tweets of a single user. This size would perfectly fit the information that a Longformer can manage. However, the results suggest that this is not enough to capture relevant user features based on the whole thread of each user. These low performances could be motivated by the fact that the user is not represented with a long, consistent text. The 100-tweet sequence per user cannot be considered as a text (i.e., a coherent stretch of language), because each tweet is not always and directly linked to the previous and to the next one. To this content fragmentation could be due the poor performances of the Longformer. Contrarily to Transformers, for the CNN, fragmentation could be a positive feature. In fact, the 100-tweet thread per user could be seen as a picture composed by 100 different parts, each representing an aspect—represented in 280 pixels

Table 5.1: Models accuracies and standard deviation. For non-deterministic models, accuracy is the median over five runs. In the table, the best results are shown in bold.

| | **English** | | **Spanish** | |
|---|---|---|---|---|
| | **Acc** | **$\sigma$** | **Acc** | **$\sigma$** |
| CNN | **0.715** | 0.022 | **0.815** | **0.005** |
| Multi-CNN | 0.545 | **0.004** | 0.670 | 0.013 |
| BERT | 0.625 | 0.036 | 0.735 | 0.018 |
| RoBERTa | 0.695 | 0.014 | 0.735 | 0.024 |
| ELECTRA | 0.630 | 0.016 | 0.760 | 0.015 |
| DistilBERT | 0.645 | 0.016 | 0.725 | 0.014 |
| XLNet | 0.675 | 0.020 | 0.710 | 0.070 |
| Longformer | 0.685 | 0.041 | 0.695 | 0.007 |
| Naive Bayes | 0.695 | - | 0.695 | - |
| SVM | 0.630 | - | 0.755 | - |

(because their longest sequence of tweets is 280 characters)—of the full picture. Some users are more diverse than others, depending on the variety in their feed. Since CNN filters are able to scan each content window and focus on the relevant features, it is not surprising that they are able to cope well with image classification/recognition, which in my opinion is comparable to the content fragmentation I have in this task. Apart from this, from my experimental evaluation it emerges that non-deep models are not a second choice compared to Transformers. In fact, a simple ensemble of NB and SVM models could achieve better performances than Transformers on this and on similar binary classification datasets.

On the Hate Speech Spreaders dataset, the overall best result has been obtained by [259] with a 100-dimension word embedding representation to feed a CNN. The two ex aequo second best performing teams, respectively fine-tuned a transformer to replicate and modify the method previously used in authorship verification [276], and used a meta-classifier fed with combinations of n-grams [21].

On the ISS dataset, the best approach is obtained by [308] with a BERT feature-based CNN model. The second-best result has been achieved by [273] with a combination of SBERT and emojis. The third best performing teams [114] used a Random Forest fed with unigrams pre-selected with several techniques such as PMI and TF-IDF with the aim to maximize the probability difference of each feature for each class.

Sentiment analysis, news categorization, topic labeling, and NLI tasks all differ significantly from one another and cannot be described simply as a TC task. The performance of the primary models presented in their articles on classic datasets is tabulated in this part and evaluated by classification accuracy. Due to experiments on datasets using a less conventional TC model, the performance of the NB and SVM algorithms is provided from [163] in Figure 5.4. For SST-2, NB and SVM had

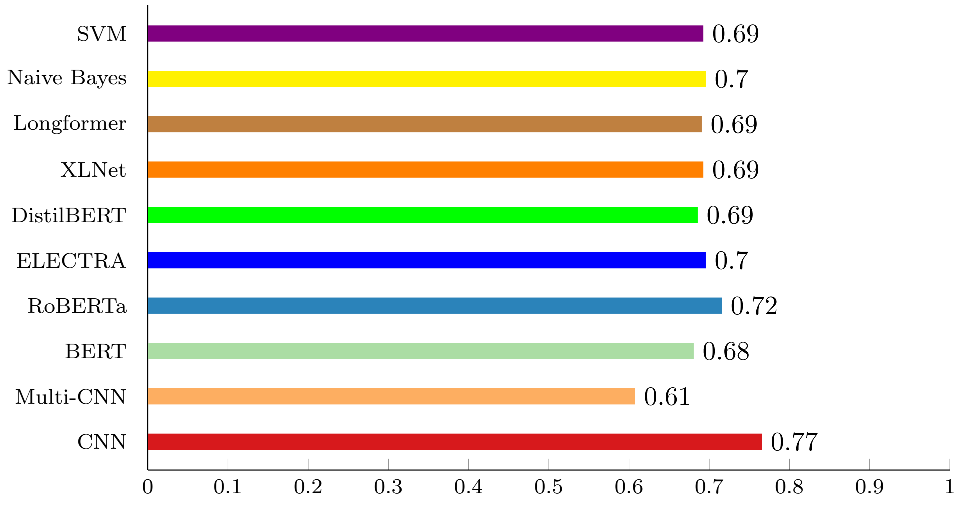


Figure 5.3: Average accuracy of each model evaluated. AVG accuracy is calculated averaging the accuracy in each language (Spanish and English).

Table 5.2: Model accuracies for the first ten participants with a referenced paper at the shared tasks discussed in [239, 45]. Where multilanguage datasets were provided, results on both languages, i.e., English (EN) and Spanish(ES) are reported. The best results per language and dataset are shown in bold.

| **Hate Speech Spreaders** | | | |
|---|---|---|---|
| Model | EN | ES | AVG |
| CNN [259] | 0.730 | **0.850** | **0.790** |
| META-Classifier [21] | 0.730 | 0.830 | **0.780** |
| Transformer [276] | 0.740 | 0.820 | 0.780 |
| BERT-BETO [13] | 0.720 | 0.820 | 0.770 |
| BERT [272] | 0.730 | 0.800 | 0.765 |
| CNN [113] | 0.730 | 0.790 | 0.760 |
| BERT [86] | **0.670** | **0.830** | 0.750 |
| TF-IDF+SVM [127] | 0.690 | 0.810 | 0.750 |
| Ensemble [51] | 0.700 | 0.800 | 0.750 |
| Stack Ensemble [82] | 0.700 | 0.790 | 0.745 |

| **Irony and Stereotype Spreaders** | |
|---|---|
| Model | EN |
| BERT+CNN [308] | 0.9944 |
| SBERT+Emojis [273] | 0.9778 |
| RF [114] | 0.9722 |
| Transformers [79] | 0.9667 |
| Iro-Net [98] | 0.9611 |
| BERT [271] | 0.9556 |
| GPT2+NB [112] | **0.9556** |
| RF [243] | 0.9556 |
| TF-IDF+RF [279] | 0.9500 |
| LR Ensemble [262] | 0.9444 |

accuracy rates of 81.8 and 79.4 percent, respectively. NB has more accuracy than SVM in the SST-2 dataset, with just two categories. It might be as a result of NB's rather consistent categorization accuracy on fresh data sets. On tiny data sets, the performance is also consistent. NB performs worse than the deep learning model in terms of performance. The advantage of NB over deep models is that it requires less computing. Yet because it needs manually created categorization features, it's challenging to apply the model straight to other data sets. Pre-trained models perform better on the majority of datasets for deep learning models. It means that, except for MR and 20NG, which haven't been tested on BERT-based models, pre-trained models should be evaluated first for a TC task. RNN

Capsule-accuracy for the MR dataset is 83.8 percent, yielding the best outcome. It recommends that for sentiment analysis, RNN-Capsule creates a capsule for each category. Without using linguistic expertise, it may output words together with sentiment trends indicating capsule qualities. BLSTM-2DCNN receives a 96.5 percent score for 20NG dataset, which is the best accuracy score. It might show how well the 2D max-pooling operation works for getting a fixed-length representation of the text, and how well 2D convolution works for sampling more valuable matrix data.

Table in Figure 5.5 is presented in [200]. The table lists the sentiment analysis dataset findings from Yelp, IMDB, SST, and Amazon. Since the debut of the first DL-based sentiment analysis model, it is visible that accuracy has significantly improved, as evidenced by, for instance, a 78 percent relative reduction in classification error (on SST-2).

Table in Figure 5.6 is discussed in [200] and reports the performance on three news categorization datasets (i.e., AG News, 20-NEWS, Sogou News) and two topic classification datasets (i.e., DBpedia and Ohsummed). A similar trend to that in sentiment analysis is observed.

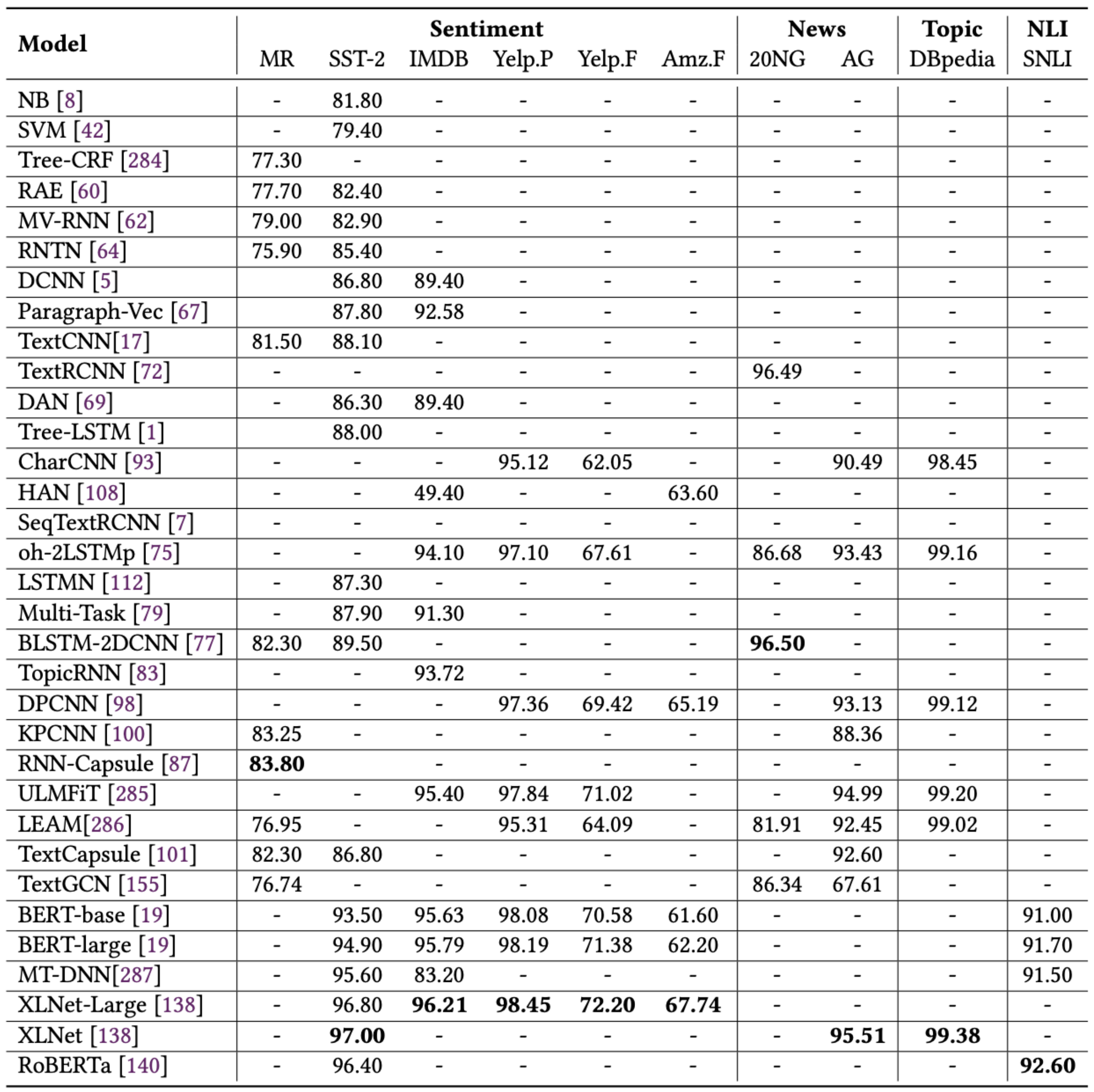
## 5.3 Post-hoc analysis

In this section, I report an example of analysis of the results and the behavior of a deep model trained on a specific dataset. The analysis presented here was conducted focusing on the FNS dataset to investigate the behavior of a simple CNN and its predictions on the test set after completing the training phase [256]. This further step can be employed in the TC pipeline to improve the performance of a model and for a better understanding of its behaviors (RQ3).

In Section 2.3, I observe that keywords are good indicators to distinguish the two FNS and nFNS classes, as corroborated also by the results of the Bayesian model reported in Table 5.2. However, the CNN-based model must go beyond these frequency differences, as its results suggest. In this section, I provide a post-hoc analysis of intermediate model outputs, devised to shed light on the CNN behavior. In particular, I analyze the outputs of two hidden layers: the convolutional layer and the global average pooling layer. These are the model layers that can be analyzed by relating the outputs to the inputs to better understand the overall classification decision. Although hybrid approaches have been exploited to eXplainable AI [133], the CNN tested here can be defined as a shallow neural model. Thus, it can be analyzed mapping each layer outputs to its inputs.

### 5.3.1 Convolutional Layer Output

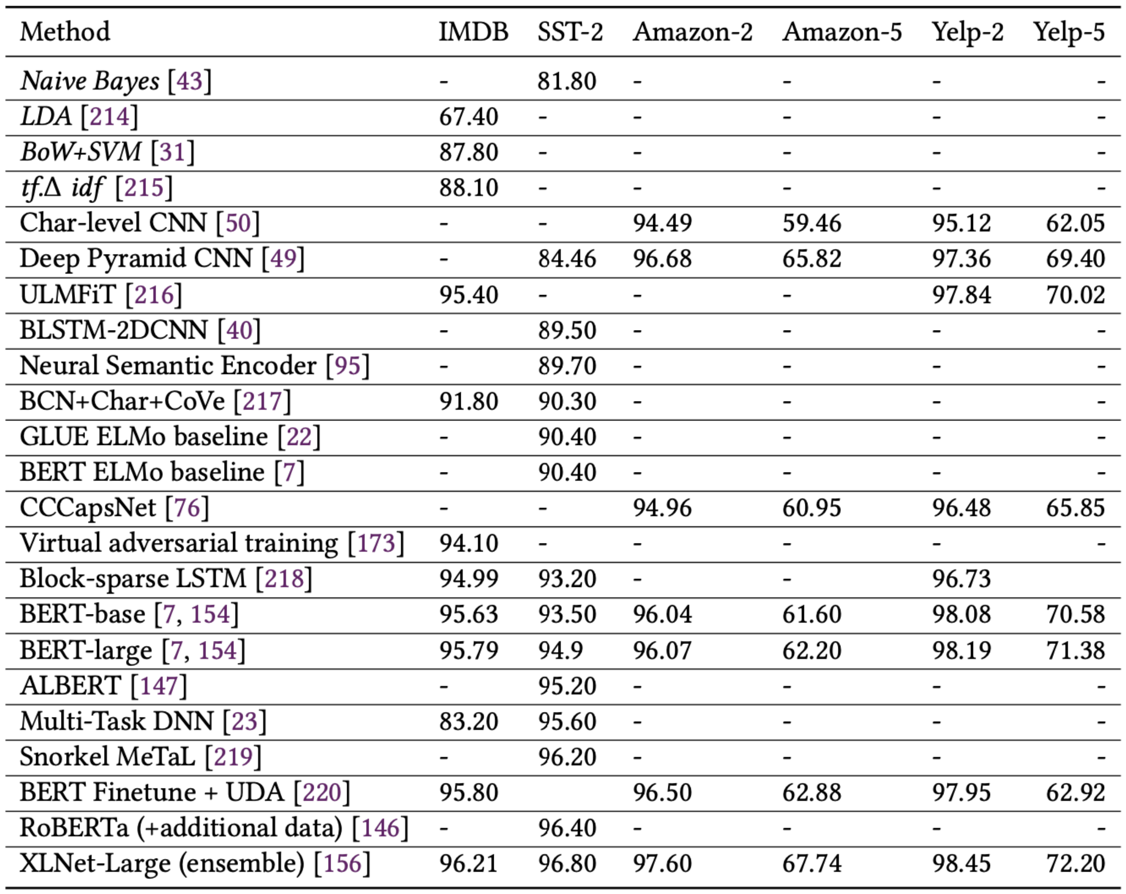
The output of each filter of the convolutional layer was searched for finding maximum and minimum values in the output tensor. The hypothesis is that these values correspond to some tweets, captured by the filter window, showing some relevant linguistic features I found during my preliminary



| Model | Sentiment | | | | | | News | | Topic | NLI |
|---|---|---|---|---|---|---|---|---|---|---|
| | MR | SST-2 | IMDB | Yelp.P | Yelp.F | Amz.F | 20NG | AG | DBpedia | SNLI |
| NB [8] | - | 81.80 | - | - | - | - | - | - | - | - |
| SVM [42] | - | 79.40 | - | - | - | - | - | - | - | - |
| Tree-CRF [284] | 77.30 | - | - | - | - | - | - | - | - | - |
| RAE [60] | 77.70 | 82.40 | - | - | - | - | - | - | - | - |
| MV-RNN [62] | 79.00 | 82.90 | - | - | - | - | - | - | - | - |
| RNTN [64] | 75.90 | 85.40 | - | - | - | - | - | - | - | - |
| DCNN [5] | | 86.80 | 89.40 | - | - | - | - | - | - | - |
| Paragraph-Vec [67] | | 87.80 | 92.58 | - | - | - | - | - | - | - |
| TextCNN[17] | 81.50 | 88.10 | - | - | - | - | - | - | - | - |
| TextRCNN [72] | - | - | - | - | - | - | 96.49 | - | - | - |
| DAN [69] | - | 86.30 | 89.40 | - | - | - | - | - | - | - |
| Tree-LSTM [1] | | 88.00 | - | - | - | - | - | - | - | - |
| CharCNN [93] | - | - | - | 95.12 | 62.05 | - | - | 90.49 | 98.45 | - |
| HAN [108] | - | - | 49.40 | - | - | 63.60 | - | - | - | - |
| SeqTextRCNN [7] | - | - | - | - | - | - | - | - | - | - |
| oh-2LSTMp [75] | - | - | 94.10 | 97.10 | 67.61 | - | 86.68 | 93.43 | 99.16 | - |
| LSTMN [112] | - | 87.30 | - | - | - | - | - | - | - | - |
| Multi-Task [79] | - | 87.90 | 91.30 | - | - | - | - | - | - | - |
| BLSTM-2DCNN [77] | 82.30 | 89.50 | - | - | - | - | **96.50** | - | - | - |
| TopicRNN [83] | - | - | 93.72 | - | - | - | - | - | - | - |
| DPCNN [98] | - | - | - | 97.36 | 69.42 | 65.19 | - | 93.13 | 99.12 | - |
| KPCNN [100] | 83.25 | - | - | - | - | - | - | 88.36 | - | - |
| RNN-Capsule [87] | **83.80** | | - | - | - | - | - | - | - | - |
| ULMFiT [285] | - | - | 95.40 | 97.84 | 71.02 | - | - | 94.99 | 99.20 | - |
| LEAM[286] | 76.95 | - | - | 95.31 | 64.09 | - | 81.91 | 92.45 | 99.02 | - |
| TextCapsule [101] | 82.30 | 86.80 | - | - | - | - | - | 92.60 | - | - |
| TextGCN [155] | 76.74 | - | - | - | - | - | 86.34 | 67.61 | - | - |
| BERT-base [19] | - | 93.50 | 95.63 | 98.08 | 70.58 | 61.60 | - | - | - | 91.00 |
| BERT-large [19] | - | 94.90 | 95.79 | 98.19 | 71.38 | 62.20 | - | - | - | 91.70 |
| MT-DNN[287] | - | 95.60 | 83.20 | - | - | - | - | - | - | 91.50 |
| XLNet-Large [138] | - | 96.80 | **96.21** | **98.45** | **72.20** | **67.74** | - | - | - | - |
| XLNet [138] | - | **97.00** | - | - | - | - | - | **95.51** | **99.38** | - |
| RoBERTa [140] | - | 96.40 | - | - | - | - | - | - | - | **92.60** |

Figure 5.4: Accuracy of TC models on primary datasets evaluated by classification accuracy (in terms of publication year). Bold is the most accurate. (Original image taken from [163])

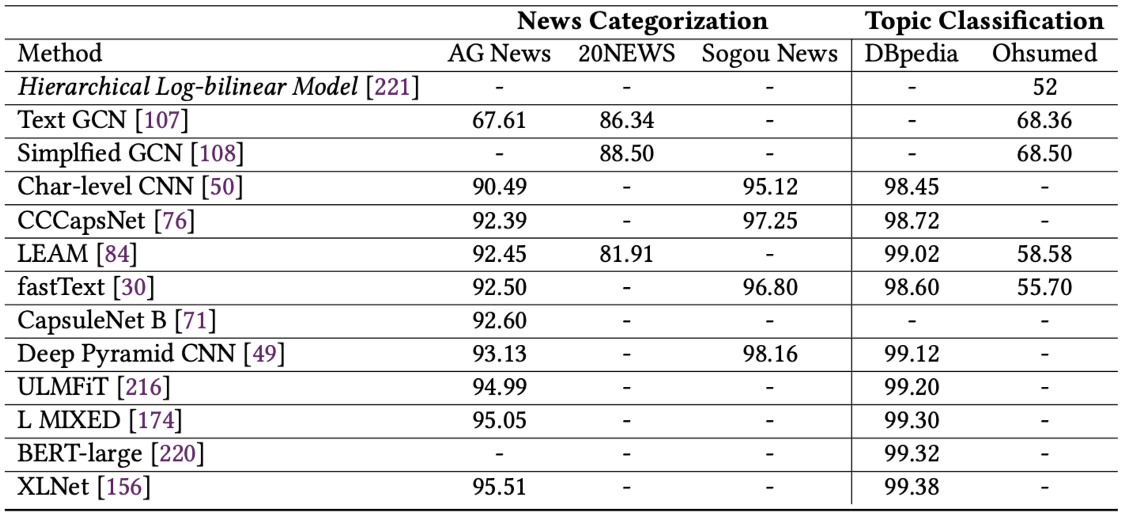
.



| Method | IMDB | SST-2 | Amazon-2 | Amazon-5 | Yelp-2 | Yelp-5 |
|---|---|---|---|---|---|---|
| *Naive Bayes* [43] | - | 81.80 | - | - | - | - |
| *LDA* [214] | 67.40 | - | - | - | - | - |
| *BoW+SVM* [31] | 87.80 | - | - | - | - | - |
| *tf.Δ idf* [215] | 88.10 | - | - | - | - | - |
| Char-level CNN [50] | - | - | 94.49 | 59.46 | 95.12 | 62.05 |
| Deep Pyramid CNN [49] | - | 84.46 | 96.68 | 65.82 | 97.36 | 69.40 |
| ULMFiT [216] | 95.40 | - | - | - | 97.84 | 70.02 |
| BLSTM-2DCNN [40] | - | 89.50 | - | - | - | - |
| Neural Semantic Encoder [95] | - | 89.70 | - | - | - | - |
| BCN+Char+CoVe [217] | 91.80 | 90.30 | - | - | - | - |
| GLUE ELMo baseline [22] | - | 90.40 | - | - | - | - |
| BERT ELMo baseline [7] | - | 90.40 | - | - | - | - |
| CCCapsNet [76] | - | - | 94.96 | 60.95 | 96.48 | 65.85 |
| Virtual adversarial training [173] | 94.10 | - | - | - | - | - |
| Block-sparse LSTM [218] | 94.99 | 93.20 | - | - | 96.73 | |
| BERT-base [7, 154] | 95.63 | 93.50 | 96.04 | 61.60 | 98.08 | 70.58 |
| BERT-large [7, 154] | 95.79 | 94.9 | 96.07 | 62.20 | 98.19 | 71.38 |
| ALBERT [147] | - | 95.20 | - | - | - | - |
| Multi-Task DNN [23] | 83.20 | 95.60 | - | - | - | - |
| Snorkel MeTaL [219] | - | 96.20 | - | - | - | - |
| BERT Finetune + UDA [220] | 95.80 | | 96.50 | 62.88 | 97.95 | 62.92 |
| RoBERTa (+additional data) [146] | - | 96.40 | - | - | - | - |
| XLNet-Large (ensemble) [156] | 96.21 | 96.80 | 97.60 | 67.74 | 98.45 | 72.20 |

Figure 5.5: Accuracy of TC models on sentiment analysis datasets . (Original image taken from [200])

.



| | News Categorization | | | Topic Classification | |
|---|---|---|---|---|---|
| Method | AG News | 20NEWS | Sogou News | DBpedia | Ohsumed |
| *Hierarchical Log-bilinear Model* [221] | - | - | - | - | 52 |
| Text GCN [107] | 67.61 | 86.34 | - | - | 68.36 |
| Simplfied GCN [108] | - | 88.50 | - | - | 68.50 |
| Char-level CNN [50] | 90.49 | - | 95.12 | 98.45 | - |
| CCCapsNet [76] | 92.39 | - | 97.25 | 98.72 | - |
| LEAM [84] | 92.45 | 81.91 | - | 99.02 | 58.58 |
| fastText [30] | 92.50 | - | 96.80 | 98.60 | 55.70 |
| CapsuleNet B [71] | 92.60 | - | - | - | - |
| Deep Pyramid CNN [49] | 93.13 | - | 98.16 | 99.12 | - |
| ULMFiT [216] | 94.99 | - | - | 99.20 | - |
| L MIXED [174] | 95.05 | - | - | 99.30 | - |
| BERT-large [220] | - | - | - | 99.32 | - |
| XLNet [156] | 95.51 | - | - | 99.38 | - |

Figure 5.6: Accuracy of TC models on sentiment analysis datasets (in terms of classification accuracy), evaluated on the IMDB, SST, Yelp, and Amazon datasets . (Original image taken from [200])
.

analysis. Reverse mapping the input tokens corresponding to the filter window, I identified the 32-token windows with maximum and minimum values assigned. The 32-token windows receiving the maximum value are those considered important by the convolutional layer filters and, consequently, pass also the max pooling layer. Thus, I randomly downloaded 15 samples per class (10% of the train) together with the 32-token windows with the maximum and minimum values assigned. These 32-token windows consist of maximum three complete tweets. I noticed that the majority of the 32 filters outputted a maximum or minimum value for the same windows of tokens (with a variation of a few tokens) per author sample. This behavior suggests that a lower number of filters could have been enough for capturing the token patterns which are more relevant for classifying an author as FNS or nFNS. I observed that giving as input the whole collection of 100 tweets per author produced two or three peaks in the filter output that are clearly distinguishable from the other local maximum values. An example of output corresponding to the complete filtering of a reference author found by the first convolutional filter is graphically shown in Figure 5.7. The document—i.e., the author's 100 tweets—consists of about 2000 tokens, then it is padded up to 4060. The output of this filter shows a global maximum in position 1739, indicating that in that 32-token window there are relevant features. To see what this window contains, I looked at the vocabulary and did a reverse mapping. I applied this procedure to all the windows with maximum and minimum valued tokens, allowing an analysis of the linguistic features that the best performing model considers most or less important when classifying the sample.

Analyzing FNS and nFNS 32-token windows considered important by the filters, I found some

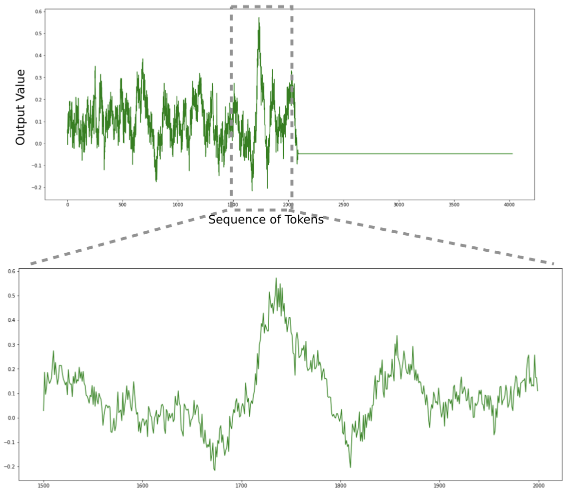


Figure 5.7: Output of the first convolutional layer after convolving one of the 32 filters over the input provided. The maximum value corresponds to the token in position 1739 and the minimum corresponds to the token in position 1673. The sample shown consists of less than 1500 tokens, hence the document is padded up to 4060.

patterns, reported below, corresponding to specific topics and tweet style, such as the usage of the first person or the formulation of a question.

*FNS Patterns.* I found both features in accordance with my preliminary analysis and not. On the one hand, in these windows of FNS samples, I found information about: 1. tricks, miracle foods or home remedies (e.g., *El truco para secar la ropa sin necesidad de tenderla — VÍDEO #URL#*, 'The trick to drying clothes without hanging them out to dry'); 2. sensitive (o strong) images or videos (e.g., *FUERTE VÍDEO – Matan Hombre Por Violar Niñas #URL# #URL#*, 'STRONG VIDEO — Man Killed For Raping Girls'); 3. music (e.g., *Chimbala anuncia union entre algunos dembowseros para cambiar el sonido musical de ese genero!!! #URL# Unete #USER#>*, 'Chimbala announces union between some dembowsers to change the musical sound of this genre!!!'). On the other hand, I also found tweets containing: 1. personal opinions (e.g., *no te vas a poner a dialogar sobre la cosntruccion de un nuevo pais,sobre aristotles,pitagoras o engels.*, 'you are not going to start a dialogue about the construction of a new country, about Aristotle, Pythagoras or Engels.'); 2. political news (e.g., *El nuevo Gobierno boliviano detendrá a diputados del partido de Morales por [UNK] y sedición" #URL#*, 'The new Bolivian government will arrest deputies from Morales' party for [UNK] and sedition').

*nFNS Patterns.* I noticed in nFNS sample windows: 1. complete questions (e.g., *¿Por qué se nos riza el pelo? ¿Por qué crece pero las pestañas y el vello no? #URL# vía #USER#*, 'Why does our hair get frizzy? Why does it grow but the eyelashes and hair don't? #URL# via #USER#'); 2. series of mentions (from three up; two mentions in a row are also present in FNS sample data) (e.g.,

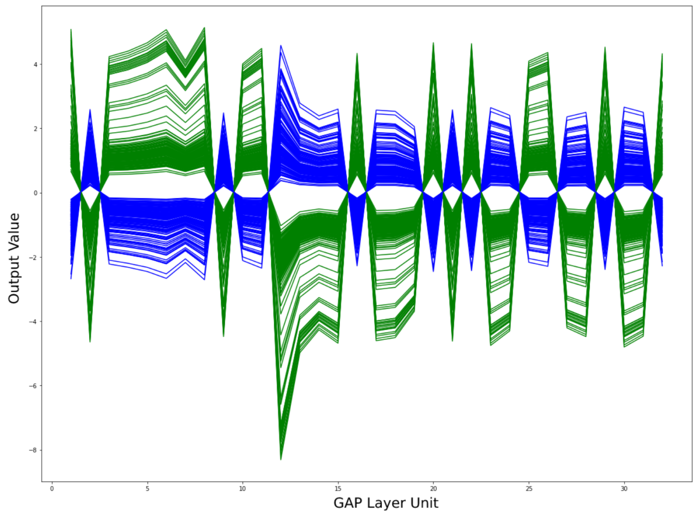


Figure 5.8: Global average pooling layer output, providing the training test as model input. For both classes (i.e., FNS and nFNS) every sample is correctly classified. In this case no overlapped lines are visible between the two groups of lines (i.e., green or blue). Each line corresponds to an author.

*#USER# #USER# #USER# #USER# #USER# #USER# #USER# Quería poner tocaros,no tocarlos...*, '#USER# #USER# #USER# #USER# #USER# #USER# #USER# I wanted to touch you, not touch you...'); 3. politics (e.g., *Se ha visto Sr$^{a}$ #USER# en estas imágenes, a mi me da verguenza, una diputada del congreso*, 'It has been seen Miss #USER in these images, it gives me shame, a deputy of the congress'); 4. emojis (almost absent in FNS maximum outputs).

This analysis suggests that the CNN model might consider important the features highlighted in the preliminary analysis of the dataset. However, what emerges is also that this CNN model might be biased towards some topics (e.g., music for FNS and politics for nFNS).

### 5.3.2 Global Average Pooling Output

Figure 5.3.2 shows the output of the global average pooling layer when the training set is provided as input. On x-axis, I represent the 32 units of the layer, on the y-axis the values associated to each unit. For every sample of the set, a line is drawn connecting the 32 output values of each unit of the level. Blue lines represent FNS, while green nFNS. Similarly, Figure 5.3.2 shows values of the 32-GAP-output units when test set samples are provided to the CNN. In this case, some lines near to 0 values output fall outside their actual area. This might suggest that wrongly predicted samples are similar to the opposite class, hence confusing my classifier when making predictions.

Thus, I extracted two documents per class selecting one document whose 32-GAP-output values are far from the 0 threshold and one near it, because I imagined that highly characterized documents

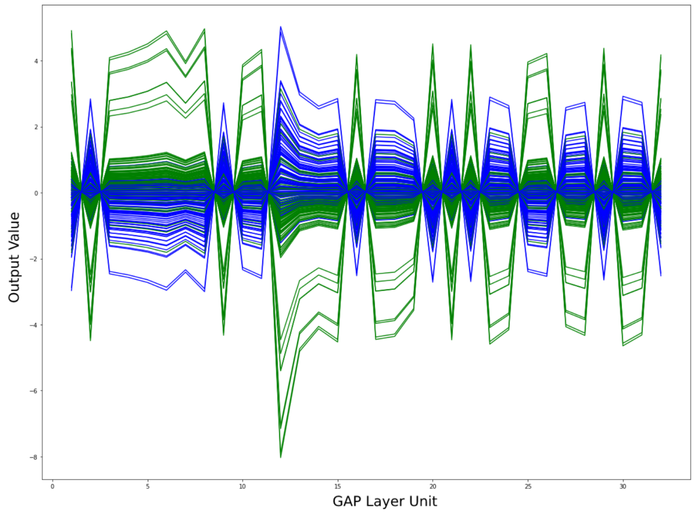


Figure 5.9: Global average pooling layer output, providing the full test set as model input. In this case some errors are visible (i.e., green lines in blue-line zone and vice versa). It is worth noting that errors in detection are often near to 0 values output. This might suggest that the 0.0 threshold value used to separate the classes is small, and this could possibly explain model mistakes.

(i.e., documents which contain a high number of features characteristics of their class) should be far from 0. As expected, the features highlighted in the preliminary analysis are in a higher number in those documents whose 32-GAP-output values are far from 0. In particular, 52% of tweets in the far-from-0 FNS document start with *VIDEO*, *DE ULTIMO MINUTO*[4] 'breaking news', *ESTRENO*[5], *IMPACTANTE*[6], or *DESCARGAR*[7], 76% contain *Unete* at the end of the tweet (i.e., contain keywords of FNS as reported in Table 2.3). Similarly, in the far-from-0 nFNS document, 19% of the total number of tokens is made of *#HASHTAG#*, in addition to other keywords reported also in Table 2.3 such as *Samsung*, *bulos*[8], *qué*[9], *informacíon*[10], but also complete questions (starting with *¿* and ending with *?*) as emerged as important feature analyzing the first convolutional layer output. In the two documents whose 32-GAP-output values are near to 0, I found a similar tweet (nFNS: *He publicado una foto nueva en Facebook #URL#*, 'I have posted a new photo on Facebook #URL#.', and FNS: *He publicado un vídeo nuevo en Facebook #URL#.*, 'I have posted a new video on Facebook #URL#.') repeated more than once, 33 and 7 times out of 100 in nFNS and FNS, respectively. This, not only, reduces the variety of features available for classifying each document, but also it is a similar behavior shared by the two opposite-class authors. In addition, in both

[4]In English: *last minute.*
[5]In English: *premiere.*
[6]In English: *shocking.*
[7]In English: *discharge.*
[8]In English: *hoaxes.*
[9]In English: *that.*
[10]In English: *information.*

documents at least a quarter of tweets are retweets (25% and 29% in nFNS and FNS, respectively), though different in nature. In particular, the analyzed nFNS author retweeted mostly users' personal opinion (e.g., about politics), whilst the FNS author retweeted mostly crime news.

### 5.3.3 Qualitative error analysis

The CNN tested, in the best performing run on the Spanish dataset, reaches an accuracy of 0.82 and fails to recognize 19 FNS and 17 nFNS authors, confirming that FNS are slightly more difficult to identify than nFNS. Since it is worst to mistakenly label a nFNS as an FNS, I decided to analyze the features of wrongly and correctly identified nFNS—following the suggestion by [28] who propose that error analysis should also be done on correctly identified samples to verify why the system performs well, especially when using black-box models.

Since I suppose that the CNN model considers important for the classification the distribution of keywords, I selected three nFNS samples—one wrongly identified as FNS and two correctly identified as nFNS—containing keywords identified as a good predictor of FNS. I found that the CNN model is able to distinguish different usages of the same keyword. In Table 5.3.3, I show three different examples in which the lemma *remedio*[11] (cfr. Table 2.3) is used in two different ways. Examples 1 and 3 are similar to what can be found in FNS tweets. Example 2 is very different from FNS authors' usage. Since the model does not make its decision based only on one tweet (the first convolutional layer takes 32-token windows corresponding to maximum three complete tweets), I can suppose that the presence of the tweets reported in Example 1 and 3 are not enough to assign the FNS label to a nFNS. The author sharing the tweet in Example 1 was wrongly labelled as FNS by the CNN model. The authors sharing the tweets in Examples 2 and 3 were correctly identified as nFNS by the CNN model.

The author that shared the tweet in Example 2, shared also several features in line with what I found in the preliminary analysis for nFNS. This author, indeed, always publishes where to find information concerning what they say and also shares information on how to counteract misinformation, thus I might suppose that the CNN model pays more attention to these features and not on the presence of that precise tweet containing a keyword of FNS. Then, the question is why the authors sharing Example 1 and 3 are not both wrongly—or correctly—predicted. The author who shared the tweet in Example 1, not only uses the keyword *remedio* (used by many FNS), but also contains several variants of one of the high discriminant keywords pinpointed both by Sketch Engine and by the Bayesian model, i.e., *video*. Conversely, the author sharing the tweet in Example 3, apart from sharing *powerful remedies*, they ask many questions (and I saw in SubSection 5.3.1 that the convolutional filters consider questions as good predictors of nFNS) and publishes personal opinions in both explicit and implicit form (e.g., *yo opino*, 'I think'; *yo digo*, 'I say'; *yo comento*, 'I comment').

[11]In English: *remedy*.

Table 5.3: Examples drawn from the nFNS Spanish test set.

| Example | Tweet Text | English translation |
|---|---|---|
| 1 | RT #USER#: Remedio casero para limpiar las juntas del azulejo. #URL# | RT #USER#: Home remedy to clean the joints of the tile. #URL# |
| 2 | La venta de medicamentos con receta bajó todos los años entre 2016 y 2019. Además, en 2018 la mitad de los hogares pobres de CABA y el Conurbano debieron dejar de comprar remedios por problemas económicos. Más info en esta nota #URL# de #USER#. #URL# | The sale of prescription drugs fell every year between 2016 and 2019. In addition, in 2018 half of poor households in CABA and the suburbs had to stop buying medicines due to economic problems. More info in this note #URL# de #USER#. #URL# |
| 3 | Poderoso remedio casero para eliminar el colesterol de los vasos sanguíneos y perder peso -... #URL# | Powerful home remedy to remove cholesterol from blood vessels and lose weight -... #URL# |

Hence, I supposed that the CNN model is able to discriminate also on the basis of the presence of overtly expressed personal pronouns. I checked if FNS and nFNS use the first-person pronoun *yo* differently. I performed a Welch $t$ test and found a statistically significant $p$ value of 0.0194 when inspecting together test and train, a $p$ value not quite statistically significant looking only at train data (0.0833), and a $p$ value of 0.1158 in test data, which is not statistically significant. Thus, I might suppose that since it was trained only on train data, this difference should not be so discriminant. Then, I wanted to measure if nFNS use more first-person verbs and pronouns than FNS (both singular and plural). To obtain this type of information, I automatically parsed the dataset using the AnCora pretrained model with UDPipe[12] The linguistic annotation confirmed that nFNS tweets contain more first-person tokens than FNS. Hence, I performed a Welch $t$. test to determine if this difference is statistically significant. I found a $p$ value less than 0.0001, thus extremely statistically significant. I also investigated if also the second and the third person features were significantly different and found a $p$ value less than 0.0001 for each person (1, 2 and 3, taken singularly) and also when aggregated. This last result suggests that these two classes use differently verbs (and auxiliaries) and pronouns—the only parts of speech that can have this morphological feature (i.e., person).

[12] https://lindat.mff.cuni.cz/services/udpipe/.

# Chapter 6

# Applications for competitive tasks

Following the introduction of each stage in the TC pipeline in preceding chapters, the current chapter showcases architectures that incorporate these stages comprehensively. These architectures undergo evaluation against contemporary tasks and SOTA benchmarks to establish correlations between their structures, model functionalities, and the outcomes achieved in specific tasks. Therefore, after introducing and discussing some contemporary international challenges in the field of NLP, I provide a detailed presentation of some of the classification models that I have developed to propose my original contribution to the field.

The NLP community has been characterized for several years by some main evaluation campaigns, in particular those proposed in SemEval. SemEval[1] (Semantic Evaluation) arose from the Senseval word sense evaluation series and is a continuing series of evaluations of computational semantic analysis systems. The assessments aim to investigate the nature of meaning in language. Although people have an intuitive understanding of meaning, applying such understanding to computational analysis has proven difficult.

A technique is being provided by this set of evaluations to more precisely define what is required to compute in meaning. As a result, the evaluations offer an emergent mechanism to pinpoint the issues and fixes for calculations that have purpose. These activities have developed to express a wider range of the factors that influence the use of the language. They started with what appeared to be straightforward efforts to computationally determine word senses.

Some tasks related to this chapter are the ones presented and discussed in [284, 25]. Other interesting recent tasks presented at SemEval or in other venues are related to abusive language detection [10], toxic language detection [69, 244] and automatic misogyny identification [85].

In the first section of this chapter, I report all the architectures I developed to address some recent TC tasks. In the last section, I report the conclusions to motivate and try to generalize on

[1]`https://semeval.github.io/`

some common patterns found across the different tasks proposed in this chapter. The tasks are a subset of the ones discussed in the Chapter 2.

## 6.1 Detection of hate speech spreaders using CNN

### 6.1.1 Task description

The aim of the PAN 2021 Profiling Hate Speech Spreaders (HSSs) on Twitter task [33, 239] was to investigate whether the author of a given Twitter thread is likely to spread tweets containing hate speech or not. The multilingual dataset, namely English and Spanish datasets provided by the organizers of the task, consisted of 120,000 tweets: 200 tweets per author, 200 authors per each language training set and 100 authors per each language test set [238]. The model I used to compete for the task consists of a shallow CNN. Broadly speaking, my network preprocess each sample in the dataset to build a dictionary[2] where the key is an integer number and the value is an n-gram resulting from my custom preprocessing function. Each integer value (e.g., a key in the dictionary) is then mapped to a single point into a 100-dimensional word embedding space. Then, a 1-dimensional convolution is applied. The output of the convolution layer is then fed into an average pooling and then into a global average pooling layer fully connected to a single dense layer output.

### 6.1.2 Proposed model

The architecture of the model is presented in Figure 6.1, in which the dimensions of inputs and outputs of each model layer are highlighted.

In what follows, I present each layer of the network and the chosen hyperparameter values. Before discussing the network architecture, it is important to bear in mind that each set of the dataset (training and test per language) is made of XML files—each XML is related to a single author—containing 200 tweets of the author. In addition, a ground truth file containing the labels 0 and 1 matched to each XML file is provided for the training set. For handling these files and before training, my system organizes these XML files into two folders (i.e., 0 and 1) while reading the ground truth file. Then each sample (i.e., a single XML file) is read by the model for training or test, depending on the number of fold validation. These function for reading samples is accomplished by the first layer (namely, *InputLayer*).

[2]In this paper, I use the well known computer science concept of a dictionary. A dictionary (or associative array) is a data type composed by a collection of (key, value) pairs. However, in the TextVectorization class definition used in my model is used the word *vocabulary* referring to a list of tokens (e.g., n-grams returned after preprocessing some input text) in which keys are token indices for such a list and values are the tokens.

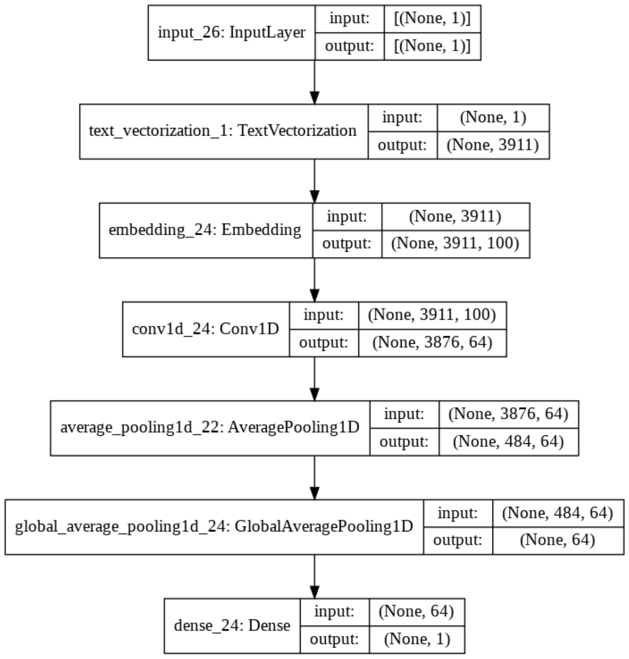


Figure 6.1: Model architecture. Numbers in brackets indicate tensor dimensions; *None* stands for the batch dimension not yet known before running the model. Layers as depicted on my Google Colab Notebook.

### Text vectorization

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

- *Standardize.* It is the preprocessing function applied to the text before proceeding with its vectorization. In my case, this function, in addition to removing tabulations and newline characters, substitutes the occurrences of the CDATA tag with a space followed by a *minus than* sign, adds a space between the closing of one tag and the next, and then split each n-gram at each space;
- *Max tokens.* This parameter refers to the dictionary size. To get this value, I simply count the numbers of different n-grams resulting from my preprocessing step. It is worth noting that my dictionary size is developed scanning both the Spanish and the English training sets;
- *Output mode.* This parameter is the type of token index returned by the vectorization function. I used the INT type, so that every word is mapped to a positive integer number;
- *Sequence length.* Although each XML document contains 200 tweets, the size in terms of produced n-grams is different for each sample because of the different length of each tweet. For this reason, I decided to consider the longest sample of the training set as size value, padding the shorter documents. As shown in Figures 6.1 and 6.2, this size is 3,911. As mentioned, padding is used for documents with a resulting number of token indices less than 3,911. Eventually, longer documents in the test set would be cut at this value.

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

...

$$\textit{rock} \rightarrow 210$$
$$...$$
$$\textit{Hi!} \rightarrow 2315$$
$$...$$
$$\textit{pregunta} \rightarrow 1508$$
$$...$$

**Embedding**

This layer takes as input a tensor of 3,911 integer numbers generated as described in the previous subsection. Each integer value of this tensor is mapped to a 100-dimension word embedding tensor. In this way, each integer from the previous layer is mapped to a single tensor consisting of 100 floating point values. A notable difference with the previous layer is that the 100 coordinate values of each tensor is updated at each optimization step while training the model. More precisely, I trained and tested multiple models as the word embedding space varies from 2 to 800 dimensions, as also discussed in a similar Twitter TC problem [303]. The best performances over different tests on a 5-fold cross validation were obtained with a 100-dimension embedding space.

**Convolution**

In my model, a single 1D-convolution layer is implemented. This layer consists of 64 filters of size 36. The layer then performs convolution on 36-ngram windows with stride value of 1 (i.e., after each convolution, the convolutional filter is shifted of one word embedding tensor). For this layer, no padding is added and ReLu [88, 89] is used as activation function on the output values. Number of filters and filters size (i.e., the two main parameters of this layer) are of paramount importance for the global performance of the model. Indeed, the filter size determines the size of the windows over the text of the input sample provided. In this way, I observed that a filter of size 36 generally gets n-grams from 3–4 different tweets each time. Similarly, the number of filters used (i.e., 64) determines the number of different feature maps relevant for the classification task. Both parameters are determined after extensive experiments conducted over the training set on many 5-fold cross validation runs. To fine-tune these two hyperparameters, I performed a binary search [298, 143] for both, looking in the range values 1–1,024. I discovered that a number of filters greater than 256 increases the overfitting of the model while a filter size greater than 1,024 does not allow the model to reach an accuracy of 1.0 not even on the train fold considered.

### Average and global average pooling

The average pooling layer [280] downsamples the input representation by taking the average value over the window defined by a pool size parameter. The window is shifted by strides. As an example, consider a single dimension array X = [1.0, 2.0, 3.0, 4.0, 5.0]. Defining a 1D-average pooling layer having pool size of 2 and stride of 1 and providing X as input to such a layer, the array Y=[1.5, 2.5, 3.5, 4.5] is returned. In this case too, in the attempt of finding the best value for the hyperparameters of this layer, I performed a binary search and found an optimum value of 8 for the pool size and 1 for the stride. The pool size of 8 represents the number of averaged values outputted from the convolution layer at each step. I suppose that the optimum of 1 as stride value might be maybe due to my tokenization choices.

A final 1D-Global Average Pooling layer is similar to the previously described average pooling one. In this case, it is not the average value over a window of the pool size defined that is returned as output but, instead, a global average along the first dimension from the previous layer outputs. Looking at the Figure 6.1, the output of AveragePooling1D layer is made of 484×64 elements.

### Dense

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

### Model training

The values assigned to the various hyperparameters were originally set taking into account many of the decisions adopted in the studies conducted in [313, 117] and subsequently fine-tuned to improve the accuracy achieved by the model. To initialize the weights of the model, I used a Glorot uniform initializer [134]. The model is compiled with a binary cross entropy loss function; this function calculates loss with respect to two classes (i.e., 0 and 1) as defined in 6.1.

$$Loss_{BCE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n \times \log(h_\theta(x_n)) + (1 - y_n) \times \log(1 - h_\theta(x_n))] \tag{6.1}$$

where:

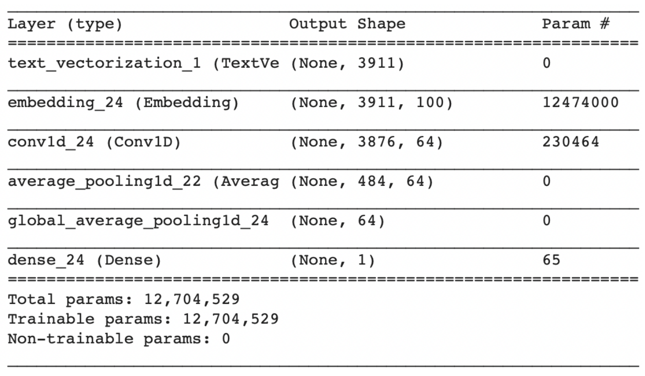
- N is the number of training examples;
- $y_n$ is the target label for the training sample $n$;
- $x_n$ is the input sample $n$;

Table 6.1: Dataset summary showing the number of samples (i.e., authors) for each set, language, and class.

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

- $h_\theta$ is the neural network model with weights $\theta$.

Optimization is performed with an Adamic optimizer [141] after giving each batch of data as input. I performed a binary search for finding the optimal batch size. The model achieved the best overall accuracy with a batch size of 2. The model architecture is depicted in Figure 6.2, where the number of the various network hyperparameters are provided.



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

Figure 6.2: Model representation showing the number of parameters involved at each layer. Such a few parameters allows low computational load for training and testing. Figure as depicted on my Google Colab notebook.

### 6.1.3 Experimental evaluation and results

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

#### Experiments

Table 6.1 reports the number of samples within each set. Each sample in all the sets is an XML file whose name corresponds to the author ID. Each XML file contains 200 tweets of the considered author. Considering that there are 200 tweets per author (the number of authors is shown in Table 6.1), the whole dataset consists of 120,000 tweets.

Table 6.2: The 5-fold splitting applied to the complete multilingual training set. $T$ indicates that this percentage is used for training and $V$ for validation on this fold.

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

Table 6.3: Results achieved by the model on a 5-fold cross validation on the complete multilingual training set (i.e., Spanish and English data). Both loss and accuracy are computed for the validation set used at the fold indicated on the upper row. In the last two columns, I report the values of the arithmetic mean and the standard deviation over the 5 folds.

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

Task organizers invited participants to deploy their models on TIRA[231]. As communicated by email by the task organizers, TIRA has been experiencing technical issues. Therefore, my model was developed and tested as a Jupyter Notebook in Google Colab using TensorFlow. The complete source code is publicly available and reusable.[3]

To validate my model and fine-tune its hyperparameters, I ran 5-fold cross validations for each test performed. I considered the full training set made by the union of both language sets, then I shuffled this multilanguage training set and used it for the cross validations. Then I split 80–20 for each of the fold. Specifically, the first fold was made using the first 80% of the samples for training and the remaining 20% for validation. The remaining folds were made as reported in Table 6.2 with the order of the percentage of samples taken for both sets (train and validation) from the complete training set.

### Results

In Table 6.3 the results obtained adopting a 5-fold cross validation on the complete multilingual training dataset are reported. The 5-folds were made as explained in the previous subsection. Table 6.3 reports accuracy and loss values achieved on the validation set used at each fold, together with the arithmetic mean and standard deviation. For each fold I trained the model for 15 epochs, then I reported the higher accuracy and the related loss over the 15 epochs of training with respect to the validation set used for the fold indicated in the upper row. As can be noted, some splits achieved a better performance, and this could be due to a higher level of similarity between the considered train and validation sets.

Finally, as reported in the PAN website, my model achieved an accuracy of 0.73 on the English

[3]Model notebook: `https://colab.research.google.com/drive/1hUwn_ukOYPC6Tpo3MK1gDVGPQxmzPh_E`.

test set and of 0.85 on the Spanish test set[4]. Considering these results, the overall accuracy (i.e., the arithmetic mean of the accuracy achieved per language) is 0.79.

## 6.2 ELECTRA and graph neural networks to detect harmful tweets

### 6.2.1 Task description

Along to the COVID-19 outbreak, the spread of misleading information online related to news on the pandemic could be observed, for example on social media. The three tasks proposed for the CheckThat! Lab@CLEF2022 [209, 208] aim at addressing related issues, namely: (1) Identifying relevant claims in tweets, (2) Detecting previously fact checked claims and (3) Fake news detection. All tasks are framed as classification problems and build on collective effort to tackle the COVID-19 related infodemic.

The aim of this work is to propose a model to address the first task [207]. Furthermore, this task includes four different subtasks. Subtask 1A is about determining check-worthiness of tweets (i.e., given a tweet, predict whether it is worth fact-checking). Subtask 1B is about verifiable factual claims detection: given a tweet, predict whether it contains a verifiable factual claim. In Subtask 1C, the focus is on harmful tweet detection: given a tweet, predict whether it is harmful to the society and why. Finally, Subtask 1D is related to attention-worthy tweet detection: given a tweet, predict whether it should get the attention of policymakers and why. This task is defined with eight class labels. For the subtasks 1A and 1C the official evaluation metric is the binary F1 score with respect to the positive class, for subtask 1B the metric used is the accuracy and for subtask 1D the metric used is the weighted F1 score.

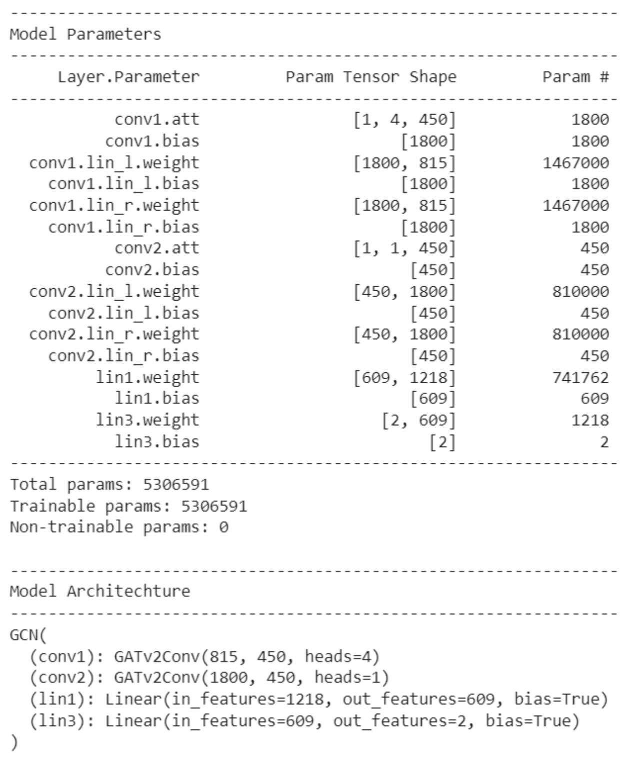
I present the model proposed for Subtask 1C — English language. The model takes advantage of an ELECTRA-based document embedding as well as a text graph that is processed using a Graph Convolutional Network (GCN). The goal is to introduce a novel method that can handle different types of heterogeneous textual or social information. I show how a first version of such a model performs on the proposed task, leaving room for improvements on future research in the domain. To support reproducibility and future research directions, I make my code publicly available[5].

### 6.2.2 Proposed model

The model architecture with input and output shapes of each layer is shown in Figure 6.2.2 along with parameter distributions of each layer. The proposed model is composed by two modules:

[4]Pan 2021 task results: `https://pan.webis.de/clef21/pan21-web/author-profiling.html#results`.
[5]`https://github.com/sagacemente/CLEF2022CheckThat.git`

- Graph creation and embedding
- Pretrained document embedding



```
---------------------------------------------------------------
Model Parameters
---------------------------------------------------------------
     Layer.Parameter        Param Tensor Shape        Param #
---------------------------------------------------------------
           conv1.att             [1, 4, 450]             1800
          conv1.bias                  [1800]             1800
  conv1.lin_l.weight             [1800, 815]          1467000
    conv1.lin_l.bias                  [1800]             1800
  conv1.lin_r.weight             [1800, 815]          1467000
    conv1.lin_r.bias                  [1800]             1800
           conv2.att             [1, 1, 450]              450
          conv2.bias                   [450]              450
  conv2.lin_l.weight             [450, 1800]           810000
    conv2.lin_l.bias                   [450]              450
  conv2.lin_r.weight             [450, 1800]           810000
    conv2.lin_r.bias                   [450]              450
         lin1.weight             [609, 1218]           741762
           lin1.bias                   [609]              609
         lin3.weight                [2, 609]             1218
           lin3.bias                     [2]                2
---------------------------------------------------------------
Total params: 5306591
Trainable params: 5306591
Non-trainable params: 0

---------------------------------------------------------------
Model Architechture
---------------------------------------------------------------
GCN(
  (conv1): GATv2Conv(815, 450, heads=4)
  (conv2): GATv2Conv(1800, 450, heads=1)
  (lin1): Linear(in_features=1218, out_features=609, bias=True)
  (lin3): Linear(in_features=609, out_features=2, bias=True)
)
```

Figure 6.3: Model parameters (Top) numbers in brackets indicate parameters' tensor dimensions; last column indicates the number of parameters in each layer. Model architecture (bottom) model input and output shapes in each layer (figure taken from my Google Colab notebook).

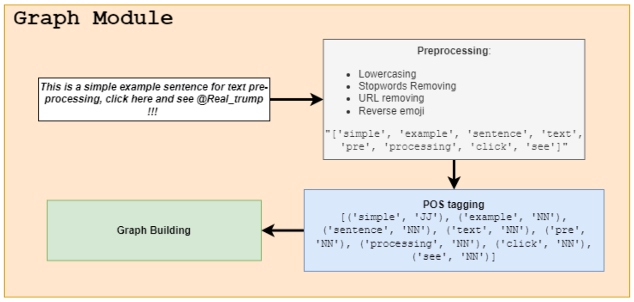
Geometric deep learning [43, 142] has led to a growing number of new architectures as well as novel applications, including text modelling [81]. The representation of each tweet into a graph starts with text preprocessing and POS tagging. After these steps each unique tagged word in the tweet corresponds to a node in the graph and the adjacency matrix is populated connecting each node with all words in a window equal to 3. Each node is annotated with various features, as discussed later. The proposed architecture is composed of two graph attention convolution (i.e., GATV2Conv) layers proposed by [42]; the node-wise representation outputted by the GATV2Conv layers is passed to a max pooling operator and a dropout layer. The output is then concatenated with the document embedding generated using ELECTRA [57]. Finally, two dense layers and a rectified linear unit (ReLu) activation function between them output the predictions of the model for each class.

Before discussing the network architecture as well as the hyperparameter settings, I want to mention that each split of the dataset (training and test per language) consists of individual tweets and their corresponding labels.

### Graph creation

The graph module takes as input a raw sample (tweet) and outputs the tweet represented as an undirected, attributed graph. In Figure 6.2.2 each step of the preprocessing pipeline is depicted. The custom preprocessing function uses the python NLTK package [34]. Below, I list all the preprocessing steps involved:

- *Lowercasing.* This step is used to get the same embedding, e.g., for the words **Hello** and **hello**.
- *Removing stopwords.* Stopwords are generally speaking used with high frequency but they are in many cases not really informative, e.g., preposition and articles belong to this category.
- *POS tagging.* In this step, each word in the tweet is classified into its parts of speech class and labelled accordingly using a one-hot encoding. These vectors correspond to the respective POS tag out of all 43 POS classes in the NLTK package.
- *URL removing.* All URLs in each tweet have been removed.
- *Hashtag symbol and tagged accounts.* All hashtag symbols have been removed along with tagged users.



```
"['simple', 'example', 'sentence', 'text',
'pre', 'processing', 'click', 'see']"
```

```
[('simple', 'JJ'), ('example', 'NN'),
('sentence', 'NN'), ('text', 'NN'), ('pre',
'NN'), ('processing', 'NN'), ('click', 'NN'),
('see', 'NN')]
```

Figure 6.4: Graph representation: each tweet is represented as a graph after pre-processing and POS tagging.

Starting from the output of POS-tagging, I adopt a strategy that associates an edge to each word with all words in a window equal to 3. If a word is repeated more than once, only the first occurrence is considered as node, while edges are updated accordingly. Edges are unweighted.

### Node characterization

As mentioned above, in Figure 6.2.2 each node is characterized as a 815-dimensional vector. The first 768 features correspond to the pretrained ELECTRA document embedding, obtained using

FLAIR, applying the introduced preprocessing steps [6][3]. To select the best embeddings, I evaluated different transformer-based models using the tweet text data as input. The official evaluation metric was used during this experiment, and it turned out that ELECTRA outperformed other pretrained language models. Using ELECTRA, I collected both word embedding for each node in the graph and document embedding for the whole tweet. Each node was also annotated with the corresponding one-hot-encoded vector of its POS tag (45 features). Given the fact that graph networks are order invariant w.r.t the nodes processed during message passing, the order of words in the original tweet is lost. To maintain this information, I characterized each node with a feature vector of two dimensions that encodes the distance from the origin of each node (word) in the graph using sine and cosine positional encoding of a transformer model [286]. This vector is concatenated to the other node features.

#### Graph attention convolution and max pooling layer

In the model, I use two GATV2Conv [42] layers. This layer is characterized by the computation of dynamic attention scores. Moreover, I adopt multiple heads in the first layer where the number of heads is set to four, because (as demonstrated previously by [288]) the learning process can benefit from employing multi-head attention and concatenating their outputs. As highlighted in Figure 6.2.2 the number of features used to represent each node is halved between the 2 layers. The output of the graph attention layer is a 2D matrix with shape: Number of nodes ($d$) * Number of features ($N$). The maximum value is calculated along the dimension of size $N$, in fact reducing the dimension of the input tensor by one.

#### Dense

The max pooling layer's output is concatenated with the tweet embedding obtained from ELECTRA. This vector is fully connected to a dense layer which is followed by a rectified linear unit function element-wise (e.g., $Relu(x) = max(0, x)$) and finally a dense layer with two units as output. This float values correspond to the softmax logits, a vector of raw (non-normalized) predictions.

### 6.2.3 Experimental evaluation

I participated in *Subtask C (harmful tweet detection)* of CheckThat!'s Task 1 for the English language. Before addressing experimental setup and model training, I briefly describe the provided dataset.

[6]Documentation available here: `https://github.com/flairNLP/flair`

Table 6.4: Dataset statistics of all provided splits for English.

| **DS** | **Number of Samples** | Label 0 | Label 1 |
|---|---:|---:|---:|
| Train | 3323 | 3031 | 292 |
| Development | 307 | 276 | 31 |
| Dev-Test | 910 | 828 | 82 |
| Test | 251 | 211 | 40 |

**Dataset**

The corpus includes a list of tweets that are either labelled as harmful (1) or not (0). In addition, the ID of the tweets and its URLs are available. All samples are related to COVID-19. In general, a train, development and dev-test set are provided as well as an official test set that was used for evaluation of the submissions. While for the first three dataset splits the gold labels were available, those of the official test set were held out till the end of the evaluation phase. In general, the number of samples for all parts of the dataset are distributed as shown in table 6.2.3. The data were released as multiple tab-separated files (one per split)

An exploratory analysis shows the dataset imbalance with respect to the class labels. For the training set, the positive class samples correspond to only 8% of the total entries. Using the Tweet IDs provided, I crawled Twitter data via the official Twitter API [7]. However, only a small subset of the samples (w.r.t. the training set only 20% of the original tweets) was still available as the rest of this information was already deleted by Twitter. Given this observation, I choose to discard including social context information such as the number of tweet interaction (favorites, shares), author features (follower following relationships as well as user timeline tweets). Besides using the graph based approach introduced in Section 6.2.2 I performed further experiments on the dataset featuring transformer based methods as well as alternative graph construction techniques. I present details on the results of these experiments in Section 3.4.3.

**Model training**

The hyperparameter settings are in line with many of the decisions made in a study conducted in [297] and used to subsequently fine-tune my proposed model. To initialize the weights of the model, I used a Glorot uniform initializer [134]. The model was compiled using binary cross entropy loss; this function calculates the loss with respect to two classes (i.e., 0 and 1) as defined in 6.2.

$$Loss_{BCE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n \times \log(h_\theta(x_n)) + (1-y_n) \times \log(1-h_\theta(x_n))] \tag{6.2}$$

where:

[7] https://developer.twitter.com/en/docs/twitter-api

- N is the number of training examples;
- $y_n$ is the target label for the training sample $n$;
- $x_n$ is the input sample $n$;
- $h_\theta$ is the neural network model with weights $\theta$.

To improve the model performance and counteract class imbalance, I set up class weights that correspond to a manual rescaling weight assigned to each class. Optimization is performed using the Adam optimizer [141]. To reduce overfitting I take advantage of a learning rate scheduler reducing the learning rate by a factor of 0.9 with a step size of 25 epochs. The model architecture is depicted in figure 6.2.2, where the number of the various network hyperparameters are provided.

### 6.2.4 Results

**Baseline**

The organizers provide official baseline results based on random predictions. The metric to evaluate the task is the binary F1 score of the positive class label (harmful tweet). The official baseline result amounts to 0.200 binary F1 on the evaluation test set. I compare the baseline results to the values my approaches did achieve in Table 6.5.

I established an additional, strong baseline based on a fine-tuned transformer model. I evaluated different transformer architectures including BERT, RoBERTa and ELECTRA regarding the classification task. It turned out that ELECTRA achieves the best results among these models (using 3 epochs for fine-tuning, as recommended by [73]). The performance of this approach amounts to 0.250 binary F1 score.

**The proposed approach**

The main experiments are based on the model proposed in Section 6.1.2. I will report results on the test set used for evaluation using the official binary F1 metric, as well as binary precision and binary recall of the positive class label.

As presented in Table 6.5 the submitted approach (*GCN+ELECTRA*) outperforms the official baseline by 8%. The official baseline approach generates class labels in random order.

Compared to the performance of my own baseline using ELECTRA, the *GCN+ELECTRA* outperforms this approach by 3%. I also evaluated a ELECTRA fine-tuning setup using 50 epochs, resulting in a performance almost as good as the finally submitted approach. However, the high number of epochs lead to strong overfitting on the training data.

In addition, I report results obtained by experimental setups that I evaluated as part of the development process of the submitted approach. In Table 6.5 *GCN+POS w/o word embeddings*

| **Approach** | **Binary Precision** | **Binary Recall** | **Binary F1** |
|---|---|---|---|
| Baseline | 0.200 | 0.200 | 0.200 |
| GCN+3-gram-ELECTRA | 0.138 | 0.625 | 0.226 |
| ELECTRA (3 epochs) | 0.263 | 0.250 | 0.256 |
| GCN+POS w/o word embeddings | 0.166 | 0.650 | 0.264 |
| ELECTRA (50 epochs) | 0.275 | 0.275 | 0.275 |
| GCN+ELECTRA | 0.166 | 0.875 | 0.280 |

Table 6.5: Results (binary Precision, Recall and F1 of the positive class label) on the official test set for English with respect to different approaches.

refers to a setting where I omit word embeddings and represent graph nodes by only considering one-hot encoded POS-tag vectors. In the *GCN+3-gram-ELECTRA* model, I characterize graph nodes by mean-pooled word embeddings of 3 subsequent words at each position. Thus, I also wanted to take into account word orders that can be lost during graph convolution.

## 6.3 Detecting Irony and Stereotype Spreaders on Twitter

### 6.3.1 Task description

The task proposed at PAN@CLEF2022 [31] was about Profiling Irony and Stereotype Spreaders (ISSs) on Twitter [46]. The task was to investigate whether an author of a Twitter feed is likely to spread tweets containing irony and stereotypes. The organizers provided a labelled English dataset, consisting of 420 authors. In the dataset, each sample represents a single author's feed. For each author, a set of 200 tweets is provided. The unlabeled test set provided consists of 180 samples. In the rest of this subsection, I discuss the three models developed to participate at shared task.

### 6.3.2 T100: A modern classic ensamble

The model I used to compete for the task consists of a Logistic Regressor (LR) that get as input the predictions provided by a first stage of classifiers (named *the voters*). The voters are a Convolutional Neural Network (CNN), a Support Vector Machine (SVM), a Naïve Bayes classifier (NB) and a Decision Tree (DT).

The model proposed and described in this section is named T100. This name is motivated by the *modern classic* class of motorcycles produced by the UK-owned manufacturer[8]. In fact, T100 consists of both modern and classic elements to perform its task[9]. T100 include an LR model trained on the predictions provided by a first stage of classifiers. Details about the training phase of T100 are provided in the following subsection.

[8]`https://www.triumphmotorcycles.co.uk/`

[9]...that is TC, not yet able to run at 100 MPH. Yet...

As a first step I preprocess each sample in my dataset to remove information common to all samples. More specifically, I remove the tag CDATA before each tweet of any author's feed. Then I remove the starting tag $< documents >$ opening each sample. Finally, I remove the opening and closing tag $< authorlang = "en" >$. Finally, I lowercase all the text. The resulting text is then vectorized using the Keras Text Vectorization layer[10]. The preprocessing discussed above is performed by the text vectorization layer. Therefore, the text vectorization layer performs the following operations:

1. Preprocess the text of each sample
2. Split the text in each preprocessed sample into words (at each space character)
3. Recombine words into tokens (ngrams)
4. Index tokens (associate a unique int value with each token)
5. Transform each sample, using this index, into a vector of ints.

While the vectorized text is provided as-is to the word embedding layer inside the CNN, another step is performed for other voters. The vectorized text is translated into a Bag-of-Words (BoW) representation and provided as input to the other voters (i.e., NB, SVM and DT).

It is worth noting that the outputs from the first stage of classifiers have different meanings. In fact, the CNN outputs a float value in the range ($-\infty$,$+\infty$), while other classifiers output the probability that a given sample is an ISS. In the case of the CNN the threshold value is set equal to 0, therefore any negative value corresponds to a nISS while a positive one corresponds to an ISS.

The CNN network is implemented accordingly to the work discussed in [259] and in [258]. The network consists of a word embedding layer followed by a convolutional layer, an average pooling layer, a global average pooling layer and a single dense unit as output. The other voters are implemented using the *Scikit-learn* packages[11].

At a very first implementation, I tried to normalize each voter's output. Specifically, I performed several experiments; as an instance, using the normalization techniques discussed in [5, 217]. However, I discovered that keeping the original output range from each voter notably increases the performance of T100. So I lastly did not make use of any kind of normalization technique for any voter's output.

### Model training

The training of my model is based on a 5-fold strategy. As a first step, I train each voter using the k-training fold. Then I let each voter predicts on the corresponding k-validation fold. Then I

[10]`https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization`
[11]`https://scikit-learn.org/stable/`

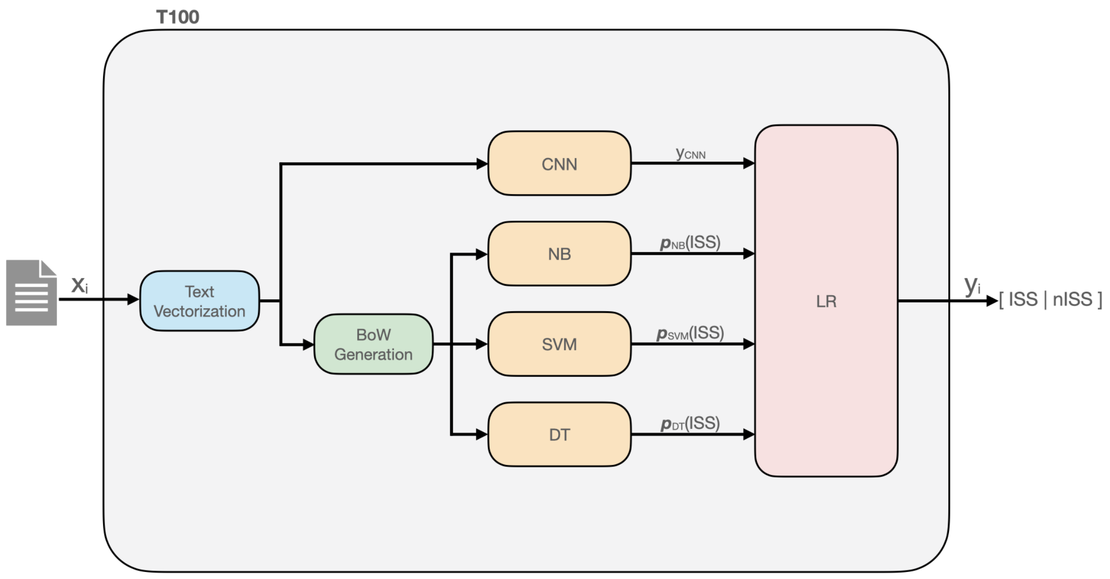


Figure 6.5: The overall architecture of T100. The sample $\mathbf{x}_i$ is the Twitter feed of the i-th author. The shallow CNN used here is built as discussed in [259]. Other classifiers are included into the *Scikit-learn* package. LR uses the predictions provided by the voters to predict the label $\mathbf{y}_i$ corresponding to the input sample $\mathbf{x}_i$.

merge the five sets of predictions on the validation folds. In such a way, a new predictions' dataset is generated. In this new generated predictions dataset, samples consist of voter's predictions and of the original corresponding label (i.e., *nISS* or *ISS*) of the input sample. This new predictions' dataset is used to train the LR.

After the training phase, the simulation phase is performed as follows. Using the official test set, I provide the unlabeled samples to the voters. Predictions of the voters are provided as input to the LR, then I collect and submit the final predictions made by the LR. This last prediction phase is depicted in Figure 6.5.

### Experimental evaluation

The model, developed in TensorFlow, is publicly available as a Jupyter Notebook on GitHub[12]. The architecture of the CNN-based model used in my work is very similar to the one discussed in [259]. It is a shallow CNN compiled with a binary cross entropy loss function; this function calculates loss with respect to two classes (i.e., 0 and 1). Optimization is performed with an Adamic optimizer [141] after giving each batch of data as input. For each fold, I trained the CNN for five epochs. That is motivated by the fact that some overfitting starts after the fifth epoch. I performed a binary search to find the optimal batch size. The model achieved the best overall accuracy with a batch size equal to 1. For the NB voter, I use *MultinomialNB* from the Scikit-learn package. The SVM voter uses a linear kernel with a C-value equal to 0.5. Finally, for the DT classifier, I set a *random_state* equal to 0.

The dataset provided by the PAN organizers consists of a set of 600 Twitter authors. For each author, a set of 200 tweets is provided. A single XML file corresponds to an author and contains 200 tweets of the author. The labelled training set provided by the organizers contains 420 authors. The test set consists of the remaining 180 ones. Authors in the training set are labelled as "I" (ISS) or "NI" (nISS). my final submission consists of a zip file containing predictions for each non-labelled author in the test set.

### Results

The official metric used for the author profiling task at PAN@CLEF2022 is the accuracy. Before performing the 5-fold cross validation, I shuffled the 420 labelled samples, and then I left out the last 40 samples as a labelled test set. In Table 6.6 are reported the results obtained by the single voters both on the test set and adopting a 5-fold cross validation on the labelled training set. In the table are reported the arithmetic mean and the standard deviation over the 5-folds. Table 6.7 reports the results of T100 on the validation set at each fold and on the labelled 40 samples I used as a test set. In terms of accuracy, each classifier used individually performs worse than T100. Furthermore, the

[12]`https://github.com/marco-siino/T100-PAN2022`

Table 6.6: Results in terms of accuracy achieved by each voter of T100 at each fold. Models are evaluated on the corresponding validation set at each fold and on the same test set. Performance of the classifiers at the first stage of T100 are lower compared to the ensemble model presented here. In the last two columns, I report the values of the arithmetic mean and the standard deviation over the five folds.

| Voter | Set | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| CNN | Val | 0.8947 | 0.8684 | 0.9079 | 0.8684 | 0.8947 | 0.8868 | 0.0158 |
| | Test | 0.9000 | 0.8750 | 0.9250 | 0.9250 | 0.9500 | 0.9150 | 0.0255 |
| NB | Val | 0.8947 | 0.8553 | 0.8816 | 0.8289 | 0.8289 | 0.8579 | 0.0268 |
| | Test | 0.9000 | 0.9000 | 0.9000 | 0.8750 | 0.8750 | 0.8900 | 0.0122 |
| SVM | Val | 0.9210 | 0.9342 | 0.9079 | 0.8816 | 0.8947 | 0.9079 | 0.0186 |
| | Test | 0.8750 | 0.8500 | 0.8750 | 0.8750 | 0.8500 | 0.8650 | 0.0122 |
| DT | Val | 0.7368 | 0.8421 | 0.8684 | 0.7631 | 0.8816 | 0.8184 | 0.0579 |
| | Test | 0.7750 | 0.8000 | 0.7500 | 0.8500 | 0.8750 | 0.8100 | 0.0464 |

Table 6.7: Results achieved by the model on a 5-fold cross validation on the training set provided. The results shown in the table are obtained using a Logistic Regressor as a final classifier of T100.

| T100 — Logistic Regressor | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| Val | 0.9210 | 0.9342 | 0.9342 | 0.8553 | 0.9342 | 0.9158 | 0.0307 |
| Test | 0.9250 | 0.9250 | 0.9250 | 0.9250 | 0.9250 | 0.9250 | 0.0000 |

standard deviation of the single voters and of T100 is comparable on the validation sets. However, the standard deviation is equal to 0 on the test set for T100 and higher for the single voters.

I performed several tests to investigate the best classifier as the very last predictor of T100. From Table 6.8 to Table 6.10 these results are reported.

How it is shown in the tables, the LR is consistent over different training fold, with a null standard deviation on the test set. In terms of consistency, the Gradient Boosting Classifier performs similarly, with a standard deviation of 0.010. However, results in terms of binary accuracy are poor using Gradient Boosting Classifier as long as the other models tested.

Finally, I used the T100 trained at the fifth fold to generate the predictions on the official unlabeled test set provided by the organizers. As announced by the organizers, such a final version of my model is able to reach an accuracy of 0.9444 with respect to the official test set.

Table 6.8: Results achieved by a T100 ensemble using a Decision Tree at the final prediction stage.

| T100 — Decision Tree | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| Val | 0.8421 | 0.8158 | 0.8947 | 0.8421 | 0.8158 | 0.8421 | 0.0288 |
| Test | 0.9000 | 0.8000 | 0.8500 | 0.8250 | 0.8500 | 0.8450 | 0.0331 |

Table 6.9: Results achieved by a T100 ensemble using a Random Forest at the final prediction stage.

| T100 — Random Forest | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| Val | 0.9079 | 0.9342 | 0.9210 | 0.8816 | 0.9210 | 0.9131 | 0.0178 |
| Test | 0.8750 | 0.9000 | 0.9000 | 0.8750 | 0.8750 | 0.8850 | 0.0122 |

Table 6.10: Results achieved by a T100 ensemble using a Gradient Boosting Classifier at the final prediction stage.

| T100 — Gradient Boosting | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| Val | 0.8816 | 0.9079 | 0.9210 | 0.8684 | 0.9210 | 0.9000 | 0.0214 |
| Test | 0.8750 | 0.8500 | 0.8500 | 0.8500 | 0.8500 | 0.8550 | 0.0100 |

### 6.3.3 A CNN-based model using data augmentation with back translation

Before the data augmentation layer, I preprocess the dataset to remove useless information. More specifically, I remove the opening tag *CDATA* from every XML file. Then I removed the starting tag $< documents >$ opening each sample. Finally, I removed the opening and closing tag $< authorlang = "en" >$ and I lowercased all the text.

#### Dataset augmentation

In Figure 6.6 is shown the overview of my proposed framework. The very first stage takes a sample from the dataset. The sample is preprocessed as described in the previous section. Such a preprocessed sample is then augmented. To perform the augmentation, my framework back-translates each sample. I implement this operation, within my framework, performing an online request to the *Google Translator* API from the *deep_translator* library. Full documentation of this module is available online[13]. Thanks to this module, I translate each sample in dataset from English to Italian. Then I translate back from Italian to English and finally, I merge the original sample with the back-translated one.

The rationale behind such an augmentation strategy is easily explainable with the following running example. In the example, making use of the Google Translate Tool online, a tweet contained in one of the samples from the provided dataset is translated and then back-translated.

- ORIGINAL (ENG): *"Yeah; on paper, kinda shitty business model expecting banks et al to buy something they have never needed, and never will need. But I know the real business model is extracting fiat cash from morons; it seems to be playing out swimingly."*
- TRANSLATED (ENG to IT): *"Sì; sulla carta, un modello di business di merda che prevede*

[13] `https://deep-translator.readthedocs.io/en/latest/`

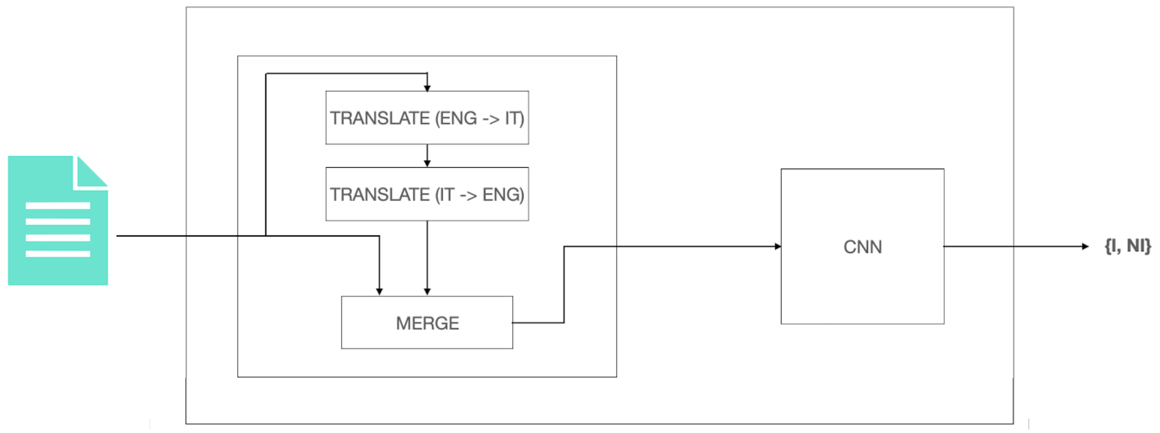


Figure 6.6: Overall architecture of the proposed framework. Both for training and testing phase, the CNN proposed in [259] operates on an augmented dataset built as shown in Figure. Each sample is augmented back-translating the contained text and merging it with the original source.

*che le banche e altri acquistino qualcosa di cui non hanno mai avuto bisogno e di cui non avranno mai bisogno. Ma sappiamo che il vero modello di business è estrarre denaro fiat dagli idioti; sembra che stia nuotando."*

- TRANSLATED (IT to ENG): *"Yup; on paper, a shitty business model that requires banks and others to buy something they never needed and never will. But I know that the real business model is to extract fiat money from idiots; it looks like it is swimming."*

As can be seen from my running example from the dataset provided, the process of back-translating a sample replace, delete and insert words, in fact augmenting the information available for the subsequent CNN-based model. It is worth reporting that while maintaining the characteristics of ironic text (in the first part of an augmented sample), an increased number of words and sentences could provide more information available for the training of the CNN.

In my experiments, I also used back-translation to generate a separate sample for each original sample in the training set. However, there was no improvement in terms of accuracy over the five-fold validation.

### Model training

The architecture of the CNN-based model used in my work is very similar to the one discussed in [259]. It is a shallow CNN, as depicted in Figure 6.1.

The model is compiled with a binary cross entropy loss function; this function calculates loss with respect to two classes (i.e., 0 and 1) as defined in 6.3.3. These classes are obtained after threshold the output of the last single dense unit of the CNN. Positive values as output are considered as ISS

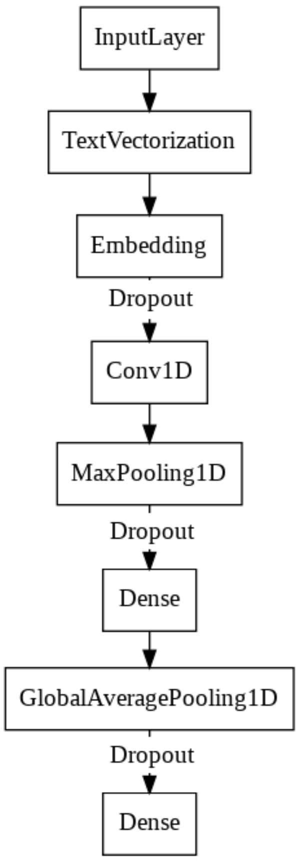


Figure 6.7: The shallow CNN used and discussed in [259].

Table 6.11: Results achieved by the model on a 5-fold cross validation on the training set provided. In this case the DA layer is used before using each augmented sample as input for the CNN.

| | Fold Nr. | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | AVG | $\sigma$ |
| Accuracy | 0.8947 | 0.8684 | 0.9211 | 0.8684 | 0.9079 | 0.8921 | 0.0210 |
| Loss | 0.3322 | 0.2710 | 0.3024 | 0.4102 | 0.2129 | 0.3057 | 0.0655 |

(i.e., 1) and negative ones as nISS (i.e., 0).

$$\text{Loss}_{BCE} = -\frac{1}{N}\sum_{n=1}^{N}[y_n \times \log(h_\theta(x_n)) + (1-y_n) \times \log(1-h_\theta(x_n))] \tag{6.3}$$

Optimization is performed with an Adamic optimizer [141] after giving each batch of data as input. I performed a binary search for finding the optimal batch size. The model achieved the best overall accuracy with a batch size of 1. my model, developed in TensorFlow, is publicly available as a Jupyter Notebook on GitHub.

#### Results

In Table 6.11 are reported the results obtained adopting a 5-fold cross validation. The table reports accuracy and loss values achieved on the validation set used at each fold, together with the arithmetic mean and standard deviation. For each fold, I trained the model for 5 epochs. That is motivated by the start of overfitting by the sixth epoch. In the same table, the higher accuracies and the related losses are shown over the training epochs, with respect to the validation set used at the fold indicated. As can be noted, some splits achieved a better performance, and this could be due to a higher level of similarity between the considered train and validation sets. The model trained on the best fold was used to make predictions on the test set provided for the competition. Predictions were uploaded on TIRA [231]. As reported in the final ranking[14], the proposed model (namely, *stm*) reaches an accuracy of 0.9278.

### 6.3.4 A simple SVM ensemble

#### The proposed model

The proposed model is shown in Figure 6.8. After a *Text Vectorization*[15] layer, I provided the tokenized text to the CNN, Naive Bayes and Decision Tree classifiers.

The Naive Bayes and Decision Tree are implemented using the *Scikit-learn* package, while for

[14]https://pan.webis.de/clef22/pan22-web/author-profiling.html

[15]https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization

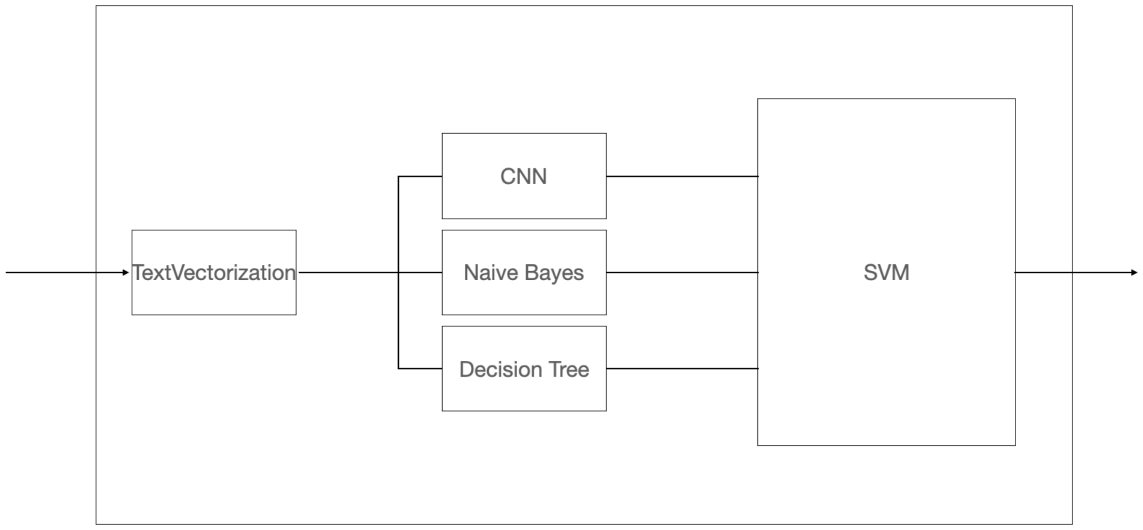


Figure 6.8: Overview of the proposed ensemble model.

the CNN I implemented the shallow network discussed in [259].

After I collected the prediction from CNN, NB and DT on each sample of the dataset, I provided these predictions as input to an SVM. Same pipeline is implemented both for training and testing phase of my model. During the training phase, I provide predictions related labels to the SVM. For the test phase, I provided the unlabeled sample from the test set to the CNN, NB and the DT. Providing the output predictions from these classifiers as input to the SVM, I collected the final prediction to be submitted to TIRA.

### Experimental setup

I developed my software using the Python language (version 3.7) on Google Colab[16]. To build my models I mainly used the Scikit-learn[17] package, NumPy[18] and TensorFlow[19]. My code is available in Google Drive as a Jupyter Notebook[20].

### Results

In Table 6.12 are shown the results obtained by the single classifiers used and by the SVM ensemble on the best running fold over a 5-fold cross validation. Results of the SVM are obtained using as samples the predictions of the first layer classifier over the five folds.

[16]https://colab.research.google.com/
[17]https://scikit-learn.org/
[18]https://numpy.org/
[19]https://www.tensorflow.org
[20]https://colab.research.google.com/drive/1EWCxAHxWWAkFg-Y8dveXuxrh82hyOh96?usp=sharing

Table 6.12: Results of single classifiers over 5-fold. The best results over 5-fold are expressed in terms of binary accuracy. The standard deviation over the 5-fold is shown in the latest column.

| Model | Accuracy | $\sigma$ |
|---|---|---|
| CNN | 0.9079 | **0.0158** |
| NB | 0.8947 | 0.0268 |
| DT | 0.8816 | 0.0579 |
| SVM (ensemble) | **0.9474** | 0.0377 |

As can be noted, the performance of the SVM ensemble significantly outperforms single classifiers within my proposed framework. However, the standard deviation over the five folds is not smaller with regard to the CNN and Naive Bayes. As communicated by the organizers, on the test set provided, my model is able to reach an accuracy of 0.9389.

## 6.4 Detection and categorization of PCL

With the exponential growth of contents shared on social networks, a lot of new challenging tasks have emerged. Many are currently studied and addressed by scholars, and a plethora of novel machine learning approaches have been proposed [16], [111], [257]. Some of the most common tasks, often co-located with international conferences, are those about fake news [216], hate speech [39], misogyny [85] and cyberbullying [151] detection.

For these purposes, there is a constantly growing need for tools that can automatically extract and classify information from online feeds, to face with consolidated as well as with emerging social issues. Interest in NLP has increased in recent years with advances in machine and deep learning architectures. There have been significant efforts in developing methods to automatically detect and classify text content available online nowadays.

Together with the already mentioned tasks, an emerging one is about detecting Patronizing and Condescending Language (PCL) [224]. The PCL Detection Task hosted at SemEval-2022 is covered in detail in [225] and briefly discussed here. The main task is made of two subtasks. The first one is a binary classification problem where, given a paragraph, a model has to predict whether the paragraph contains or not PCL. The second one is a multi-label classification task where each paragraph has to be labelled with one to seven categories of PCL. Classes are not mutually exclusive, and so a paragraph could express one or more categories of PCL.

To face with the first subtask, I propose two deep models. The first one is a multichannel CNN. Such a network consists of parallel word embedding and convolutional layers to allow different sets of weights for trained embeddings — because of different kernel sizes employed by convolutional layers. In terms of *Precision*, *Recall* and *F1*, results of my model show certain room for improvements in future work. The second model is a hybrid bidirectional LSTM. Such a network is composed by a convolutional layer and two bidirectional LSTM layers.

For the second subtask, I propose two Transformer-based models [286]. The first one is a lighter and faster version of BERT (i.e., DistilBERT) [250]. The model is opportunistically trained on an undersampled version of the training dataset. The model is able to outperform RoBERTa [173]. The second is an XLNet-based one [304]. The model is based on a generalized autoregressive pretraining method. It enables learning bidirectional contexts by maximizing the expected likelihood over all permutations of the factorization order. Under comparable experiment setting, XLNet outperforms BERT [73] on several tasks, often by a large margin, including question answering, natural language inference, sentiment analysis, and document ranking. My model implementation is opportunistically trained on an undersampled version of the training DS. The model is able to outperform RoBERTa [173] in terms of average F1.

### 6.4.1 Background

In this section, I provide some background about the Task 4 hosted at SemEval-2022. The aim of this task is to identify PCL, and to categorize the linguistic techniques used to express it, specifically when referring to communities identified as being vulnerable to unfair treatment by the media. Participants at the Task 4 received a dataset with sentences in context (paragraphs), extracted from news articles. Although news articles were collected from different countries, they were all provided in English. The task consists of the two subtasks listed below.

1. Subtask 1: Binary classification. Given a paragraph, a system must predict whether it contains any form of PCL. Two opposite labelled samples from the dataset provided are shown below.

   **Non-PCL Sample Text:** *"Council customers only signs would be displayed . Two of the spaces would be reserved for disabled persons and there would be five P30 spaces and eight P60 ones ."*

   **Non-PCL Sample Label:** [0]

   **PCL Sample Text:** *"It can not be right to allow homes to sit empty while many struggle to find somewhere to live, others having to sleep rough on pavements during Christmas, hoping against hope, for some charity to provide shelter. The number left homeless and destitute is alarming not necessarily at Christmas?"*

   **PCL Sample Label:** [1]

2. Subtask 2: Multi-label classification. Given a paragraph, a system must identify which PCL categories express the condescension. The PCL taxonomy has been defined based on previous works on PCL. The proposed categories are:

- Unbalanced power relations
- Shallow solution
- Presupposition
- Authority voice
- Metaphor
- Compassion
- The poorer, the merrier

Two samples from the dataset provided are shown below. For each sample, the label is an array containing seven elements. For each element, symbol *1* means that the corresponding PCL category is expressed in the paragraph.

**Sample Text 1:** *"Yes ... because there is NO HOPE where he lives . India is a third-world country . Do n't be fooled by call centers in big cities . Most of the country is rural and most of the population is illiterate and hopeless ."*

**Sample Label 1:** [1, 0, 1, 0, 0, 1, 0]

**Sample Text 2:** *"For refugees begging for new life , Christmas sentiment is a luxury most of them could n't afford to expect under the shadow of long-running conflicts."*

**Sample Label 2:** [0, 0, 1, 0, 0, 1, 0]

Task organizers released a training and a dev set before the competition officially started. For both sets, the gold labels were provided. During the first phase — Practice phase — participants were able to develop and test their models, uploading predictions on Coda Lab. After releasing the unlabeled test set, the second phase — Evaluation phase — started. Results for both phases are available online [21].

### 6.4.2 System overview

In this section, I discuss the models presented for each subtask and the design choices made by my team, motivating them. For both models, the code is publicly available and reusable. Further details are provided in Section 3.4.2.

#### Subtask 1: Binary Classification

Given the binary nature of the task and his subject, for my first submission I developed a more versatile CNN based on the one presented in [259]. Such a network is composed of parallel word

[21]`https://sites.google.com/view/pcl-detection-semeval2022/ranking`

embedding and convolutional layers to allow different weights for embeddings and convolutional filters. A general overview of the model architecture is shown in Figure 6.9. The rationale of the model presented is to have more parallel convolutional-based channel, each with different word embeddings and kernel filter weights. More properly, I set kernel size of 1, 2, 16 and 32 for each of the 32 *Conv1D* layer filters. In this way I drive my model to focus more on a single token, a pair of tokens, a group of 16 and of 32 tokens respectively. On the basis of my experiments, these are the best-performing kernel sizes for the proposed task on my preliminary 10 cross-fold validation. In addition to this behavior, I expect different coordinates for each word/token in each word embedding channel, with the aim of getting a more fine-grained positioning of words/tokens in the embedding space.

Based on my preliminary experiments, I found that on five different seeds initialization, the best word embedding size for my model is 50. This size is consistent with the common values reported in literature [194]. For each dense layer, I did not use any activation function. I trained my model with a binary cross-entropy loss and using the Adam optimization algorithm [141].

For my second submission, I developed a light Hybrid LSTM. The model consists of a convolutional layer followed by two bidirectional LSTM layers. Such a strategy is motivated by my decision to extract relevant features from the word embedding layer before the first bidirectional one. A general overview of the model architecture is shown in Figure 6.10. Based on my preliminary experiments on five different seeds initialization, I found that the best word embedding size for the model was 50. For each dense layer, I did not use any activation function. I trained my model with a binary cross-entropy loss and using the Adam optimization algorithm [141].

### Subtask 2: Multi-Label Classification

For my first submission at the Subtask 2 I chose a transformer-based model lighter than BERT (i.e., DistilBERT). Due to the high number of experiments to perform, I needed a faster model to train. DistilBERT is a smaller general-purpose language representation model. In DistilBERT the original size of BERT model is reduced by 40%, while retaining 97% of its language understanding capabilities and being 60% faster. In terms of knowledge distillation, while BERT is the teacher, DistilBERT is the student. The student is represented by a compact model and is trained to reproduce the behavior of the larger model (i.e., the teacher). Such a compact model is trained with a linear combination of three losses: the *distillation loss* (i.e., $L_{ce}$), the *masked language modeling loss* (i.e., $L_{mlm}$), and the *cosine embedding loss* (i.e., $L_{cos}$). Because of the distilled nature of the model, training and fine-tuning on a specific dataset for a specific task is of prominent importance. For a detailed discussion of DistilBERT refer to [250]. While I firstly compared the results on the dev set provided, I finally trained my model on the full training set — union of train and dev set — providing predictions on the test set. In addition, I found beneficial maintaining the information about casing of characters. So I did not lowercase the text provided, implementing a cased version

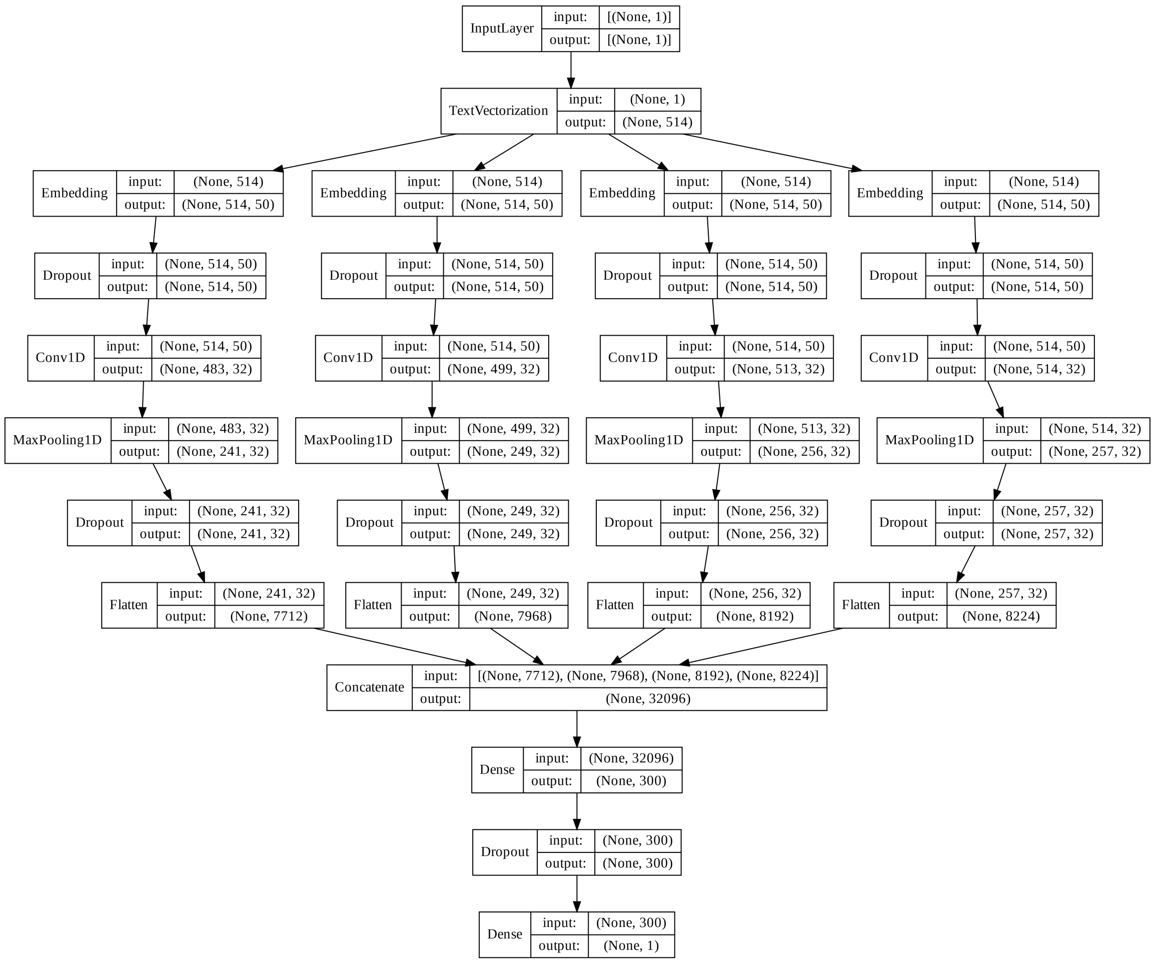

InputLayer input: [(None, 1)] output: [(None, 1)]
TextVectorization input: (None, 1) output: (None, 514)
Embedding input: (None, 514) output: (None, 514, 50)
Dropout input: (None, 514, 50) output: (None, 514, 50)
Conv1D input: (None, 514, 50) output: (None, 483, 32)
Conv1D input: (None, 514, 50) output: (None, 499, 32)
Conv1D input: (None, 514, 50) output: (None, 513, 32)
Conv1D input: (None, 514, 50) output: (None, 514, 32)
MaxPooling1D input: (None, 483, 32) output: (None, 241, 32)
MaxPooling1D input: (None, 499, 32) output: (None, 249, 32)
MaxPooling1D input: (None, 513, 32) output: (None, 256, 32)
MaxPooling1D input: (None, 514, 32) output: (None, 257, 32)
Dropout input: (None, 241, 32) output: (None, 241, 32)
Dropout input: (None, 249, 32) output: (None, 249, 32)
Dropout input: (None, 256, 32) output: (None, 256, 32)
Dropout input: (None, 257, 32) output: (None, 257, 32)
Flatten input: (None, 241, 32) output: (None, 7712)
Flatten input: (None, 249, 32) output: (None, 7968)
Flatten input: (None, 256, 32) output: (None, 8192)
Flatten input: (None, 257, 32) output: (None, 8224)
Concatenate input: [(None, 7712), (None, 7968), (None, 8192), (None, 8224)] output: (None, 32096)
Dense input: (None, 32096) output: (None, 300)
Dropout input: (None, 300) output: (None, 300)
Dense input: (None, 300) output: (None, 1)

Figure 6.9: Overview of the multichannel CNN presented for the first subtask at SemEval-2022. Each channel has a different kernel size at *Conv1D*, driving model attention on different sized windows of words. The kernel size of filters used at each *Conv1D* are 1, 2, 16 and 32. Each convolutional layer has 32 filters separately trained during training phase. Such a strategy allows extraction of different-sized features for a fine-grained learning.

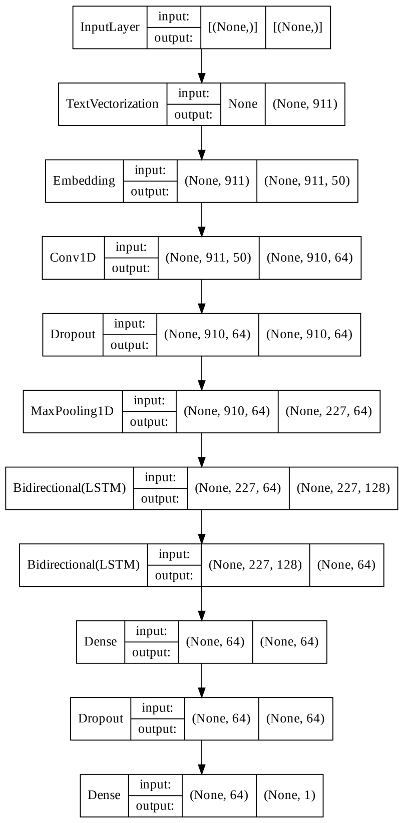


Figure 6.10: Overview of the Hybrid LSTM presented for first subtask hosted at SemEval-2022. The presence of the *Conv1D* layer is motivated by my intention to extract relevant features from the previous embedding layer. The kernel size of the 64 filters used by the convolutional layer is 2. Such a strategy should allow extraction of relevant bi-grams from the input text.

of DistilBERT and setting as output for each label seven digits corresponding to the seven categories of PCL. Finally, I preprocessed each sample to include the country and keyword of each paragraph in the input text.

For the second submission, I implemented an XLNet-based model. Different unsupervised pre-training objectives have been explored in literature. XLNet implements a generalized autoregressive pretraining method that uses a permutation language modeling objective to combine the advantages of autoregressive and autoencoding methods. The neural architecture of XLNet is developed to work seamlessly with the autoregressive objective, including the integration of Transformer-XL and the careful design of the two-stream attention mechanism. XLNet achieves substantial improvement over previous pretraining objectives on various tasks. Among them, autoregressive language modeling and autoencoding have been the two most successful pretraining objectives. Furthermore, XLNet integrates ideas from Transformer-XL [66] into pretraining. An XLNet model integrates two techniques from Transformer-XL, namely the relative positional encoding scheme and the segment recurrence mechanism. The relative positional encodings are applied based on the original sequence. Furthermore, the recurrence mechanism is included into the proposed permutation setting and enable the model to reuse hidden states from previous segments.

Training and fine-tuning of an XLNet for a specific task is of prominent importance. While I firstly compared the results on the dev set provided, I finally trained my model on the full training set — e.g., union of train and dev set — providing predictions on the test set.

### 6.4.3 Experimental setup

I implemented my first two models using Keras[22] and TensorFlow[23]. The dataset provided for the binary classification task is unbalanced in terms of negative and positive PCL instances. To face with this issue, I undersampled the negative instances. On the basis of my preliminary experiments, I found beneficial undersampling negative instances to be just six times more the positive ones. Furthermore, I found it beneficial to include in each sample (both for training and prediction) the keyword and the country field of each paragraph from the dataset. Then I used a batch size of 100. I empirically found that a good early stopping point for the training phase is obtained with 10 epochs and a learning rate of 0.001. I ran the experiments on Google Colab using the default GPU (NVIDIA Tesla K80). The training time was around 15 seconds for each of the ten epochs. The official metrics used for the task were Precision, Recall and F1 on positive instances (sample containing PCL). But during my development phase, I focused on the model loss (i.e., *binary cross entropy loss*). This chooses was dictated by the fact that the gold labels of the test set were not provided.

[22]https://keras.io/
[23]https://www.tensorflow.org/

The models for Subtask 2 were implemented using Simple Transformers[24]. I used DistilBERT and XLNet as the pre-trained language models. I preprocessed the dataset to include, within the text of each sample, the country, and the keyword of the paragraph. To train my final models, I built a single dataset consisting of the train and the dev set. Then I undersampled negative instances (i.e., Non-PCL samples) to alleviate bias in the unbalanced dataset provided. I ran the experiments on Google Colab, using an NVIDIA Tesla K80 GPU. The official metrics used for the task were F1 for each category and average F1 among them. In this case too, during my development phase, I focused only on the loss of the models to perform some fine-tuning.

### 6.4.4 Results

For Subtask 1 the metrics used are Precision, Recall and F1. Each *True Positive* (TP) is computed on the positive instances (i.e., paragraphs containing PCL). So a TP is a sample containing PCL and correctly classified, a *False Positive* (FP) is a sample without PCL but wrongly classified as a PCL sample, a *False Negative* (FN) is a sample containing PCL but wrongly classified as not containing PCL. Therefore, Precision is the number of the correctly predicted PCL samples over the total number of predicted PCL samples. Recall is the number of the correctly predicted PCL samples over the total number of actual PCL samples. Finally, the F1 Score is the harmonic mean of Precision and Recall.

The final ranking for the first subtask is drawn up accordingly to the F1 score on the test set provided.

In Table 6.13 are shown the results on the dev set provided by the organizers. Results are ordered according to F1 score. my model based on multichannel CNN is able to outperform the Random-baseline provided in terms of F1 and Precision, obtaining similar results in terms of Recall. RoBERTa-baseline performs better along the three metrics provided.

It is interesting to note that RoBERTa is a model pre-trained on over 160 GB of text. Compared to my proposed model, it requires much more in terms of resources and time needed. Despite such efforts, RoBERTa outperforms my model by only 16% and around 11% in terms of F1 and Precision. The most significant difference is with recall. This means that the proportion of actual positives identified correctly by my model is lower compared to RoBERTa. This could be mainly due to the inability of my model at contrasting the bias learned because of the unbalanced dataset provided, where Non-PCL paragraphs are, in fact, the vast majority. Our team did an additional submission involving two deep models based on a Hybrid LSTM (i.e., made of convolutional and bidirectional LSTM layers) and on an XLNet [304]. Our proposed Hybrid LSTM is able to outperform the Random-baseline provided in terms of F1 and precision. RoBERTa-baseline performs

[24]`https://github.com/ThilinaRajapakse/simpletransformers`

Table 6.13: Performance comparison on dev set. The results of the two baseline methods provided by the organizers (i.e., RoBERTa and Random baseline) compared to my models based on a multi-channel CNN and Hybrid LSTM.

| | F1 | P | R |
|---|---|---|---|
| RoBERTa-baseline | 48.29 | 34.99 | 77.89 |
| Multi-Channel CNN | 32.29 | 23.46 | 51.76 |
| Hybrid LSTM | 26.32 | 31.47 | 22.61 |
| Random-baseline | 17.35 | 10.40 | 52.26 |

Table 6.14: Performance comparison on test set. In the table are shown the RoBERTa-baseline, the first classified (i.e., *hudou*), the two last classified and my models results. In parentheses are shown the positions in the final ranking according to F1 score. *NA* stands for *Not Assigned* because only the best result of the two model submitted is considered for final ranking.

| | F1 | P | R |
|---|---|---|---|
| hudou (1) | 65.10 | 64.60 | 65.62 |
| RoBERTa (44) | 49.11 | 39.35 | 65.30 |
| Multi-CNN (69) | 29.28 | 23.40 | 39.12 |
| Hybrid LSTM (*NA*) | 28.15 | 29.62 | 26.81 |
| mahangchao (79) | 4.48 | 10.59 | 2.84 |
| makahleh (80) | 0.0 | 0.0 | 0.0 |

better along the three metrics provided. Compared to the Hybrid LSTM model, the multichannel CNN outperforms the Hybrid LSTM. However, the Hybrid LSTM performs better with regard to precision. Such a result leads to the conclusion that Hybrid LSTM correctly predicts a higher number of actual PCL paragraphs with respect to the total predicted PCL paragraphs. Therefore, further investigation might be conducted on combinations of main components of the two proposed models in the effort to improve the F1.

In Table 6.14 are shown the results on the test set provided by the organizers without the gold labels. Results are ordered based on the F1 score. Compared to the winner (i.e., *hudou*), RoBERTa exhibits the most significant gap in Precision. Which means that proportion of positive instances correctly classified by the winner team is significantly more compared to RoBERTa. However, in this case too, RoBERTa outperforms my model with a similar gap along the three metrics with respect to the results presented for the dev set. My two submitted models exhibit similar performances on the test set. In this case too, the most significant gap is in recall.

For Subtask 2 the metric used is F1 along the seven categories provided, and the final ranking was drawn up considering the average F1 along the seven categories on the test set provided. For this subtask, there is an important bias due to the unbalanced nature of the dataset with regard to each category. In Table 6.15*(a)* the results on the dev set are shown. Results are ordered based on the average F1 score. For each category, my XLNet is able to outperform the Random-baseline. The average F1 is 15% more than such a baseline. It is worth noting that results with a random predictor are not uniformly distributed along each category. This distribution provides further evidences about the unbalanced nature of the dataset with regard to this multi-label classification subtask. Furthermore, the random predictor outperforms F1 score of RoBERTa in four of the seven categories provided. However, RoBERTa performs a lot better in detecting *Unb*, *Pre* and *Com* language (namely, *Unbalanced power relations*, *Presupposition* and *Compassion*). These performances could be motivated by the greater number of samples in the dataset expressing the first category. Compared to RoBERTa my DistilBERT-model does better for five categories out of seven. And for this single category (i.e., *Presupposition*) the gap is under 4%. Compared to my other submission, the XLNet heavily outperforms DistilBERT in terms of F1 for each category and in the final average F1. In Table 6.15*(b)* I report the results of the first model, my proposed models, RoBERTa and the last classified one, according to the final ranking drawn up considering the average F1. In this case too my models outperform RoBERTa, in terms of F1, for six out of seven categories. On the test set, RoBERTa performs better in detecting *Unb*. However, compared to the results on dev set, my two proposed models perform with a lower average F1 gap. And there is just a category (i.e., *Metaphor*) where DistilBERT significantly outperforms the XLNet. It is worth noting that the best performing model is able to reach an average F1 of 46.89, outperforming of over 20% and 36% my proposed models and RoBERTa respectively. This lead to a conclusion about the very large room for improvement in this multi-label task. Some difficulties in reaching an average F1 of at least 50% could be due to the unbalanced dataset as much as the intrinsic complexity of the task.

## 6.5 Conclusion

What emerges from the results presented in this chapter is that the choice of model is strongly correlated to the type of the evaluated task. More precisely, to the type of input data considered. While the CNN-based shallow architecture has been shown to classify effectively in the case of HSS, similar results have not been obtained in the case of ISS. However, from a textual point of view, the two datasets had the same format. In both cases, a Tweet feed represented the author to be rated. Analyzing the results presented in [256], it emerges that a simple CNN architecture trained from scratch actually separates at the word level already at the embedding level, identifying different vector spaces for the two available classes. It is therefore possible that in the case of ISS, this separation could not be carried out with performances comparable to those obtained with the HSS dataset.

| | Unb | Sha | Pre | Aut | Met | Com | The | **AVG** |
|---|---|---|---|---|---|---|---|---|
| XLNet | 47.99 | 20.41 | 24.61 | 20.06 | 16.67 | 39.24 | 8.89 | 25.41 |
| DistilBERT | 47.60 | 15.90 | 23.84 | 15.53 | 10.91 | 31.23 | 0.0 | 20.72 |
| RoBERTa-baseline | 35.35 | 0.0 | 29.63 | 0.0 | 0.0 | 28.78 | 0.0 | 13.40 |
| Random-baseline | 11.30 | 3.23 | 5.09 | 3.22 | 6.04 | 8.21 | 1.31 | 5.48 |

*(a)*

Table 6.15: Performance comparison on dev set *(a)* and test set *(b)* for Subtask 2. The table shows F1 calculated for each category and the average F1 in the last column. For Subtask 2 my proposed models based on DistilBERT and XLNet outperform RoBERTa on both dev and test set. In parentheses are shown positions in final ranking. NA stands for *Not Assigned* in this case too.

| | Unb | Sha | Pre | Aut | Met | Com | The | **AVG** |
|---|---|---|---|---|---|---|---|---|
| guonihe (1) | 65.60 | 52.94 | 36.90 | 40.66 | 35.90 | 49.18 | 47.06 | 46.89 |
| XLNet (29) | 32.32 | 32.93 | 19.18 | 20.55 | 22.22 | 26.35 | 7.14 | 22.96 |
| DistilBERT (*NA*) | 32.62 | 30.49 | 18.80 | 18.31 | 26.00 | 25.37 | 0.0 | 21.65 |
| RoBERTa-baseline (37) | 35.35 | 0.0 | 16.67 | 0.0 | 0.0 | 20.87 | 0.0 | 10.41 |
| nikss (49) | 0.0 | 1.01 | 0.0 | 0.0 | 0.0 | 0.0 | 1.09 | 0.03 |

*(b)*

It is interesting to note that the best performances in the case of the ISS task were obtained from an ensemble-based model (i.e., T100). The model actually considers the predictions provided by a first layer of classifiers and makes the prediction of the class on these. Although not yet demonstrated, in the field of information theory, the ability of the model to overcome — or in the worst case equal — the performance of the single classifiers that compose it should be the subject of further investigations. The particular nature of the ISS task (which includes the recognition of irony) could partly motivate these results. It is possible that some models are able to disambiguate better than others in certain cases and, taking into account the predictions provided, the ensemble judges appropriately from time to time.

The results obtained by ELECTRA and GNN in relation to tweet harmful detection are not sufficient to support the approach based on the use of pre-trained embeddings in combination with graph-based networks. Therefore, it would perhaps not be the case to investigate further in this direction.

Finally, with respect to the PCL dataset, it was interesting to note how opportunistically implemented fine-tuning strategies can have such an impact on the final performance of a pre-trained model. Furthermore, the results presented further highlighted the fact that attention-based models generally achieve their best performance by operating on small sample sizes such as in the case of newspaper articles, single tweets or reviews. In contrast to shallow convolutional models trained from scratch, which would appear to fit convolution windows to particular text frames sufficient to perform a correct classification.

# Chapter 7

# Conclusion

## 7.1 Conclusions and future perspectives

### 7.1.1 Conclusions

One of the most relevant challenges in the field of NLP is the TC. The creation and publication of supervised machine learning methods is becoming increasingly important, especially for TC as text and document datasets multiply. Determining these methods is necessary to have a better document categorization system for this information. However, the need to have a better understanding of the complete process involved in TC tasks, models, and algorithms that are already in use could eventually operate more effectively. Currently, a pipeline of this kind can be broadly split in subsequent stages as follows: (I) Present challenges and datasets (II) Applying various strategies and techniques to the raw text during preprocessing, (III) Text representation techniques as Term Frequency-Inverse Document Frequency (TF-IDF), Term Frequency (TF), and Word2Vec, contextualized word representations, Global Vectors for Word Representation (GloVe), and FastText. (IV) Existing classification architectures such as random forest and deep learning models, Transformers, logistic regression, Bayesian classifier, k-nearest neighbor, support vector machine, decision tree classifier, and k-nearest neighbor. (V) Real-world applications and, last but not least, (VI) future perspectives on performance and comprehension of the TC pipeline.

The following are my primary findings and contributions. I listed the prominent dataset used and available in the literature in Chapter 2 along with the current tasks, problems, and applications for TC. Here, I add a method for performing a preliminary analysis on the dataset under consideration. Then, another proposed contribution is about performing data augmentation to enhance or make explicit the latent information available in the text (RQ1). The most popular preprocessing methods for preparing raw text are shown and explored in Chapter 3. In this chapter, I investigated the impact of common preprocessing techniques on a TC model performance. Thanks to a series of studies and experiments, I was able to conclude that selecting a preprocessing method wisely can considerably enhance a model's performance. Furthermore, it is also possible to outperform the performance of large pre-trained model using simpler classifiers adopting the proper preprocessing strategy (RQ2). In Chapter 4, methods for numerically representing text were described, together with a thorough introduction to the attention mechanism. In addition, as a further contribution, I proposed a methodology for a thorough examination of a trained word embedding for a real-case problem and I used the results to improve the model's design (RQ3). Traditional and contemporary classifiers used for TC are covered in Chapter 5. The reference materials for a number of contemporary Transformers are listed. Contributions to this chapter regard several cross-experiments on real world datasets and a methodology for a post-hoc analysis of a CNN layers to investigate further the behavior of a deep learning model and to improve its design (RQ3). I go over all the designs and models created to handle various current international and competitive TC tasks in Chapter 6. In Chapter 7 future perspectives are provided along with the conclusions of this PhD thesis.

I discovered that the traditional approach enhances TC performance primarily by enhancing the classifier design, preprocessing, and text representation scheme. The deep learning model, in contrast, improves performance by enhancing the presentation learning process, the model structure, and the inclusions of new information and data. I can finally say that serious attention to the very initial stages of the categorization pipeline can lead to significant improvements in TC tasks (i.e., data augmentation (RQ1), text preprocessing (RQ2) and representation models (RQ3)). The importance of the ensuing stages varies according to the task being considered as well as the dataset involved.

### 7.1.2 Future perspectives

Two primary paths can be seen on the roadmap for NLP. The first is driven by bigger Transformer Models like GPT-3 and its future relatives. The second important breakthrough will be in dialogue models, where Google, Facebook, and other businesses are investing millions of dollars in R&D. At the moment, in almost every sector, GPT models are sensitively impacting on everyone's life. GPT-3 was created by Open AI, a research company that Elon Musk and other well-known figures like Sam Altman co-founded. A multitasking system called GPT-3 can speak with a human, interpret text, extract text, and, if you're bored, amuse you with its poems. GPT-3 has, nonetheless, developed expertise (and actual utility) in the area of producing computer code. Given the right guidelines, GPT-3 can create full programs in Python, Java, and a number of other languages, opening up interesting new possibilities. Bigger and bigger transformer models, like the GPT-4 or the Chinese variant known as Wu Dao 2.0, are on the horizon.

The second significant development in NLP is the study of dialog models and conversational AI by Google and Facebook. For instance, Google unveiled a demonstration of the LAMDA conversational AI system. Unlike contemporary chatbots, which are programmed for specific conversations, LAMDA has the advantage of being able to communicate with people on a seemingly limitless range of themes. If LAMDA is effective, it will probably disrupt customer service, help desks, and ”whole new types of useful applications,” as one Google blog put it.

In conclusion, the recent strides in Natural Language Processing (NLP) not only render it an appealing investment for professionals and IT enthusiasts but also mark a pivotal moment in its widespread adoption across key sectors such as finance, insurance, and healthcare. The swift expansion of the NLP market as a composite of various technologies underscores the need for practitioners to astutely identify the underlying systems with the utmost commercial potential and strategically time their implementation. Looking forward, the bright future of NLP is unequivocal, characterized by continual enhancements in user experience and the emergence of novel opportunities in unexplored markets. As NLP continues to evolve, its trajectory appears to be one of sustained growth and transformative impact in almost every area of knowledge.

# Bibliography

[1] Lincoln A. Mullen, Kenneth Benoit, Os Keyes, Dmitry Selivanov, and Jeffrey Arnold. Fast, consistent tokenization of natural language text. *Journal of Open Source Software*, 3(23):655, 2018.

[2] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the workshop on language in social media (LSM 2011)*, pages 30–38, 2011.

[3] Alan Akbik, Tanja Bergmann, Duncan Blythe, Kashif Rasul, Stefan Schweter, and Roland Vollgraf. Flair: An easy-to-use framework for state-of-the-art nlp. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

[4] Ahmet Afsin Akın and Mehmet Dündar Akın. Zemberek, an open source nlp framework for turkic languages. *Structure*, 10:1–5, 2007.

[5] Selim Aksoy and Robert M Haralick. Feature normalization and likelihood-based similarity measures for image retrieval. *Pattern recognition letters*, 22(5):563–582, 2001.

[6] Saqib Alam and Nianmin Yao. The impact of preprocessing steps on the accuracy of machine learning algorithms in sentiment analysis. *Computational and Mathematical Organization Theory*, 25(3):319–335, 2019.

[7] Yahya Albalawi, Jim Buckley, and Nikola S Nikolov. Investigating the impact of pre-processing techniques and pre-trained word embeddings in detecting arabic health information on social media. *Journal of big Data*, 8(1):1–29, 2021.

[8] Abdullah Aljebreen, Weiyi Meng, and Eduard Dragut. Segmentation of tweets with urls and its applications to sentiment analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12480–12488, 2021.

[9] Esam Alzahrani and Leon Jololian. How different text-preprocessing techniques using the bert model affect the gender profiling of authors. *arXiv preprint arXiv:2109.13890*, 2021.

[10] Maaz Amjad, Alisa Zhila, Grigori Sidorov, Andrey Labunets, Sabur Butt, Hamza Imam Amjad, Oxana Vitman, and Alexander F. Gelbukh. Overview of abusive and threatening language detection in urdu at FIRE 2021. In Parth Mehta, Thomas Mandl, Prasenjit Majumder, and Mandar Mitra, editors, *Working Notes of FIRE 2021 - Forum for Information Retrieval Evaluation, Gandhinagar, India, December 13-17, 2021*, volume 3159 of *CEUR Workshop Proceedings*, pages 744–762. CEUR-WS.org, 2021.

[11] Murugan Anandarajan, Chelsey Hill, and Thomas Nolan. Text preprocessing. In *Practical Text Analytics*, pages 45–59. Springer, 2019.

[12] Giulio Angiani, Laura Ferrari, Tomaso Fontanini, Paolo Fornacciari, Eleonora Iotti, Federico Magliani, and Stefano Manicardi. A comparison between preprocessing techniques for sentiment analysis in twitter. In *2nd International Workshop on Knowledge Discovery on the WEB (KDWEB)*, volume 1748, 2016.

[13] Talha ANWAR. Identify hate speech spreaders on twitter using transformer embeddings features and automl classifiers. In Faggioli et al. [84], pages 1808–1812.

[14] Egor Araslanov, Evgeniy Komotskiy, and Ebenezer 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)*, pages 1–4. IEEE, 2020.

[15] Murahartawaty Arief and Mustafa Bin Matt Deris. Text preprocessing impact for sentiment classification in product review. In *2021 Sixth International Conference on Informatics and Computing (ICIC)*, pages 1–7. IEEE, 2021.

[16] Ibrahim Arpaci, Mostafa Al-Emran, Mohammed A Al-Sharafi, and Khaled Shaalan. A novel approach for predicting the adoption of smartwatches using machine learning algorithms. In *Recent advances in intelligent systems and smart applications*, pages 185–195. Springer, 2021.

[17] Nastaran Babanejad, Ameeta Agrawal, Aijun An, and Manos 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*, pages 5799–5810, Online, 2020. Association for Computational Linguistics.

[18] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[19] Akshat Bakliwal, Piyush Arora, Senthil Madhappan, Nikhil Kapre, Mukesh Singh, and Vasudeva Varma. Mining sentiments from tweets. In *Proceedings of the 3rd Workshop in Computational Approaches to Subjectivity and Sentiment Analysis*, pages 11–18, 2012.

[20] Alexandra Balahur. Sentiment analysis in social media texts. In *Proceedings of the 4th workshop on computational approaches to subjectivity, sentiment and social media analysis*, pages 120–128, 2013.

[21] Fazlourrahman Balouchzahi, Hosahalli Lakshmaiah Shashirekha, and Grigori Sidorov. Hssd: Hate speech spreader detection using n-grams and voting classifier. In Faggioli et al. [83], pages 1829–1836.

[22] Himani Bansal, Gulshan Shrivastava, Gia Nhu Nguyen, and Loredana-Mihaela Stanciu. *Social network analytics for contemporary business organizations.* IGI Global, 2018.

[23] Yanwei Bao, Changqin Quan, Lijuan Wang, and Fuji Ren. The role of pre-processing in twitter sentiment analysis. In *International conference on intelligent computing*, pages 615–624. Springer, 2014.

[24] Luciano Barbosa and Junlan Feng. Robust sentiment detection on twitter from biased and noisy data. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, page 36–44, USA, 2010. Association for Computational Linguistics.

[25] Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Manuel Rangel Pardo, Paolo Rosso, and Manuela Sanguinetti. Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In *Proceedings of the 13th international workshop on semantic evaluation*, pages 54–63, 2019.

[26] Djamila Romaissa Beddiar, Md Saroar Jahan, and Mourad Oussalah. Data expansion using back translation and paraphrasing for hate speech detection. *Online Social Networks and Media*, 24:100153, 2021.

[27] Dorothée Behr. Assessing the use of back translation: The shortcomings of back translation as a quality testing method. *International Journal of Social Research Methodology*, 20(6):573–584, 2017.

[28] Emily M. Bender and Alexander Koller. Climbing towards NLU: On meaning, form, and understanding in the age of data. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5185–5198, Online, 2020. Association for Computational Linguistics.

[29] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[30] Slim Benzarti and Rim 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*, pages 227–238. Springer, 2015.

[31] Janek Bevendorff, Berta Chulvi, Elisabetta Fersini, Annina Heini, Mike Kestemont, Krzysztof Kredens, Maximilian Mayerl, Reyner Ortega-Bueno, Piotr Pezik, Martin Potthast, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, Benno Stein, Matti Wiegmann, Magdalena Wolska, and Eva Zangerle. Overview of PAN 2022: Authorship Verification, Profiling Irony and Stereotype Spreaders, and Style Change Detection. In Alberto Barron-Cedeno, Giovanni Da San Martino, Mirko Degli Esposti, Fabrizio Sebastiani, Craig Macdonald, Gabriella Pasi, Allan Hanbury, Martin Potthast, Guglielmo Faggioli, and Nicola Ferro, editors, *Experimental IR Meets Multilinguality, Multimodality, and Interaction. Proceedings of the Thirteenth International Conference of the CLEF Association (CLEF 2022)*, volume 13390 of *Lecture Notes in Computer Science*. Springer, 2022.

[32] Janek Bevendorff, Berta Chulvi, Elisabetta Fersini, Annina Heini, Mike Kestemont, Krzysztof Kredens, Maximilian Mayerl, Reynier Ortega-Bueno, Piotr Pkezik, Martin Potthast, et al. Overview of pan 2022: Authorship verification, profiling irony and stereotype spreaders, and style change detection. In *International Conference of the Cross-Language Evaluation Forum for European Languages*, pages 382–394. Springer, 2022.

[33] Janek Bevendorff, BERTa Chulvi, Gretel Liz De La Peña Sarracén, Mike Kestemont, Enrique Manjavacas, Ilia Markov, Maximilian Mayerl, Martin Potthast, Francisco Rangel, Paolo Rosso, Efstathios Stamatatos, Benno Stein, Matti Wiegmann, Magdalena Wolska, , and Eva Zangerle. Overview of PAN 2021: Authorship Verification,Profiling Hate Speech Spreaders on Twitter,and Style Change Detection. In *12th International Conference of the CLEF Association (CLEF 2021)*, page 1. Springer, 2021.

[34] Steven Bird. Nltk: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pages 69–72, 2006.

[35] Thomas Body, Xiaohui Tao, Yuefeng Li, Lin Li, and Ning Zhong. Using back-and-forth translation to create artificial augmented textual data for sentiment analysis models. *Expert Systems with Applications*, 178:115033, 2021.

[36] Erik Boiy, Pieter Hens, Koen Deschacht, and Marie-Francine Moens. Automatic sentiment analysis in on-line 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*, pages 349–360, 2007.

[37] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[38] Erik Borra and Bernhard Rieder. Programmed method: Developing a toolset for capturing and analyzing tweets. *Aslib journal of information management*, 66(3):262–278, 2014.

[39] Cristina Bosco, Dell'Orletta Felice, Fabio Poletto, Manuela Sanguinetti, and Tesconi Maurizio. Overview of the evalita 2018 hate speech detection task. In *Proceedings of the Sixth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2018) co-located with the Fifth Italian Conference on Computational Linguistics (CLiC-it 2018), Turin, Italy, December 12-13, 2018*, volume 2263 of *CEUR Workshop Proceedings*, pages 1–9. CEUR, 2018.

[40] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[41] Richard W Brislin and Carolina Freimanis. *Back-translation: A tool for cross-cultural research*, volume 1. The Chinese University Press Hong Kong, CN, 1995.

[42] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.

[43] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[44] Jakab Buda and Flora Bolonyai. An ensemble model using n-grams and statistical features to identify fake news spreaders on twitter. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[45] Reynier Ortega Bueno, Berta Chulvi, Francisco Rangel, Paolo Rosso, and Elisabetta Fersini. Profiling irony and stereotype spreaders on twitter (irostereo). overview for pan at clef 2022. In Faggioli et al. [83], pages 2314–2343.

[46] Reynier Ortega Bueno, Berta Chulvi, Francisco Rangel, Paolo Rosso, and Elisabetta Fersini. Profiling irony and stereotype spreaders on twitter (irostereo). overview for pan at clef 2022. In Faggioli et al. [83], pages 2314–2343.

[47] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on scientific computing*, 16(5):1190–1208, 1995.

[48] Jose Camacho-Collados and Mohammad Taher 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*, pages 40–46, 2018.

[49] Erik Cambria, Björn Schuller, Yunqing Xia, and Catherine Havasi. New avenues in opinion mining and sentiment analysis. *IEEE Intelligent systems*, 28(2):15–21, 2013.

[50] Fazli Can, Seyit Kocberber, Erman Balcik, Cihan Kaynak, H Cagdas Ocalan, and Onur M Vursavas. Information retrieval on turkish texts. *Journal of the American Society for Information Science and Technology*, 59(3):407–421, 2008.

[51] Riccardo Cervero. Use of lexical and psycho-emotional information to detect hate speech spreaders on twitter - notebook for pan at clef 2021. In Faggioli et al. [84], pages 1883–1891.

[52] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):1–27, 2011.

[53] Guanhua Chen, Shuming Ma, Yun Chen, Li Dong, Dongdong Zhang, Jia Pan, Wenping Wang, and Furu 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*, pages 15–26, 2021.

[54] Guibin Chen, Deheng Ye, Zhenchang Xing, Jieshan Chen, and Erik Cambria. Ensemble application of convolutional and recurrent neural networks for multi-label text categorization. In *2017 International joint conference on neural networks (IJCNN)*, pages 2377–2383. IEEE, 2017.

[55] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. In Jian Su, Xavier Carreras, and Kevin Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 551–561. The Association for Computational Linguistics, 2016.

[56] Davide Chicco. Siamese neural networks: An overview. In Hugh M. Cartwright, editor, *Artificial Neural Networks - Third Edition*, volume 2190 of *Methods in Molecular Biology*, pages 73–94. Springer, 2021.

[57] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020.* OpenReview.net, 2020.

[58] Fabrice Colas and Pavel Brazdil. Comparison of svm and some older classification algorithms in text classification tasks. In *IFIP International Conference on Artificial Intelligence in Theory and Practice*, pages 169–178. Springer, 2006.

[59] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[60] Cherie Courseault Trumbach and Dinah Payne. Identifying synonymous concepts in preparation for technology mining. *Journal of Information Science*, 33(6):660–677, 2007.

[61] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[62] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley, 2001.

[63] Daniele Croce, Domenico Garlisi, and Marco Siino. An svm ensembler approach to detect irony and stereotype spreaders on twitter. In Faggioli et al. [83], pages 2426–2432.

[64] Washington Cunha, Sérgio Canuto, Felipe Viegas, Thiago Salles, Christian Gomes, Vitor Mangaravite, Elaine Resende, Thierson Rosa, Marcos André Gonçalves, and Leonardo Rocha. Extended pre-processing pipeline for text classification: On the role of meta-feature representations, sparsification and selective sampling. *Information Processing & Management*, 57(4):102263, 2020.

[65] Washington Cunha, Vítor Mangaravite, Christian Gomes, Sérgio Canuto, Elaine Resende, Cecilia Nascimento, Felipe Viegas, Celso França, Wellington Santos Martins, Jussara M. Almeida, Thierson Rosa, Leonardo Rocha, and Marcos André Gonçalves. On the cost-effectiveness of neural and non-neural approaches and representations for text classification: A comprehensive comparative study. *Information Processing & Management*, 58(3):102481, 2021.

[66] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G Carbonell, Quoc Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988, 2019.

[67] Robert Dale. Gpt-3: What's it good for? *Natural Language Engineering*, 27(1):113–118, 2021.

[68] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280, 2007.

[69] Angel Felipe Magnossão de Paula, Paolo Rosso, and Damiano Spina. Mitigating negative transfer with task awareness for sexism, hate speech, and toxic language detection. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2023.

[70] Jane Elizabeth Demmen and Jonathan Vaughan Culpeper. Keywords. In Douglas Biber and Randi Reppen, editors, *The Cambridge handbook of English corpus linguistics*, pages 90–105. Cambridge University Press, 2015.

[71] Lingjia Deng and Janyce Wiebe. Mpqa 3.0: An entity/event-level sentiment corpus. In *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1323–1328, 2015.

[72] Matthew J Denny and Arthur Spirling. Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. *Political Analysis*, 26(2):168–189, 2018.

[73] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[74] Adji B. Dieng, Chong Wang, Jianfeng Gao, and John W. Paisley. Topicrnn: A recurrent neural network with long-range semantic dependency. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[75] Xiao Ding, Kuo Liao, Ting Liu, Zhongyang Li, and Junwen Duan. Event representation learning enhanced with external commonsense knowledge. *arXiv preprint arXiv:1909.05190*, 2019.

[76] Nemanja Djuric, Jing Zhou, Robin Morris, Mihajlo Grbovic, Vladan Radosavljevic, and Narayan Bhamidipati. Hate speech detection with comment embeddings. In *Proceedings of the 24th international conference on world wide web*, pages 29–30, 2015.

[77] Ljiljana Dolamic and Jacques Savoy. When stopword lists make the difference. *J. Assoc. Inf. Sci. Technol.*, 61(1):200–203, 2010.

[78] Huu-Thanh Duong and Tram-Anh Nguyen-Thi. A review: preprocessing techniques and data augmentation for sentiment analysis. *Computational Social Networks*, 8(1):1–16, 2021.

[79] José Antonio García Díaz, Miguel Ángel Rodríguez-García, Francisco García-Sánchez, and Rafael Valencia-Garcia. Umuteam at irostereo: Profiling irony and stereotype spreaders on twitter combining linguistic features with transformers. In Faggioli et al. [83], pages 2445–2454.

[80] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, 2018.

[81] Daniel Embarcadero-Ruiz, Helena Gómez-Adorno, Alberto Embarcadero-Ruiz, and Gerardo Sierra. Graph-based siamese network for authorship verification. *Mathematics*, 10(2):277, 2022.

[82] Flora Bolonyai Eszter Katona, Jakab Buda. Using n-grams and statistical features to identify hate speech spreaders on twitter. In Faggioli et al. [84], pages 2025–2034.

[83] Guglielmo Faggioli, Nicola Ferro, Allan Hanbury, and Martin Potthast, editors. *Proceedings of the Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum (CLEF)*, number 3180 in CEUR Workshop Proceedings, Aachen, 2022.

[84] Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi, editors. *Proceedings of the Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum (CLEF 2021)*, number 2936 in CEUR Workshop Proceedings, Aachen, 2021.

[85] Elisabetta Fersini, Paolo Rosso, and Maria Anzovino. Overview of the task on automatic misogyny identification at ibereval 2018. In Paolo Rosso, Julio Gonzalo, Raquel Martínez, Soto Montalvo, and Jorge Carrillo de Albornoz, editors, *Proceedings of the Third Workshop on Evaluation of Human Language Technologies for Iberian Languages (IberEval 2018) co-located with 34th Conference of the Spanish Society for Natural Language Processing (SEPLN 2018), Sevilla, Spain, September 18th, 2018*, volume 2150 of *CEUR Workshop Proceedings*, pages 214–228. CEUR-WS.org, 2018.

[86] Evgeny Finogeev, Mariam Kaprielova, Artem Chashchin, Kirill Grashchenkov, George Gorbachev, and Oleg Bakhteev. Hate speech spreader detection using contextualized word embeddings. In Faggioli et al. [84], pages 1937–1944.

[87] Barbara J. Flood. Historical note: The start of a stop list at biological abstracts. *J. Am. Soc. Inf. Sci.*, 50(12):1066, 1999.

[88] Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969.

[89] Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[90] Jianfeng Gao, Baolin Peng, Chunyuan Li, Jinchao Li, Shahin Shayandeh, Lars Liden, and Heung-Yeung Shum. Robust conversational ai with grounded text generation. *arXiv e-prints*, pages arXiv–2009, 2020.

[91] Neha Garg and Kamlesh Sharma. Text pre-processing of multilingual for sentiment analysis based on social network data. *International Journal of Electrical & Computer Engineering (2088-8708)*, 12(1), 2022.

[92] Fahriye Gemci and Kadir A Peker. Extracting turkish tweet topics using lda. In *2013 8th International Conference on Electrical and Electronics Engineering (ELECO)*, pages 531–534. IEEE, 2013.

[93] Alexander Genkin, David D. Lewis, and David Madigan. Large-scale bayesian logistic regression for text categorization. *Technometrics*, 49(3):291–304, 2007.

[94] Martin Gerlach, Hanyu Shi, and Luís A Nunes Amaral. A universal information theoretic approach to the identification of stopwords. *Nature Machine Intelligence*, 1(12):606–612, 2019.

[95] Mykhailo Granik and Volodymyr Mesyura. Fake news detection using naive bayes classifier. In *2017 IEEE first Ukraine conference on electrical and computer engineering (UKRCON)*, pages 900–903. IEEE, 2017.

[96] Vishal Gupta and Gurpreet Singh Lehal. Punjabi language stemmer for nouns and proper names. In *Proceedings of the 2nd Workshop on South Southeast Asian Natural Language Processing (WSSANLP)*, pages 35–39, 2011.

[97] Emitza Guzman and Walid Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd International Requirements Engineering Conference (RE)*, pages 153–162, 2014.

[98] Claudia Gómez and Daniel Parres. Bert sentence embeddings in different machine learning and deep learning models for author profiling applied to irony and stereotype spreaders on twitter. In Faggioli et al. [83], pages 2467–2474.

[99] Yaakov HaCohen-Kerner, Daniel Miller, and Yair Yigal. The influence of preprocessing on text classification using a bag-of-words representation. *PloS one*, 15(5):e0232525, 2020.

[100] Emma Haddi, Xiaohui Liu, and Yong Shi. The role of text pre-processing in sentiment analysis. In Yong Shi, Youmin Xi, Peter Wolcott, Yingjie Tian, Jianping Li, Daniel Berg, Zhengxin Chen, Enrique Herrera-Viedma, Gang Kou, Heeseok Lee, Yi Peng, and Lean Yu, editors, *Proceedings of the First International Conference on Information Technology and Quantitative Management, ITQM 2013, Dushu Lake Hotel, Sushou, China, 16-18 May, 2013*, volume 17 of *Procedia Computer Science*, pages 26–32. Elsevier, 2013.

[101] Ummu Hani' Hair Zaki, Roliana Ibrahim, Shahliza Abd Halim, and Izyan Izzati 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*, pages 50–61. Springer, 2022.

[102] Marcus Hassler and Günther Fliedl. Text preparation through extended tokenization. *WIT Transactions on Information and Communication Technologies*, 37, 2006.

[103] Tomoki Hayashi, Shinji Watanabe, Yu Zhang, Tomoki Toda, Takaaki Hori, Ramon Astudillo, and Kazuya Takeda. Back-translation-style data augmentation for end-to-end asr. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 426–433. IEEE, 2018.

[104] Delia Irazú Hernández Farías, Rosa María Ortega-Mendoza, and Manuel Montes-y Gómez. Exploring the use of psycholinguistic information in author profiling. In *Mexican Conference on Pattern Recognition*, pages 411–421. Springer, 2019.

[105] Louis Hickman, Stuti Thapa, Louis Tay, Mengyang Cao, and Padmini Srinivasan. Text preprocessing for text mining in organizational research: Review and recommendations. *Organizational Research Methods*, 25(1):114–146, 2022.

[106] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[107] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *International conference on learning representations*, 2018.

[108] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[109] Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[110] Alexander Hogenboom, Daniella Bal, Flavius Frasincar, Malissa Bal, Franciska De Jong, and Uzay Kaymak. Exploiting emoticons in sentiment analysis. In *Proceedings of the 28th annual ACM symposium on applied computing*, pages 703–710, 2013.

[111] Ali Hosseinalipour and Reza Ghanbarzadeh. A novel approach for spam detection using horse herd optimization algorithm. *Neural Computing and Applications*, pages 1–15, 2022.

[112] Xinting Huang. Profiling irony and stereotype spreaders with language models and bayes' theorem. In Faggioli et al. [83], pages 2496–2505.

[113] Julian Höllig, Yeong Su Lee, Nina Seemann, and Michaela Geierhos. Effective detection of hate speech spreaders on twitter. In Faggioli et al. [84], pages 1976–1986.

[114] Catherine Ikae. Unine at pan-clef 2022: Profiling irony and stereotype spreaders on twitter. In Faggioli et al. [83], pages 2506–2514.

[115] ST Indra, Liza Wikarsa, and Rinaldo Turang. Using logistic regression method to classify tweets into the selected topics. In *2016 international conference on advanced computer science and information systems (icacsis)*, pages 385–390. IEEE, 2016.

[116] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*, pages 1681–1691, 2015.

[117] Alon Jacovi, Oren Sar Shalom, and Yoav Goldberg. Understanding convolutional neural networks for text classification. In Tal Linzen, Grzegorz Chrupala, and Afra Alishahi, editors, *Proceedings of the Workshop: Analyzing and Interpreting Neural Networks for NLP, BlackboxNLP@EMNLP 2018, Brussels, Belgium, November 1, 2018*, pages 56–65. Association for Computational Linguistics, 2018.

[118] Shengyi Jiang, Guansong Pang, Meiling Wu, and Limin Kuang. An improved k-nearest-neighbor algorithm for text categorization. *Expert Systems with Applications*, 39(1):1503–1509, 2012.

[119] Zhao Jianqiang and Gui Xiaolin. Comparison research on text pre-processing methods on twitter sentiment analysis. *IEEE Access*, 5:2870–2879, 2017.

[120] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8018–8025. AAAI Press, 2020.

[121] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer, 1998.

[122] Thorsten Joachims. Transductive inference for text classification using support vector machines. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 200–209. Morgan Kaufmann, 1999.

[123] Thorsten Joachims. A statistical learning model of text classification for svms. In *Learning to Classify Text Using Support Vector Machines*, pages 45–74. Springer, 2002.

[124] David E. Johnson, Frank J. Oles, Tong Zhang, and Thilo Goetz. A decision-tree-based symbolic rule induction system for text categorization. *IBM Systems Journal*, 41(3):428–437, 2002.

[125] Rie Johnson and Tong Zhang. Supervised and semi-supervised text categorization using lstm for region embeddings. In *International Conference on Machine Learning*, pages 526–534. PMLR, 2016.

[126] Tang-Mui Joo and Chan-Eang Teng. Impacts of social media (facebook) on human communication and relationships: A view on behavioral change and social unity. *International Journal of Knowledge Content Development & Technology*, 7(4):27–50, 2017.

[127] José Andrés Jorge Alcañiz. Profiling hate spreaders using word n-grams. In Faggioli et al. [84], pages 1790–1795.

[128] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[129] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.

[130] Dan Jurafsky and James H. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition, 2nd Edition.* Prentice Hall series in artificial intelligence. Prentice Hall, Pearson Education International, 2009.

[131] Ammar Ismael Kadhim. An evaluation of preprocessing techniques for text classification. *International Journal of Computer Science and Information Security (IJCSIS)*, 16(6), 2018.

[132] Abhinav Kathuria, Anu Gupta, and RK Singla. A review of tools and techniques for preprocessing of textual data. *Computational Methods and Data Engineering*, pages 407–422, 2021.

[133] Eoin M Kenny, Courtney Ford, Molly Quinn, and Mark T Keane. Explaining black-box classifiers using post-hoc explanations-by-example: The effect of explanations and error-rates in XAI user studies. *Artificial Intelligence*, 294:103459, 2021.

[134] Keras. Layer weight initializers. `https://keras.io/api/layers/initializers/`, 2021.

[135] Lida Ketsbaia, Biju Issac, and Xiaomin 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)*, pages 751–758. IEEE, 2020.

[136] Adam Kilgarriff. Comparing corpora. *International journal of corpus linguistics*, 6(1):97–133, 2001.

[137] Adam Kilgarriff. Getting to know your corpus. In *International conference on text, speech and dialogue*, pages 3–15. Springer, 2012.

[138] Adam Kilgarriff, Vít Baisa, Jan Bušta, Miloš Jakubíček, Vojtěch Kovář, Jan Michelfeit, Pavel Rychlý, and Vít Suchomel. The Sketch Engine: ten years on. *Lexicography*, pages 7–36, 2014.

[139] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics.

[140] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

[141] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[142] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[143] Donald Knuth. *The Art Of Computer Programming, vol. 3: Sorting And Searching*. Addison-Wesley, 1973.

[144] Boshko Koloski, Senja Pollak, and Blaz Skrlj. Multilingual detection of fake news spreaders via sparse matrix factorization. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[145] Cynthia Koopman and Adalbert Wilhelm. The effect of preprocessing on short document clustering. *Archives of Data Science, Series A*, 6(1):01, 2020.

[146] Efthymios Kouloumpis, Theresa Wilson, and Johanna Moore. Twitter sentiment analysis: The good the bad and the omg! In *Proceedings of the international AAAI conference on web and social media*, volume 5, pages 538–541, 2011.

[147] Kamran Kowsari, Donald E Brown, Mojtaba Heidarysafa, Kiana Jafari Meimandi, Matthew S Gerber, and Laura E Barnes. Hdltex: Hierarchical deep learning for text classification. In *2017 16th IEEE international conference on machine learning and applications (ICMLA)*, pages 364–371. IEEE, 2017.

[148] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[149] Taku 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)*, pages 66–75, 2018.

[150] Pavan Kumar and LD Dhinesh Babu. Novel text preprocessing framework for sentiment analysis. In *Smart intelligent computing and applications*, pages 309–317. Springer, 2019.

[151] Ritesh Kumar, Guggilla Bhanodai, Rajendra Pamula, and Maheshwar Reddy Chennuru. Trac-1 shared task on aggression identification: Iit (ism)@ coling'18. In *Proceedings of the first workshop on trolling, aggression and cyberbullying (TRAC-2018)*, pages 58–65, 2018.

[152] Maria Kunilovskaya and Alistair Plum. Text preprocessing and its implications in a digital humanities project. In *Proceedings of the Student Research Workshop Associated with RANLP 2021*, pages 85–93, 2021.

[153] Aliyah Kurniasih and Lindung Parningotan Manik. On the role of text preprocessing in bert embedding-based dnns for classifying informal texts. *International Journal of Advanced Computer Science and Applications*, 13(6):927 – 934, 2022.

[154] Ilia Kuznetsov and Iryna Gurevych. From text to lexicon: Bridging the gap between word embeddings and lexical resources. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 233–244, 2018.

[155] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[156] Thomas K Landauer and Susan T Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological review*, 104(2):211, 1997.

[157] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196. PMLR, 2014.

[158] Junghoon Lee, Jounghee Kim, and Pilsung Kang. Back-translated task adaptive pretraining: Improving accuracy and robustness on text classification. *arXiv preprint arXiv:2107.10474*, 2021.

[159] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic web*, 6(2):167–195, 2015.

[160] Edda Leopold and Jörg Kindermann. Text categorization with support vector machines. how to represent texts in input space? *Machine Learning*, 46(1):423–444, 2002.

[161] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for svm protein classification. In *Biocomputing 2002*, pages 564–575. World Scientific, 2001.

[162] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. A survey on text classification: From shallow to deep learning. *arXiv preprint arXiv:2008.00364*, 2020.

[163] Qian Li, Hao Peng, Jianxin Li, Congying Xia, Renyu Yang, Lichao Sun, Philip S Yu, and Lifang He. A survey on text classification: From traditional to deep learning. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(2):1–41, 2022.

[164] Xin Li and Yuhong Guo. Active learning with multi-label SVM classification. In Francesca Rossi, editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1479–1485. IJCAI/AAAI, 2013.

[165] Xin Li and Wai Lam. Deep multi-task learning for aspect term extraction with memory interaction. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 2886–2892, 2017.

[166] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[167] Chenghua Lin and Yulan He. Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 375–384, 2009.

[168] Pierre Lison and Andrey Kutuzov. Redefining context windows for word embedding models: An experimental study. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 284–288, 2017.

[169] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*, pages 115–124, 2017.

[170] Pengfei Liu, Xipeng Qiu, Xinchi Chen, Shiyu Wu, and Xuan-Jing Huang. Multi-timescale long short-term memory neural network for modelling sentences and documents. In *Proceedings of*

*the 2015 conference on empirical methods in natural language processing*, pages 2326–2335, 2015.

[171] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. In Subbarao Kambhampati, editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2873–2879. IJCAI/AAAI Press, 2016.

[172] Xiaodong Liu, Yelong Shen, Kevin Duh, and Jianfeng Gao. Stochastic answer networks for machine reading comprehension. In Iryna Gurevych and Yusuke Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 1694–1704. Association for Computational Linguistics, 2018.

[173] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[174] Zhijie Liu, Xueqiang Lv, Kun Liu, and Shuicai Shi. Study on svm compared with the other text classification methods. In *2010 Second international workshop on education technology and computer science*, volume 1, pages 219–222. IEEE, 2010.

[175] Rachel Tsz-Wai Lo, Ben He, and Iadh 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)*, volume 5, pages 17–24, 2005.

[176] Francesco Lomonaco, Gregor Donabauer, and Marco Siino. Courage at checkthat! 2022: Harmful tweet detection using graph neural networks and electra. In Faggioli et al. [83], pages 573–583.

[177] Francesco Lomonaco, Marco Siino, and Maurizio Tesconi. Text enrichment with japanese language to profile cryptocurrency influencers. In Mohammad Aliannejadi, Guglielmo Faggioli, Nicola Ferro, and Michalis Vlachos, editors, *Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2023), Thessaloniki, Greece, September 18th to 21st, 2023*, volume 3497 of *CEUR Workshop Proceedings*, pages 2708–2716. CEUR-WS.org, 2023.

[178] Julie Beth Lovins. Development of a stemming algorithm. *Mechanical Translation and Computational Linguistics*, 11(1-2):22–31, 1968.

[179] Eneldo Loza Mencía and Johannes Fürnkranz. Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer, 2008.

[180] Hans Peter Luhn. Key word-in-context index for technical literature (kwic index). *American documentation*, 11(4):288–295, 1960.

[181] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[182] David M Magerman. Statistical decision-tree models for parsing. In *33rd Annual Meeting of the Association for Computational Linguistics*, pages 276–283, 1995.

[183] Masoud Makrehchi and Mohamed 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*, pages 222–233. Springer, 2008.

[184] Stefano Mangione, Marco Siino, and Giovanni Garbo. Improving irony and stereotype spreaders detection using data augmentation and convolutional neural network. In Faggioli et al. [83], pages 2585–2593.

[185] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval.* Cambridge University Press, 2008.

[186] Christopher D. Manning, Hinrich Schütze, and Gerhard Weikurn. Foundations of statistical natural language processing. *SIGMOD Record*, 31(3):37 – 38, 2002.

[187] Gary Marcus and Ernest Davis. *Rebooting AI: Building artificial intelligence we can trust.* Vintage, 2019.

[188] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[189] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[190] Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6294–6305, 2017.

[191] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[192] Paul McNamee and James Mayfield. Character n-gram tokenization for european language text retrieval. *Information retrieval*, 7(1):73–97, 2004.

[193] Oren Melamud, Jacob Goldberger, and Ido Dagan. context2vec: Learning generic context embedding with bidirectional LSTM. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 51–61, Berlin, Germany, August 2016. Association for Computational Linguistics.

[194] Oren Melamud, David McClosky, Siddharth Patwardhan, and Mohit Bansal. The role of context types and dimensionality in learning word embeddings. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1030–1040, 2016.

[195] Lesly Miculicich, Dhananjay Ram, Nikolaos Pappas, and James Henderson. Document-level neural machine translation with hierarchical attention networks. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 2947–2954. Association for Computational Linguistics, 2018.

[196] Rada Mihalcea and Paul Tarau. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411, 2004.

[197] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio and Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013.

[198] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

[199] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[200] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning–based text classification: a comprehensive review. *ACM Computing Surveys (CSUR)*, 54(3):1–40, 2021.

[201] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):1979–1993, 2018.

[202] Fahim Mohammad. Is preprocessing of text really worth your time for online comment classification? *arXiv preprint arXiv:1806.02908*, 2018.

[203] Cristian Moral, Angélica de Antonio, Ricardo Imbert, and Jaime Ramírez. A survey of stemming algorithms in information retrieval. *Information Research: An International Electronic Journal*, 19(1), 2014.

[204] Mohamad Syahrul Mubarok, Adiwijaya, and Muhammad Dwi Aldhi. Aspect-based sentiment analysis to review products using naïve bayes. In *AIP Conference Proceedings*, volume 1867, page 020060. AIP Publishing LLC, 2017.

[205] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[206] Tony Mullen and Robert Malouf. A preliminary investigation into sentiment analysis of informal political discourse. In *AAAI spring symposium: computational approaches to analyzing weblogs*, pages 159–162, 2006.

[207] Preslav Nakov, Alberto Barrón-Cedeño, Giovanni Da San Martino, Firoj Alam, Rubén Míguez, Tommaso Caselli, Mucahid Kutlu, Wajdi Zaghouani, Chengkai Li, Shaden Shaar, Hamdy Mubarak, Alex Nikolov, Yavuz Selim Kartal, and Javier Beltrán. Overview of the CLEF-2022 CheckThat! lab task 1 on identifying relevant claims in tweets. In *Working Notes of CLEF 2022—Conference and Labs of the Evaluation Forum*, CLEF '2022, Bologna, Italy, 2022.

[208] Preslav Nakov, Alberto Barrón-Cedeño, Giovanni Da San Martino, Firoj Alam, Julia Maria Struß, Thomas Mandl, Rubén Míguez, Tommaso Caselli, Mucahid Kutlu, Wajdi Zaghouani, Chengkai Li, Shaden Shaar, Gautam Kishore Shahi, Hamdy Mubarak, Alex Nikolov, Nikolay Babulkov, Yavuz Selim Kartal, and Javier Beltrán. The clef-2022 checkthat! lab on fighting the covid-19 infodemic and fake news detection. In Matthias Hagen, Suzan Verberne, Craig Macdonald, Christin Seifert, Krisztian Balog, Kjetil Nørvåg, and Vinay Setty, editors, *Advances in Information Retrieval*, pages 416–428, Cham, 2022. Springer International Publishing.

[209] Preslav Nakov, Alberto Barrón-Cedeño, Giovanni Da San Martino, Firoj Alam, Ruben Miguez, Tommaso Caselli, Mucahid Kutlu, Wajdi Zaghouani, Chengkai Li, Shaden Shaar, Hamdy Mubarak, Alex Nikolov, and Yavuz Selim Kartal. Overview of the clef-2022 checkthat! lab task 1 on identifying relevant claims in tweets. In Faggioli et al. [83], pages 368–392.

[210] Usman Naseem, Imran Razzak, and Peter W Eklund. A survey of pre-processing techniques to improve short-text quality: a case study on hate speech detection on twitter. *Multimedia Tools and Applications*, 80(28):35239–35266, 2021.

[211] Jakub Nowak, Ahmet Taspinar, and Rafał Scherer. Lstm recurrent neural networks for short text and sentiment classification. In *International Conference on Artificial Intelligence and Soft Computing*, pages 553–562. Springer, 2017.

[212] Uldis Ozolins, Sandra Hale, Xiang Cheng, Amelia Hyatt, and Penelope Schofield. Translation and back-translation methodology in health research–a critique. *Expert review of pharmacoeconomics & outcomes research*, 20(1):69–77, 2020.

[213] Chris D. Paice. Another stemmer. *SIGIR Forum*, 24(3):56–61, nov 1990.

[214] David 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*, pages 321–328, 1997.

[215] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up? sentiment classification using machine learning techniques. In *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*, pages 79–86, 2002.

[216] Francisco M. Rangel Pardo, Anastasia Giachanou, Bilal Ghanem, and Paolo Rosso. Overview of the 8th author profiling task at PAN 2020: Profiling fake news spreaders on twitter. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[217] S Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[218] Samuel Pecar, Marian Simko, and Maria Bielikova. Sentiment analysis of customer reviews: Impact of text pre-processing. In *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*, pages 251–256, 2018.

[219] Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 world wide web conference*, pages 1063–1072, 2018.

[220] Hao Peng, Jianxin Li, Senzhang Wang, Lihong Wang, Qiran Gong, Renyu Yang, Bo Li, S Yu Philip, and Lifang He. Hierarchical taxonomy-aware and attentional graph capsule rcnns for large-scale multi-label text classification. *IEEE Transactions on Knowledge and Data Engineering*, 33(6):2505–2519, 2019.

[221] Tao Peng, Wanli Zuo, and Fengling He. Svm based adaptive learning method for text classification from positive and unlabeled documents. *Knowledge and Information Systems*, 16(3):281–301, 2008.

[222] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[223] Jose Manuel Pereira, Mario Basto, and Amelia Ferreira da Silva. The logistic lasso and ridge regression in predicting corporate failure. *Procedia Economics and Finance*, 39:634–641, 2016.

[224] Carla Pérez-Almendros, Luis Espinosa Anke, and Steven 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*, pages 5891–5902, 2020.

[225] Carla Pérez-Almendros, Luis Espinosa Anke, and Steven Schockaert. Semeval-2022 task 4: Patronizing and condescending language detection. In *Proceedings of the 16th International Workshop on Semantic Evaluation (SemEval-2022)*, pages 298–307. Association for Computational Linguistics, 2022.

[226] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In Marilyn A. Walker, Heng Ji, and Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018.

[227] ME Peters, M Neumann, M Iyyer, M Gardner, C Clark, K Lee, and L Zettlemoyer. Deep contextualized word representations. arxiv 2018. *arXiv preprint arXiv:1802.05365*, 12, 1802.

[228] Dje Petrović and Milena Stanković. The influence of text preprocessing methods and tools on calculating text similarity. *Facta Universitatis, Series: Mathematics and Informatics*, 34:973–994, 2019.

[229] Juan Pizarro. Using n-grams to detect fake news spreaders on twitter. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[230] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[231] Martin Potthast, Tim Gollub, Matti Wiegmann, and Benno Stein. TIRA Integrated Research Architecture. In Nicola Ferro and Carol Peters, editors, *Information Retrieval Evaluation in a Changing World*, The Information Retrieval Series, page 1. Springer, Berlin Heidelberg New York, September 2019.

[232] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and Sundaraja S Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):1–36, 2018.

[233] Saurav Pradha, Malka N Halgamuge, and Nguyen Tran Quoc Vinh. Effective text data pre-processing technique for sentiment analysis in social media data. In *2019 11th international conference on knowledge and systems engineering (KSE)*, pages 1–8. IEEE, 2019.

[234] P Pradhyumna, G P Shreya, and 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)*, pages 1183–1189. IEEE, 2021.

[235] Zhaowei Qu, Xiaomin Song, Shuqiang Zheng, Xiaoru Wang, Xiaohui Song, and Zuquan Li. Improved bayes method based on tf-idf feature and grade factor feature for chinese information classification. In *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 677–680. IEEE, 2018.

[236] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[237] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[238] Francisco Rangel, Marìa Alberta Chulvi, Gretel Liz De La Pena, Elisabetta Fersini, and Paolo Rosso. Profiling Hate Speech Spreaders on Twitter [Data set]. `https://zenodo.org/record/4603578`, 2021.

[239] Francisco Rangel, Gretel Liz De la Peña Sarracén, Berta Chulvi, Elisabetta Fersini, and Paolo Rosso. Profiling hate speech spreaders on twitter task at PAN 2021. In Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi, editors, *Proceedings of the Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21st - to - 24th, 2021*, volume 2936 of *CEUR Workshop Proceedings*, pages 1772–1789. CEUR-WS.org, 2021.

[240] Sebastian Raschka. Naive bayes and text classification i-introduction and theory. *arXiv preprint arXiv:1410.5329*, 2014.

[241] Rajeev Rastogi and Kyuseok Shim. Public: A decision tree classifier that integrates building and pruning. *Data Mining and Knowledge Discovery*, 4(4):315–344, 2000.

[242] Fero Resyanto, Yuliant Sibaroni, and Ade Romadhony. Choosing the most optimum text preprocessing method for sentiment analysis: Case: iphone tweets. In *2019 Fourth International Conference on Informatics and Computing (ICIC)*, pages 1–5. IEEE, 2019.

[243] Tiago Filipe Nunes Ribeiro, Yana Nikolaeva Nikolova, and Kaja Seraphina Elisa Hano. Irony stereotype spreader detection using random forests. In Faggioli et al. [83], pages 2623–2641.

[244] Julian Risch, Anke Stoll, Lena Wilms, and Michael Wiegand. Overview of the germeval 2021 shared task on the identification of toxic, engaging, and fact-claiming comments. In *Proceedings of the GermEval 2021 Shared Task on the Identification of Toxic, Engaging, and Fact-Claiming Comments*, pages 1–12, 2021.

[245] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[246] Mochamad Alfan Rosid, Arif Senja Fitrani, Ika Ratna Indra Astutik, Nasrudin Iqrok Mulloh, and Haris Ahmad Gozali. Improving text preprocessing for student complaint document classification using sastrawi. In *IOP Conference Series: Materials Science and Engineering*, volume 874, page 012017. IOP Publishing, 2020.

[247] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. *Advances in neural information processing systems*, 30, 2017.

[248] Dom Sagolla. *140 characters: A style guide for the short form.* John Wiley & Sons, 2009.

[249] Hassan Saif, Miriam Fernandez, Yulan He, and Harith 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)*, pages 810–817, 2014.

[250] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[251] Imanol Schlag, Paul Smolensky, Roland Fernandez, Nebojsa Jojic, Jürgen Schmidhuber, and Jianfeng Gao. Enhancing the transformer with explicit relational encoding for math problem solving. *arXiv preprint arXiv:1910.06611*, 2019.

[252] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[253] Rico Sennrich, Barry Haddow, and Alexandra 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)*, pages 1715–1725, 2016.

[254] Kanish Shah, Henil Patel, Devanshi Sanghvi, and Manan Shah. A comparative analysis of logistic regression, random forest and knn models for the text classification. *Augmented Human Research*, 5(1):1–16, 2020.

[255] Sam Shleifer. Low resource text classification with ulmfit and backtranslation. *arXiv preprint arXiv:1903.09244*, 2019.

[256] Marco Siino, Elisa Di Nuovo, Ilenia Tinnirello, and Marco La Cascia. Fake news spreaders detection: Sometimes attention is not all you need. *Information*, 13(9):426, 2022.

[257] Marco Siino, Marco La Cascia, and Ilenia Tinnirello. Whosnext: Recommending twitter users to follow using a spreading activation network based approach. In *2020 International Conference on Data Mining Workshops (ICDMW)*, pages 62–70. IEEE, 2020.

[258] Marco Siino, Marco La Cascia, and Ilenia 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)*, pages 409 – 417, Seattle, United States, 2022. Association for Computational Linguistics.

[259] Marco Siino, Elisa Di Nuovo, Ilenia Tinnirello, and Marco La Cascia. Detection of hate speech spreaders using convolutional neural networks. In Guglielmo Faggioli, Nicola Ferro, Alexis Joly, Maria Maistro, and Florina Piroi, editors, *Proceedings of the Working Notes of CLEF 2021 - Conference and Labs of the Evaluation Forum, Bucharest, Romania, September 21st - to - 24th, 2021*, volume 2936 of *CEUR Workshop Proceedings*, pages 2126–2136. CEUR-WS.org, 2021.

[260] Marco Siino, Maurizio Tesconi, and Ilenia Tinnirello. Profiling cryptocurrency influencers with few-shot learning using data augmentation and ELECTRA. In Mohammad Aliannejadi, Guglielmo Faggioli, Nicola Ferro, and Michalis Vlachos, editors, *Working Notes of the Conference and Labs of the Evaluation Forum (CLEF 2023), Thessaloniki, Greece, September 18th to 21st, 2023*, volume 3497 of *CEUR Workshop Proceedings*, pages 2772–2781. CEUR-WS.org, 2023.

[261] Marco Siino and Ilenia Tinnirello. Xlnet with data augmentation to profile cryptocurrency influencers. In Mohammad Aliannejadi, Guglielmo Faggioli, Nicola Ferro, and Michalis Vlachos, editors, *Working Notes of the Conference and Labs of the Evaluation Forum (CLEF*

*2023), Thessaloniki, Greece, September 18th to 21st, 2023*, volume 3497 of *CEUR Workshop Proceedings*, pages 2763–2771. CEUR-WS.org, 2023.

[262] Marco Siino, Ilenia Tinnirello, and Marco La Cascia. T100: A modern classic ensembler to profile irony and stereotype spreaders. In Faggioli et al. [83], pages 2666–2674.

[263] Tajinder Singh and Madhu Kumari. Role of text pre-processing in twitter sentiment analysis. *Procedia Computer Science*, 89:549–554, 2016.

[264] Kirill Smelyakov, Danil Karachevtsev, Denis Kulemza, Yehor Samoilenko, Oleh Patlan, and Anastasiya 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)*, pages 187–191. IEEE, 2020.

[265] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[266] Pascal Soucy and Guy W Mineau. A simple knn algorithm for text categorization. In *Proceedings 2001 IEEE international conference on data mining*, pages 647–648. IEEE, 2001.

[267] V Srividhya and R Anitha. Evaluating preprocessing techniques in text categorization. *International journal of computer science and application*, 47(11):49–51, 2010.

[268] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

[269] Kalpathy Ramaiyer Subramanian. Influence of social media in interpersonal communication. *International Journal of Scientific Progress and Research*, 38(2):70–75, 2017.

[270] Symeon Symeonidis, Dimitrios Effrosynidis, and Avi Arampatzis. A comparative evaluation of pre-processing techniques and their interactions for twitter sentiment analysis. *Expert Systems with Applications*, 110:298–310, 2018.

[271] Alvaro Rodríguez Sánchez and Martin Barroso Ordóñez. Profiling irony and stereotype spreaders on twitter: Pan shared task (irostereo) 2022. In Faggioli et al. [83], pages 2661–2665.

[272] C. Casula T. Ceron. Exploiting contextualized word representations to profile haters on twitter. In Faggioli et al. [84], pages 1871–1882.

[273] Narjes Tahaei, Harsh Verma, Parsa Bagherzadeh, Farhood Farahnak, Nadia Sheikh, and Sabine Bergler. Identifying author profiles containing irony or spreading stereotypes with sbert and emojis. In Faggioli et al. [83], pages 2675–2681.

[274] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1556–1566. The Association for Computer Linguistics, 2015.

[275] Hirotoshi Taira and Masahiko Haruno. Feature selection in SVM text categorization. In Jim Hendler and Devika Subramanian, editors, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*, pages 480–486. AAAI Press / The MIT Press, 1999.

[276] R. L. Tamayo, D. C. Castro, and R. O. Bueno. Deep modeling of latent representations for twitter profiles on hate speech spreaders identification task. In Faggioli et al. [83], pages 2035–2046.

[277] Luchen Tan, Haotian Zhang, Charles Clarke, and Mark 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)*, pages 657–661, 2015.

[278] Songbo Tan. Neighbor-weighted k-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 28(4):667–671, 2005.

[279] Dhaval Taunk, Sagar Joshi, and Vasudeva Varma. Profiling irony and stereotype spreaders on twitter based on term frequency in tweets. In Faggioli et al. [83], pages 2682–2686.

[280] TensorFlow. AveragePooling1D layer. `https://keras.io/api/layers/pooling_layers/average_pooling1d/`, 2021.

[281] Mike Thelwall. The heart and soul of the web? sentiment strength detection in the social web with sentistrength. In *Cyberemotions*, pages 119–134. Springer, 2017.

[282] Michal Toman, Roman Tesar, and Karel Jezek. Influence of word normalization on text classification. *Proceedings of InSciT*, 4:354–358, 2006.

[283] Alper Kursat Uysal and Serkan Gunal. The impact of preprocessing on text classification. *Information processing & management*, 50(1):104–112, 2014.

[284] Cynthia Van Hee, Els Lefever, and Véronique Hoste. Semeval-2018 task 3: Irony detection in english tweets. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 39–50, 2018.

[285] C. J. van Rijsbergen. Information retrieval, 1979.

[286] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[287] Peerapon Vateekul and Miroslav Kubat. Fast induction of multiple decision trees in text categorization from large scale, imbalanced, and multi-label data. In *2009 IEEE International Conference on Data Mining Workshops*, pages 320–325. IEEE, 2009.

[288] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[289] S Vijayarani, Ms J Ilamathi, and Ms Nithya. Preprocessing techniques for text mining-an overview. *International Journal of Computer Science & Communication Networks*, 5(1):7–16, 2015.

[290] S Vijayarani and R Janani. Text mining: open source tokenization tools-an analysis. *Advanced Computational Intelligence: An International Journal (ACII)*, 3(1):37–47, 2016.

[291] Deepali Virmani and Shweta Taneja. A text preprocessing approach for efficacious information retrieval. In *Smart innovations in communication and computational sciences*, pages 13–22. Springer, 2019.

[292] Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, and Xueqi Cheng. A deep architecture for semantic matching with multiple positional sentence representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016.

[293] Hao Wang and Jorge A Castanon. Sentiment expression via emoticons on social media. In *2015 ieee international conference on big data (big data)*, pages 2404–2408. IEEE, 2015.

[294] Sida I Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 90–94, 2012.

[295] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International*

*Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 4144–4150. ijcai.org, 2017.

[296] Matti Wiegmann, Benno Stein, and Martin Potthast. Overview of the celebrity profiling task at PAN 2020. In Linda Cappellato, Carsten Eickhoff, Nicola Ferro, and Aurélie Névéol, editors, *Working Notes of CLEF 2020 - Conference and Labs of the Evaluation Forum, Thessaloniki, Greece, September 22-25, 2020*, volume 2696 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2020.

[297] Rodrigo Souza Wilkens and Dimitri Ognibene. Mb-courage @ EXIST: GCN classification for sexism identification in social networks. In Manuel Montes, Paolo Rosso, Julio Gonzalo, Mario Ezra Aragón, Rodrigo Agerri, Miguel Ángel Álvarez Carmona, Elena Álvarez Mellado, Jorge Carrillo-de-Albornoz, Luis Chiruzzo, Larissa A. de Freitas, Helena Gómez-Adorno, Yoan Gutiérrez, Salud María Jiménez Zafra, Salvador Lima, Flor Miriam Plaza del Arco, and Mariona Taulé, editors, *Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2021) co-located with the Conference of the Spanish Society for Natural Language Processing (SEPLN 2021), XXXVII International Conference of the Spanish Society for Natural Language Processing., Málaga, Spain, September, 2021*, volume 2943 of *CEUR Workshop Proceedings*, pages 420–430. CEUR-WS.org, 2021.

[298] Louis F Williams Jr. A modification to the half-interval search (binary search) method. In *Proceedings of the 14th annual Southeast regional conference*, pages 95–101, 1976.

[299] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pages 38–45, 2020.

[300] Yijun Xiao and Kyunghyun Cho. Efficient character-level document classification by combining convolution and recurrent layers. *arXiv preprint arXiv:1602.00367*, 2016.

[301] Hiroshi Yamaguchi and Kumiko 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)*, pages 969–978, 2012.

[302] Min Yang, Wei Zhao, Jianbo Ye, Zeyang Lei, Zhou Zhao, and Soufei Zhang. Investigating capsule networks with dynamic routing for text classification. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3110–3119. Association for Computational Linguistics, 2018.

[303] Xiao Yang, Craig Macdonald, and Iadh Ounis. Using word embeddings in twitter election classification. *Information Retrieval Journal*, 21(2):183–207, 2018.

[304] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32, 2019.

[305] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[306] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[307] Wentao Yu, Benedikt Boenninghoff, and Dorothea Kolossa. Bert-based ironic authors profiling. In Guglielmo Faggioli, Nicola Ferro, Allan Hanbury, and Martin Potthast, editors, *Proceedings of the Working Notes of CLEF 2022 - Conference and Labs of the Evaluation Forum, Bologna, Italy, September 5th - to - 8th, 2022*, volume 3180 of *CEUR Workshop Proceedings*, pages 2585–2593. CEUR-WS.org, 2022.

[308] Wentao Yu, Benedikt Boenninghoff, and Dorothea Kolossa. Bert-based ironic authors profiling. In Faggioli et al. [83], pages 2720–2733.

[309] Jichuan Zeng, Jing Li, Yan Song, Cuiyun Gao, Michael R. Lyu, and Irwin King. Topic memory networks for short text classification. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3120–3131. Association for Computational Linguistics, 2018.

[310] Tianyang Zhang, Minlie Huang, and Li Zhao. Learning structured representation for text classification via reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[311] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[312] Yazhou Zhang, Dawei Song, Peng Zhang, Xiang Li, and Panpan Wang. A quantum-inspired sentiment representation model for twitter sentiment analysis. *Applied Intelligence*, 49:3093–3108, 2019.

[313] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[314] Peng Zhou, Zhenyu Qi, Suncong Zheng, Jiaming Xu, Hongyun Bao, and Bo Xu. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. In Nicoletta Calzolari, Yuji Matsumoto, and Rashmi Prasad, editors, *COLING 2016, 26th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, December 11-16, 2016, Osaka, Japan*, pages 3485–3495. ACL, 2016.

[315] Xiaodan Zhu, Parinaz Sobihani, and Hongyu Guo. Long short-term memory over recursive structures. In *International Conference on Machine Learning*, pages 1604–1612. PMLR, 2015.

[316] Yonghua Zhu, Xun Gao, Weilin Zhang, Shenkai Liu, and Yuanyuan Zhang. A bi-directional lstm-cnn model with attention for aspect-level text classification. *Future Internet*, 10(12):116, 2018.

[317] Chengqing Zong, Rui Xia, and Jiajun Zhang. *Data Annotation and Preprocessing*, pages 15–31. Springer Singapore, Singapore, 2021.

[318] José Ángel González, Lluís-F. Hurtado, and Ferran Pla. Transformer based contextualization of pre-trained word embeddings for irony detection in twitter. *Information Processing & Management*, 57(4):102262, 2020.