

UNIVERSITÀ DELLA CALABRIA



Dipartimento di Ingegneria Informatica, Modellistica,
Elettronica e Sistemistica


Dottorato di Ricerca in
Information and Communication Technologies

CICLO

XXXV

ENSEMBLE OF DEEP LEARNING PREDICTION MODELS FOR DATA ANALYTICS

Coordinatore: Ch.mo Prof. Giancarlo Fortino
Firma _____

Supervisore: Ch.mo Prof. Gianluigi Folino 
Firma _____

Dottorando: Firma oscurata in base alle linee
guida del Garante della privacy



To my parents for their love and their advice in every moment of my life.

Ai miei genitori, per il loro amore e la loro dedizione e per essere un riferimento costante nella mia vita.

To my sister and my brothers with their families for their indispensable support.

A mia sorella e ai miei fratelli insieme alle loro famiglie, per essere sempre presenti e di aiuto nella mia vita.

To my son, with the hope of leaving him a better world.

A mio figlio Bruno, affinché possa crescere sereno e nella rettitudine, sperando di contribuire a costruire un mondo migliore per lui e per tutte le nuove generazioni.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Gianluigi Folino for the continuous support of my Ph.D study, leading my research with patience, motivation, and knowledge.

Besides my advisor, I would like to thank Luigi Pontieri and Massimo Guarascio, two talented and learned researchers at ICAR CNR, who helped my PhD work with their experience, thus creating with me and my advisor a small enterprising research team.

I would like to thank the ICAR CNR and the University of Calabria for supporting me in my research activities.

Abstract

The abundance of available unstructured or raw text requires the automatic extraction of information for different tasks. One of the most relevant, Text Classification, extracts this information by assigning informative labels to raw texts from a pre-defined set.

Deep Learning (DL) offers challenging solutions to the automatic text classification problem. Despite the great potentialities of DL-based text classifiers, current solutions are exposed to a number of challenging issues that frequently occur in scenarios where text categorization is used in real-life applications. First of all, a large number of labelled data are usually necessary to train a deep model adequately, while labelling texts is time-consuming, expensive, and very often requires specific knowledge. Moreover, configuring the structure and hyper-parameters of a Deep Neural Network (DNN) architecture is a difficult task, which entails long and careful design and tuning activities to make the DNN perform well. Typical scenarios are characterized by the fact that classes are often imbalanced. These issues entail a high risk of eventually obtaining a DNN-based classifier that overfits the training data and relies on non-general, biased and unreliable classification patterns. On the other hand, the black-box nature of a DNN model does not allow for easy reasoning on which features of a data instance drove the model to its classification decision.

The work in this thesis, starting from the general problem of text classification, focuses on some challenging aspects associated with using an ensemble of deep learning methods to classify raw texts.

More in detail, this work focuses on the analysis, exploration, study and test of algorithms and learning models to be employed in the proposal of novel techniques of Ensemble Deep Learning (EDL) aimed at performing classification and explanation tasks and on the research of semi-supervised strategies based on pseudo-labelling for improving classifier prediction performances in case of scarcity of labelled data.

To this aim, this thesis proposes a complete framework based on the paradigm of ensembles of deep learning algorithms. The proposed framework is designed to furnish a valid instrument for exploring, validating and testing the proposed novel deep ensemble techniques contextualised in real-life applications, covering the entire classification process, including pre-processing, learning model building, explanation of the results, self-training for scarce labelled data, human-in-the-loop validating and model refining.

Even though the methods proposed in this work could be used in any field of interest, the problem of extracting information from the raw text was specialised for two specific application contexts: automatic customer

support ticket classification and the problem of fake detection.

The first application scenario deals with the necessity of the Customer Care Department of most companies to answer their customer requests applied as tickets through several common channels like email, short message texts, social posts, etc. Ticket classification is necessary for automatic answer generation and routing to the specific human operator.

Limiting the spread of misinformation, related to the high growth of social media dissemination and sharing of information, has raised the issue of distinguishing true news from fakes, with the challenging problem of processing long texts like news for fake detection. For this reason, the second scenario deals with the critical problem of discerning fake news from the vast amount of information circulating on the Web.

In these research areas, the ensemble paradigm has been adopted only recently; thus, discovering the possible advantages when applying this technique is challenging.

Experimental tests conducted on real data collected by two Customer Relationship Management (CRM) systems have proven the framework's effectiveness in different ticket categorisation tasks and the practical value of their associated explanations. In addition, experiments conducted on two fake news datasets have proven the effectiveness of the proposed semi-supervised self-training ensemble-based strategy for improving performances when a few labelled data are available.

Abstract (Italian version)

Il forte impulso riscontrato negli ultimi anni allo sviluppo del mondo del web e dei social media ha portato ad un'ingente crescita della diffusione di documenti in formato testuale. L'ingente mole di dati non strutturati quotidianamente prodotta ha portato alla necessità di introdurre sistemi automatici di estrazione dell'informazione. La classificazione di testi, uno dei principali task relativi all'elaborazione dei linguaggi naturali (Natural Language Processing), consiste nell'assegnare a ciascun testo un'etichetta tra un insieme predefinito di possibili classi a cui esso può appartenere.

Come ampiamente dimostrato in letteratura, le tecniche di Deep Learning (DL) offrono delle soluzioni promettenti al problema della classificazione dei testi. Nonostante le elevate potenzialità dei classificatori basati su Deep Neural Network (DNN), tuttavia, le soluzioni attualmente utilizzate in contesti applicativi reali presentano una serie di problematiche che offrono interessanti spunti per introdurre motivi di innovazione e ricerca. Un aspetto che contraddistingue il processo di apprendimento di questo tipo di modelli è la necessità di disporre di una grande quantità di dati etichettati. Purtroppo, etichettare i dati è un'operazione costosa sia in termini di tempo sia a livello economico, e in molti contesti, richiede specifiche competenze. Inoltre, per ottenere prestazioni elevate, le architetture dei modelli di DL richiedono specifiche abilità e competenze per essere ben strutturate e configurate a partire dagli hyper parameter. Spesso, nelle applicazioni reali, i dati risultano sbilanciati relativamente alla loro suddivisione nelle corrispondenti classi di appartenenza, complicando ulteriormente il processo di apprendimento. Tutti questi aspetti possono portare a problemi di overfitting relativamente ai dati usati per l'addestramento, problemi di generalizzazione, di bias e a pattern di classificazione non affidabili. Inoltre, la natura di tipo black-box dei modelli DNN rende difficile la comprensione dei meccanismi interni alla classificazione, e l'identificazione delle feature che maggiormente contribuiscono al processo decisionale.

Il lavoro di tesi, partendo dal problema generale della classificazione di testi, propone delle tecniche innovative basate su Ensemble di Deep Learning (EDL) per la realizzazione di modelli di classificazione altamente performanti, capaci di fornire una explanation dei risultati decisionali, e adatti a supportare il training anche in presenza di una quantità ridotta di dati etichettati grazie alla possibilità di utilizzare strategie di apprendimento semi-supervisionato basato sulla generazione di pseudo-label. A tale scopo, questo lavoro di tesi propone un framework completo rivolto a fornire un valido strumento per l'esplorazione, validazione e test dei modelli EDL proposti, a supporto dell'intero processo di classificazione, a partire dal pre-

processing, la costruzione del modello di apprendimento, l'explanation dei risultati, il self-training in presenza di pochi dati etichettati, e un affinamento del modello tramite un feedback attuato da operatore umano di tipo human-in-the-loop.

Sebbene le tecniche, i metodi ed i modelli proposti abbiano una valenza generale, il problema dell'estrazione di informazioni da testi è stato contestualizzato relativamente a due campi applicativi: la classificazione automatica di ticket a supporto del customer care, e la rilevazione di fake news, come specifici case study per la classificazione di testi brevi e lunghi, rispettivamente. Il primo scenario applicativo riguarda la necessità riscontrata da numerose aziende di fornire un supporto rapido, efficiente ed efficace alle richieste sollevate dai propri clienti, con l'obiettivo di minimizzare i costi dell'utilizzo di operatori umani nel servizio clienti. Il secondo scenario applicativo riguarda la ormai impellente necessità di fronteggiare la minaccia di diffusione di mis-information e dis-information diventata ormai sempre più pericolosa per la facilità con cui le informazioni circolano sul web e tramite social media. In questi contesti, il paradigma di apprendimento basato su ensemble è stato adottato solo di recente, lasciando così spazio alla possibilità di introdurre soluzioni di ricerca innovative.

La sperimentazione condotta su dati provenienti da due sistemi di Customer Relationship Management (CRM) effettivamente operanti nel settore del ticket management ha permesso di dimostrare la validità e le prestazioni del framework proposto insieme all'efficacia ottenuta dall'explanation fornita ed integrata nello schema di human-in-the-loop. Inoltre, la sperimentazione condotta su due dataset provenienti da archivi contenenti notizie vere e fake estratte dal web, ha dimostrato l'efficacia delle proposte di strategie di apprendimento semi-supervisionato dei modelli EDL con tecniche di self-training basate su ensemble nella capacità di migliorare significativamente le prestazioni del task di classificazione, anche quando la scarsa disponibilità di dati etichettati risulterebbe particolarmente inficiante il corretto addestramento secondo una strategia di tipo super-visionata.

Contents

1 Introduction	13
1.1 Thesis Overview	18
1.2 Publications	19
2 Background	20
2.1 Text Classification	21
2.1.1 Text Pre-processing	21
2.1.2 Feature Extraction and Selection	22
2.1.3 DNN techniques for Text classification	24
2.2 Ensemble Learning	34
2.3 Ticket Classification	38
2.4 Fake detection	42
2.5 Explanation Techniques	46
3 Related Works	52
3.1 Text classification in Ticket Management Systems	52
3.1.1 ML models for TMS	52
3.1.2 DNN models for TMS	55
3.2 Fake Detection	57
4 A Deep Ensemble Framework for Text Classification	60
4.1 The Software Architecture of the Framework	60
4.2 The Ensemble Strategies	63
4.3 The Explanation Technique	66
4.3.1 LIME-based Classification Explanations	66
4.3.2 The Neighbor-based Word Clouds in the Latent-Space	67
4.4 The Pseudo Labelling Strategy	70
5 Case studies: Ticket Classification and Fake Detection	72
5.1 The Ensemble-based System for Ticket Classification	72
5.1.1 The Software architecture	72
5.1.2 Exploiting Novel Efficient Deep Ensemble Classification Models for Ticket Classification	73
5.1.3 Supporting ticket classifiers with a novel Core Human-In-The-Loop scheme	76
5.1.4 Introducing an Explanation-based Analysis for better interpreting classification errors	78

5.1.5	The proposed explanation workflow with a neighbour-based artifacts representation	79
5.1.6	Complexity analysis of the Approach	80
5.2	The Ensemble-based self-trained Fake Detection Classifier . .	84
5.2.1	The ensemble strategy for effective training with few labelled data	85
6	Experimental results	91
6.1	Automatic Ticket Classification Experiments	91
6.1.1	Datasets: description and statistics	91
6.1.2	Implementation, configuration, and test procedure . .	95
6.1.3	Effect of the imbalance-aware loss function	96
6.1.4	Comparison of the proposed approach with the baselines	97
6.1.5	Comparison of our approach with state-of-the-art algorithms	98
6.1.6	Analysis of statistical significance	100
6.1.7	Explanation results	101
6.2	Fake Detection Experiments	105
6.2.1	Datasets and Parameters	105
6.2.2	Experimental validation of the pseudo-labelled based self-training proposed model	107
7	Conclusions and future works	114

List of Tables

1	ML and DL-based approaches to ticket classification and intelligent ticket management.	53
2	Main features of the <i>Phone</i> and <i>Endava</i> dataset.	92
3	Class statistics computed on the <i>Phone</i> dataset (<i>MSS</i> = Mobile Special, <i>MOS</i> = Mobile Ordinary, <i>LOS</i> = Landline Ordinary, <i>LSS</i> = Landline Special Support) and the <i>Endava</i> dataset (class: <i>Urgency</i>).	93
4	Comparing the proposed deep ensemble learning algorithms with and without the imbalance-aware loss function for the <i>Phone</i> dataset (the values in bold are significantly better than the others).	97

5	Comparing the proposed deep ensemble learning algorithms with and without the imbalance-aware loss function for the <i>Endava</i> dataset (the values in bold are significantly better than the others).	97
6	Comparing the proposed deep ensemble learning methods with the four deep learning baselines on the <i>Phone</i> dataset (the values in bold are significantly better than the others).	98
7	Comparing the proposed deep ensemble learning methods with the four deep learning baselines on dataset <i>Endava</i> (the values in bold are significantly better than the others).	98
8	Comparing the proposed deep ensemble learning methods with state-of-the-art Machine Learning algorithms, on dataset <i>Phone</i> (the values in bold are significantly better than the others).	99
9	Comparing the proposed deep ensemble learning methods with state-of-the-art Machine Learning algorithms, on dataset <i>Endava</i> (the values in bold are significantly better than the others).	99
10	Main features of the <i>PolitiFact</i> and <i>GossipCop</i> dataset.	106
11	Comparison of the pseudo-labelling strategies for the <i>PolitiFact</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).	109
12	Comparison of the pseudo-labelling strategies for the <i>GossipCop</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5 %, 5%, 10% and 20%).	110
13	Delta increment of the pseudo-labelling strategies in comparison with the baseline for the <i>PolitiFact</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).	110
14	Delta increment of the pseudo-labelling strategies in comparison with the baseline for the pseudo-labelling strategies for the <i>GossipCop</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).	111
15	Delta increment of the ensemble pseudo-labelling strategies in comparison with the pseudo-labelling solution without ensemble for the <i>PolitiFact</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).	111

16	Delta increment of the ensemble pseudo-labelling strategies in comparison with the pseudo-labelling solution without ensemble for the <i>GossipCop</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).	112
----	--	-----

List of Figures

1	RNN loop scheme.	26
2	LSTM memory cell.	26
3	Computing the hidden state in a GRU model.	27
4	An example of attention applied to the terms of a sentence.	30
5	The encoder – decoder architecture.	30
6	The transformer architecture proposed in [65].	32
7	The Scaled Dot-Product Attention module.	33
8	BERT Learning scheme.	33
9	Generic ensemble classifier scheme.	37
10	Stacking ensemble.	38
11	MOE ensemble.	38
12	The word cloud of this PhD thesis.	51
13	The proposed Framework: Conceptual Architecture.	62
14	<i>Stacking ensemble</i> architecture.	64
15	<i>MOE ensemble</i> architecture.	65
16	The pseudo labelling self-training process.	71
17	The ensemble of pseudo labelling self-trained models.	71
18	Intelligent TMS Framework: Conceptual Architecture.	74
19	<i>Stacking ensemble</i> architecture.	75
20	<i>MOE ensemble</i> architecture.	76
21	Proposed human-in-the-loop scheme for intelligent ticket classification.	77
22	Error analysis use case: exploring explanations for misclassified tickets.	78
23	Detailed flow of the explanation process.	80
24	The Fake Detection architecture.	86
25	Ticket length distribution w.r.t. the four classes for the Phone dataset.	94
26	Ticket length distribution w.r.t. the four classes for the En-dava Dataset.	94

27	Top 25 most frequent words in dataset <i>Phone</i> (entities are reported in angle brackets).	95
28	Top 25 most frequent words in dataset <i>Endava</i> (entities are reported in angle brackets).	96
29	Critical difference (CD) diagram for AUC scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a $p\text{-value} \geq 0.05$) are connected through a horizontal line.	101
30	Critical difference (CD) diagram for G-mean scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a $p\text{-value} \geq 0.05$) are connected through a horizontal line.	101
31	Critical difference (CD) diagram for F-measure scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a $p\text{-value} \geq 0.05$) are connected through a horizontal line.	102
32	LIME-based explanation obtained for test ticket x_1	102
33	LIME-based explanation obtained for test ticket x_2	103
34	Word clouds summarizing the neighbour sets $\mathcal{N}_M^{100}(x_2)$ and $\mathcal{N}_M^{100} _2(x_2)$ returned by the framework for test ticket x_2 — the latter word cloud summarizes the example tickets that are both in the neighborhood of x_2 and have the class label (namely, class=2, i.e. <i>Medium Urgency</i>) as the one predicted for x_2	104
35	Delta increment of the different strategies in comparison with the baseline solution for the <i>PolitiFact</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (1.25%, 2.5%, 5%, 10%, and 20%)	112
36	Delta increment of the different strategies in comparison with the baseline solution for the <i>Gossip</i> dataset: Accuracy, AUC and F-measure for different percentages of the training set (1.25%, 2.5%, 5%, 7.5%, 10%, 15%, 20% and 25%)	113

1 Introduction

Recently, the volume of digital text documents has grown considerably. Most of the text is in the form of unstructured raw data containing helpful information.

Text Classification is the most relevant and essential task in natural language processing, where labels represent and summarise an encoding of the information details in the text. Applying labels can be useful for several tasks depending on the final application.

Deep Learning (DL) offers challenging solutions to the automatic ticket classification problem. Despite the great potentialities of DL-based text classifiers, current solutions are exposed to several challenging issues that frequently occur in scenarios where text categorisation is used in real-life applications. First of all, a large number of labelled data are usually necessary to train a deep model adequately (the deeper and more complex the model, the more example data are needed), as most of the known training algorithms suffer from limited generalisation ability, while labelling texts is time-consuming, expensive, and very often requires specific knowledge.

Moreover, configuring the structure and hyper-parameters of a Deep Neural Network (DNN) architecture is a difficult task, which entails long and careful design and tuning activities to make the DNN perform well. Typical scenarios are characterised by the fact that classes are often imbalanced. These issues entail a high risk of obtaining a DNN-based classifier that overfits the training data and relies on non-general, biased and unreliable classification patterns hinging on spurious (incidental) features. On the other hand, the black-box nature of a DNN model does not allow for easy reasoning on which data instance features drove the model to its classification decision.

As case studies, two critical problems of automatic short and long text classification have been analysed: the automatic customer support ticket classification and the automatic fake detection, respectively.

The automatic customer support ticket classification (where the class to be predicted can correspond to intrinsic hidden properties of the tickets, like the urgency, impact, or to predefined ticket routing and resource allocation schemes) and assignment for customer support has become a hot field of application for both industrial and academic research. In fact, more and more companies are investing in extending their TMSs (Ticket Management Systems) with intelligent automatic tools to increase the efficiency and quality of ticket management processes while reducing the high costs of customer support made by human operators. A customer request opens tickets

through different channels (phone calls, emails, web forms, live chats, and recently also social media like Facebook and Twitter). In literature, many solutions have been proposed to improve the capacity of customer support systems in solving ticket issues in terms of accuracy and efficiency, exploiting traditional (shallow) ML models and Deep Neural Networks. The ML solutions [3, 26, 46, 49, 52, 53, 59] for the ticket classification have explored the entire panorama of the classic algorithms like Logistic Regression, SVM (Support Vector Machines), Naive Bayesian, SGD, KNN, decision trees and Random Forests, together with different feature extraction techniques for representing texts like Term Frequency (TF) and TF-IDF (Term Frequency - Inverse Document Frequency). Recently, DL techniques were shown to be a powerful and convenient solution for modelling accurate ticket classifiers as in [45, 60, 73] where DNN architectures like an Encoder-Combiner-Decoder and CNNs (Convolutional Neural Networks) were exploited for implementing efficient ticket classification systems. It is essential to highlight that only a few existing approaches in the field took advantage of an ensemble learning strategy [46, 49, 59] by employing standard global combination schemes like majority voting and averaging. At the same time, none of them tried to exploit DL methods to learn the base classifiers and/or an instance-adaptive combiner like in the proposed research work. On the other hand, none of the DL-based approaches proposed in the field tried to complement the class prediction task with explanatory artefacts, despite the black box nature of deep classifiers, to help customer support with advanced means for better managing and verifying the ticket processing. While ensemble approaches have been shown effective in improving shallow ticket classifiers (thanks to the higher expressiveness and generalisation power of ensemble models), the adoption of DNN ensembles is not really diffused, especially in the field of TMSs.

The second text classification case study analysed in this thesis regards the fast and widespread propagation of fake news that is becoming an emergency for properly disclosing information. Information accessibility has grown exponentially due to generalised online and social media use. A considerable amount of news is generated and manipulated daily from the traditional main press, online social systems, and personal broadcasting systems. Distinguishing the truth and veracity of the information is a fundamental task for limiting the spread of misinformation and all the possible adverse effects on society. Fakes can be very similar to real news; thus, fake detection is a critical and challenging problem. The conventional solution consisting of asking trusted professionals and specialists to check claims against evidence based on previously demonstrated or proven evident facts is time-consuming

and expensive; thus, unfeasible for the vast quantity of news on the web. The automatic fake news detection of text content is a critical and challenging NLP problem. In general, there are three main types of fake news detection: knowledge-based, content-based and context-based. Fake news detection based on knowledge is also known as fact-checking because it uses the approach of checking the authenticity of news by comparing it with given documents or web resources, focusing on the semantic web, linked open data and/or information retrieval. Content-based detection focuses on text’s content and writing style and uses two main approaches: traditional machine learning and deep learning methods. Context-based detection approaches consider not only the content of information but also many other critical factors like the source, the author, the website, the topic, the propagation path and the speed of dissemination.

The content-based text fake detection problem is considered in the proposed research. In order to achieve high-quality performances, the main requirement is to have sufficient reliable labelled data, which is time-consuming, expensive and requires specific topic knowledge. Therefore, even if semi-supervised and unsupervised methods are more appropriate in this context, most existing research still uses supervised solutions. Among the semi-supervised solutions present in literature, [55] proposes a method for detecting deceptive and fake opinion reviews, which is primarily based on co-training and expectation maximisation; [24] proposes a framework based on three main modules, the tensor-based embeddings of the article text, the graph representation based on the K-Nearest Neighbor method and finally the belief propagation using the Fast Belief Propagation (FaBP) Network; while [42] proposes a Convolutional Neural Network semi-supervised framework built on the concept of the temporal ensemble. Analysing the current solutions present in literature for fake detection, the following main limitations can be highlighted: (i) most of the state-of-the-art research is focused on solving the problem with supervised techniques, which requires a considerable amount of labelled data; (ii) human annotating high volumes of data is too much time-consuming and expensive while requiring specific knowledge; (iii) extensive data labelling, crowd-sourcing or human expert annotations creates lots of annotation inconsistencies.

My research activity proposes a comprehensive classification framework for the entire processing flow of text classification, which relies on training a novel kind of ensemble of deep classifiers and on deriving different easy-to-interpret types of artefacts for explaining and debugging the predictions of the ensemble providing AI-based interpretation methods. The proposed framework is based on an ensemble deep neural network classifier leverag-

ing different types of DNN architectures (based on LSTM, GRU and Transformer architectures) as base classifiers of the ensemble model to increase the diversity, expressiveness and robustness to over-fitting and class-imbalance risks. The framework introduces two novel ensemble combination strategies based on stacking and mixture-of-expert architectures, both leveraging an ad hoc sub-net for extracting dense (latent space) text representations used to learn instance-adaptive ways of fusing the predictions of the base models. An explanation module is integrated with the framework to support a continuous, human-in-the-loop scheme for discovering, using, validating and improving the EDL model. Two kinds of summary explanations are provided. The first, based on the post hoc explanation LIME algorithm [54], furnishes the subset of the terms that influenced most of the classification results. The latter provides a word-cloud representation of the most similar text instances, together with their corresponding label, through the introduction of an “explanation” index structure built for each ensemble model for keeping a representative number of example (class-labelled) text instances in a form enabling fast similarity searches over the latent space.

The proposed framework also introduces a novel technique that exploits the idea of pseudo-labelling in the context of semi-supervised learning [4, 10, 29, 34] for improving performances when training a deep learning model with a small training set of labelled data. The proposed pseudo-labelling strategy uses predictions as pseudo-labels of the unlabelled tuples by repeating a process of self-training cycles, where training is obtained by exploiting a combination of the labelled samples and any previously pseudo-labelled samples. The used learning model is based on BERT (Bidirectional Encoder Representations from Transformers) followed by a Dropout layer for regularisation and a final dense layer with a sigmoid activation layer. Moreover, also the self-training process exploits ensemble strategies to improve the final classifier performances by considering the several models obtained at the different training iterations as base learners.

Several tests assessed the effectiveness and usefulness of the proposed framework. In particular, two real-life ticket datasets were used for the ticket classification application, *Phone* and *Endava*. An Italian phone company furnishes the *Phone* dataset; the text content of these tickets comes from two channels: SMS messages and Facebook chats. The *Endava* dataset is a publicly available dataset containing about 50K tickets submitted via email by the customers of the *Endava* company to the helpdesk.

Tests for the fake detection application have been made by using the fake news data repository named *FakeNewsNet* [58], which contains two comprehensive datasets of political and gossip news obtained by fact-checking

websites as PolitiFact (<https://www.politifact.com/>) and GossipCop (<https://www.gossipcop.com/>), respectively.

1.1 Thesis Overview

The thesis is organized as described in the following.

Chapter 2 introduces an accurate background covering the main topics treated in the thesis, like the problem of text classification in general, state of the art on ensemble learning, the ticket classification problem, and some motivations about the explanation necessity in deep learning classification.

Chapter 3 shows the most correlated works in scientific literature addressing the text classification in the two real application contexts: the ticket classification for customer support and fake detection.

Chapter 4 presents the architecture of the proposed framework with a detailed description of the main modules, including the proposed ensemble strategies for combining the base classifiers, the explanation process and the self-training strategy based on pseudo labelling.

Chapter 5 presents the two implemented architectures of the proposed framework specialized for the two specific text classification real case studies: the TMS test classifier architecture and the Fake detection architecture.

Chapter 6 discusses the experimental results obtained by testing the framework by using real datasets, thus proving the effectiveness of the proposed approaches.

Finally, Chapter 7 reviews the contributions of this work and outlines future research to be conducted.

1.2 Publications

The relevant publications inspired by the proposed research are listed below:

- P. Zicari, G. Folino, M. Guarascio, L. Pontieri, “Discovering accurate deep learning based predictive models for automatic customer support ticket classification”, in: Proc. of 36th Annual ACM Symposium on Applied Computing, 2021, p. 1098-1101.
- Paolo Zicari, Gianluigi Folino, Massimo Guarascio, Luigi Pontieri, “Combining deep ensemble learning and explanation for intelligent ticket Management”, Expert Systems With Applications, Volume 206, 2022, 117815, ISSN 0957-4174, <https://doi.org/10.1016/j.eswa.2022.117815>.
- P. Zicari, G. Folino, M. Guarascio, L. Pontieri, “Learning deep fake-news detectors from scarcely-labelled news corpora”, in Proc. of 25th International Conference on Enterprise Information Systems (ICEIS 2023), Prague, Czech Republic, 24 – 26 April 2023.

2 Background

This section is aimed to furnish a useful background of the main topics covered in the thesis.

More specifically, the complete process flow of the text classification is analyzed in detail. A specific focus is dedicated to introducing and explaining the deep learning techniques employed in the thesis. The Ensemble Learning subsection furnishes an exploration of the main ensemble techniques. Moreover, a background on the importance of the ticket classification task in the customer support application field is also furnished. Finally, the fundamental role of the explanation in the classification systems based on the use of deep neural networks is here argued, highlighting the main properties of the current explanation techniques proposed in the research studies of the last few years.

2.1 Text Classification

Text classification is the most important and essential task in natural language processing [35], used as a basic task in more complex applications, such as sentiment analysis, topic labelling, information retrieval, question answering and dialogue act classification. Text classification consists in extracting features from raw text data and predicting the categories of text data based on such features.

In recent years, the exponential growth of digital documents available through the web and social networks has pushed towards a rapid progress in the development of automatic text classification techniques. In fact, the idea of manually processing and classifying the huge amount of text in the actual era of the information explosion is absolutely unfeasible due to time, energy and cost constraints. Moreover, the automation of text classification can benefit from more reliability and, above all, less subjectivity than human efforts.

Input data for natural language tasks like text classification consist of raw, unstructured text, that, unlike numerical, image, or signal data, requires specific NLP techniques to be processed carefully, with still open challenging problems related to contextual words understanding, homonyms disambiguation, synonyms, irony and sarcasm discerning, ambiguity, the presence of syntax and semantic errors, colloquialisms and slang, domain-specific languages.

In the following sub-sections, the main conventional text classification tasks consisting of preprocessing, feature extraction, feature selection, and classification stages are analyzed.

2.1.1 Text Pre-processing

Text pre-processing is a fundamental data preparation step that requires special attention, especially in informal contexts when the text is in the form of chat messages where spelling and grammatical rules are often neglected, sentences are often very short, and punctuation is confusing. The pre-processing performs a set of operations including the cleaning of text from errors (misspelling), as well as removing space and stop-words. Moreover, the following preprocessing tasks are also performed to convert texts into sequences of tokens:

- **Tokenization:** As a form of text segmentation, the text is split into words, phrases, or other meaningful parts, namely tokens. Typically,

the segmentation is carried out considering only alphabetic or alphanumeric characters that are delimited by non-alphanumeric characters (e.g., punctuation and white space).

- **Lowercasing:** Upper case letters in each word are converted into lower case ones, in order to normalize the representation of a concept, while also reducing sparsity and vocabulary size.
- **Lemmatizing:** Tokens are replaced with the corresponding lemma, so that different forms of the same lemma are all uniformed to the same root form, representing a single distinguished vocabulary entry.
- **Stemming:** As a simpler alternative to lemmatization, each token can be replaced with the corresponding stem, representing a chopped-off form of the former so that words with the same stem have the same representation even if this is not the dictionary form.
- **Stopwords removal:** Commonly used words, usually known as stopwords, can be purged off since they are likely to convey a low amount of information; this also allows for reducing the number of features and vocabulary size.
- **Named entity recognition:** Words that can be classified as named entities like places, organisations, people, time expressions, quantities, monetary values, etc., are identified and eventually replaced with an entity token in order to generalize specific instances.
- **Noise removal:** Characters, digits and pieces of text that can interfere with the analysis of the text are removed.
- **Anonymization:** Privacy protection is obtained by removing personally identifiable information so that anonymized data cannot be associated with any individual.

2.1.2 Feature Extraction and Selection

Textual data, unlike other types of data such as images or temporal series, does not have an intrinsic numerical representation, but, needs to be projected into an appropriate numerical feature space before being classified. Thus, after preprocessing, the text is transformed into a list of separated and standardized tokens. All the tokens of the entire corpus, representing preprocessed terms, are mapped to an index-based vocabulary furnishing a numerical internal representation. Tokens represented by the corresponding

index in the vocabulary of the corpus are then transformed into a tensorial form suitable for being processed by the classifiers. Many algorithms for representing texts in the feature space have been proposed in the literature; this section provides an overview of the most popular and effective ones.

- **Bag-Of-Words (BOW):** Also called Term Frequency (TF), it is one of the most simple techniques of text extraction, where a text is considered as an unordered collection of terms (tokens), with the clear disadvantage of ignoring sentence structure and semantic relationships among the sentence elements. Each token is represented as a one-hot-encoded vector with the same size as the vocabulary. Each element of the vector is the number of times that the term occurs within a text. Another disadvantage of this approach is the direct correlation between the cardinal size of the vocabulary with the size of the term vectors, which can be problematic to manage when the vocabulary grows it happens in the case of big corpora of texts.
- **Term Frequency-Inverse Document Frequency (TF-IDF):** it uses the word frequency and inverses the document frequency to model the text. TF is the frequency of a term in a specific text, while IDF is the reciprocal of the proportion of the texts containing this term to the total number of texts in the corpus. In this way, the relevance of a term in a text increases proportionally with the number of times (TF) it appears in the text itself, but the relevance decreases when the term occurs in many texts (IDF) lessening the effect of common terms.
- **N-Gram:** it assigns probabilities to sequences of words, for the language modelling task of predicting the likelihood of a string given a sequence of preceding or surrounding context words. By adopting the Markov assumption, the probability of an upcoming word in a sentence is considered to be only depending on the previous N words. The *BOW* model can be seen as an *N-Gram* model with $N = 1$.
- **Word Embedding:** In order to overcome the limitations of the previous language representation models, able to take into account only the syntactic representation of a term and in some cases the syntactic relationship of the near terms in a sentence, *Word Embedding* furnishes a tensor representation of a term, able to capture its semantic meaning. An embedding is an n -dimensional vector representation of a term which encodes its semantic meaning. The mapping operation between each term and its corresponding embedding representation

is obtained by a learning process usually based on neural networks trained to catch the meaning of the term from its surrounding words in a sentence. The distance in the embedding n -dimensional space between two terms is representative of their semantic similarity.

- **Word2Vec:** it employs local context information to obtain fixed-length real value vectors specified as the word vectors for any word in the corpus. This approach utilises shallow neural networks with two architectural variants: the Continuous-bag-of-words (CBOW) [44] and the continuous Skip-gram [43]. CBOW learns word representations by trying to predict a word based on its surrounding context words, the bag-of-words reference in its denomination is referred to the fact that the order of the context elements is not actually taken into account. The continuous skip-gram model, instead, works in the opposite way, attempting to predict the neighbours (the context), given a word.
- **GloVe:** Global Vectors for Word Representations (GloVe) [48] is an embedding technique conceptually similar to the Word2Vec but differing from it mainly for being a count-based model instead of a predictive model like the standard Word2Vec. While embedding predictive models learning, like Word2Vec, tries to minimise the loss between target and prediction, given the context words and the vector representations. As a count-based model, it essentially learns semantic similarity between words by using the underlying statistics in the corpus, such as words co-occurrence. The locality limitation of considering only the local information (the context of each word) is overcome by GloVe thanks to the fact that embeddings are trained by considering also global co-occurrence statistics. GloVe model uses dimensionality reduction in order to reduce the large dimensions of the word co-occurrence matrix that it uses in its calculations.

2.1.3 DNN techniques for Text classification

In the last few decades, there has been a great interest in the text classification area due to the unprecedented success of deep learning techniques.

This section offers a background overview of the Deep Neural Networks employed in the experiments conducted for testing the proposed framework.

The following reported DNNs architectures that currently represent the state of the art in the text classification task are LSTM, GRU, CNN, Transformers and BERT.

LSTM. The Long Short-Term Memory (LSTM) was proposed in [27] as an improved version of conventional Recurrent Neural Networks (RNNs). RNNs are neural networks which use sequential data or time series data. They are characterized by a memory unit which is a neuron with a recurrent self-connection included in it as reported in the scheme of Figure 1. The loop inside the network allows for information to persist. Information from prior inputs influences the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of recurrent neural networks depends on the prior elements within the sequence. RNNs present two problems known as exploding gradients and vanishing gradients. These issues are defined by the size of the gradient, which is the slope of the loss function along the error curve. When the gradient becomes too small such that the weight parameters during the learning phase of the training become insignificant (i.e. very close to 0), the algorithm is no longer learning. Exploding gradients occur when the gradient is too large, creating an unstable model; in this case, the model weights will grow too much. RNN suffers from the problem of long-term dependencies, that is, if the previous state that is influencing the current prediction is not in the recent past, the RNN model may not be able to accurately predict the current state. LSTM was introduced to solve the vanishing gradient problem, addressing the problem of long-term dependencies in order to remember information for long periods of time. To this aim, LSTMs have “cells” in the hidden layers of the neural network as shown in Figure 2, which have three gates: an input gate, an output gate, and a forget gate. These gates control the flow of information which is needed to predict the output in the network. These gates are composed of a sigmoid neural net layer (σ) and a pointwise multiplication operation (\times). The sigmoid layer outputs numbers in the range $[0, 1]$, describing how much of each component should be let through.

GRU. The Gated Recurrent Unit (GRU) was proposed in [12]. GRU has gating units that modulate the flow of information inside like the LSTM unit, but without having separate memory cells. In GRUs, each recurrent unit adaptively captures dependencies of different time scales. GRUs are improved versions of standard recurrent neural networks that use two special

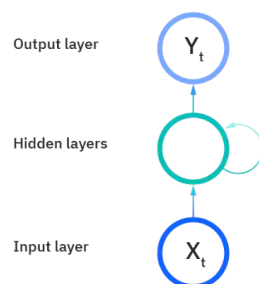


Figure 1: RNN loop scheme.

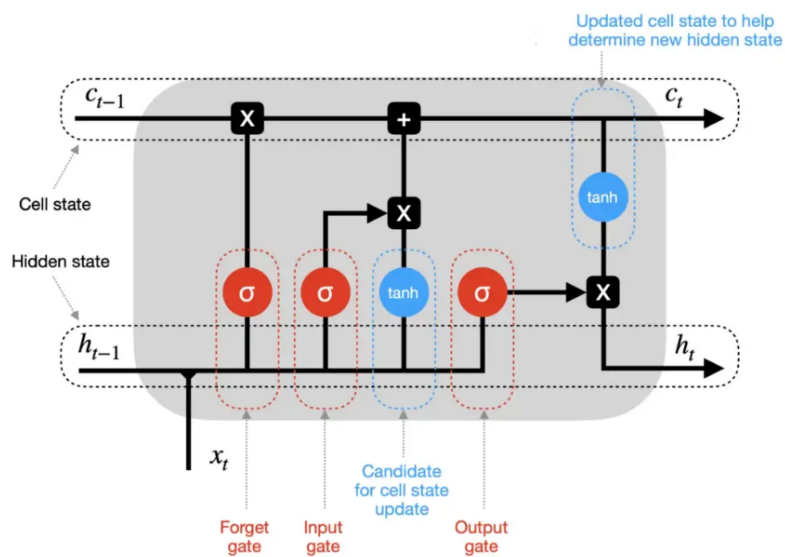


Figure 2: LSTM memory cell.

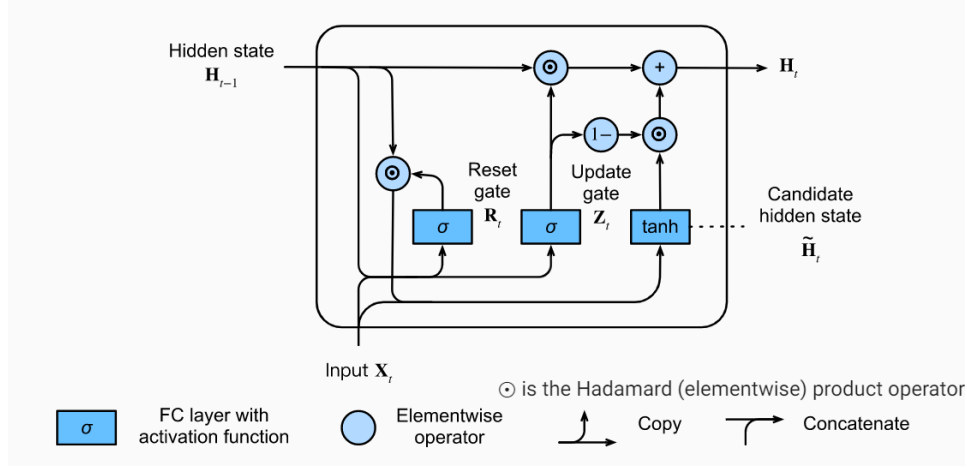


Figure 3: Computing the hidden state in a GRU model.

gates called the update gate and reset gate, instead of the three gates used in the LSTM. These two gates have the function to decide what information should be passed to the output, and they are trained to keep information from the past or remove information which is irrelevant to the prediction, thus solving the vanishing gradient problem of a standard RNN (Recurrent Neural Network). The reset gate R_t controls how much of the previous state it is desirable to remember, the update gate Z_t would control how much of the new state is just a copy of the old state. These gates implement a sigmoid activation, forcing their values to lie in the interval $(0, 1)$. With respect to the LSTM, GRU combines the forget and input gates into a single update gate, moreover, it merges the cell state and the hidden state, resulting in a simpler model. Figure 3 illustrates a GRU, with the current time step X_t and the hidden state of the previous time step H_{t-1} as input, and the new hidden state H_t as the output; the outputs of two gates are given by two fully connected (FC) layers with a sigmoid activation function.

Convolutional Neural Network (CNN). Convolution is a concept widely employed in image processing because it is able to catch the local features of the pixel representation of the images, and then applied hierarchically to find relationships at more distances. The operation of convolution is applied between a certain number of inputs and a convolution mask of learned weights, able to highlight dependencies among them. Convolution is used for extracting local information while learning dependencies among distant

positions becomes more difficult because it is necessary to climb the pyramid structure with a depth depending on the distance. CNN-based models are trained to recognize patterns in text. CNN-based models, usually, are made of several one-dimensional $(1 - D)$ convolution layers followed by one-dimensional max-pooling operations to extract a feature vector from the input word embeddings, ending with a classification layer for classification purposes. Input to CNNs is a word matrix where each row represents an embedding word vector. One-dimensional convolution operations on n embedding word vectors are able to extract n -gram based features. By using different sizes and convolution mask filters at different network layers, it is possible to create several architectures with different performances. Moreover, repeated stacked convolutional layers with max-pooling are able to capture the short and long-range relationships between the terms. Several CNNs models have been proposed in the literature for text classification, starting from the first simple implementation in [32], the Dynamic CNN in [31], arriving to the most recent architectures like *TextConvoNet* in [61] that uses a 2-dimensional convolutional filter to extract the intra-sentence and inter-sentence n -gram features from text data. The ability to find relations among terms in different position of sentences and texts, in general, makes CNNs favourable and prone to be used for sequence-to-sequence learning as in [23] and in the 2-D convolutional network proposed in [18].

Transformers and the attention mechanism. Transformers are a novel approach based on the concept of self-attention that are able to model sequences without using recurrence and convolution. Sequence modelling is the capability to take into account as much as possible previous input, possibly with the right consideration with respect to some kind of relationship among past and present inter-dependency, in order to predict the output sequence. Being a sequence, also the output prediction/generation has to take into account the previously predicted or generated outputs. The formulated problem requires a context-aware solution able to disambiguate situations that could not be solved otherwise. This is the case of the well-known Winograd schema challenge where machine intelligence is required to solve intriguing problems like the task to identify the noun referred to by a pronoun in an ambiguous sentence where only the union of context analysis together with some kind of knowledge of the terms meaning is able to disambiguate the relation between the pronoun and the right noun. The following sentences are evident examples of this kind of problem:

- “The animal didn’t cross the street because it was too tired”

- “The animal didn’t cross the street because it was too wide”

In the first sentence the “it” pronoun is referred to the animal while in the second sentence, it is related to the street. Both sentences contain two nouns: animal and street. The two adjectives, tired and wide, offer the disambiguation key to making the right correlation between the pronoun and the right noun. The fact that a street cannot be tired and the wide adjective is never used to describe an animal gives the solution to the problem.

Attention is a simple mechanism aimed to represent a sequence taking into account all the dependencies between all the pairs. The easiest way to understand the attention concept is to think about a similarity matrix where rows and columns could represent the same sequence (self-attention) or two different sequences. The generic element $W_{i,j}$ in position (i,j) of the similarity matrix W gives information on the level of relationship between the element in the i – th row and the element in the j – th column.

Self-attention is useful to help understand the disambiguation in anaphora or, in general, to find the relationships among all the elements of a sentence, in both directions, considering the elements before and after. If the attention mechanism is applied to the sentence “The animal didn’t cross the street because it was too tired” it would be desirable to have a matrix like the one reported in Fig. 4, where the different grey levels represent the different levels of relationship between the term pairs; evidencing that among it pronoun, the animal noun and the tired adjective there is a higher dependency.

Transformers are modeled as a typical encoder-decoder architecture like the one shown in Fig. 5, where the encoder maps an input sequence of symbol generically represented with $x = (x_1, \dots, x_n)$ to an intermediate representation resulting in a sequence $z = (z_1, \dots, z_k)$. From the intermediate representation z , the decoder then generates the output sequence (y_1, \dots, y_m) of symbols one element at a time.

The first proposal of an attention-based transformer in literature is by [65] with the architecture shown in Fig. 6. The transformer encoder processes the input sequence that flows through a fixed number of identical layers, the specific implementation in [1] uses 6 layers, where each layer is composed of an attention module called multi-head attention followed by a feed-forward neural network. The output of the encoder is inputted to the decoder, the latter replicates the layers of the encoder, just adding one more multi-head attention used for the output of the decoder. Thus, the attention approach in transformers is used in three different ways, a self-attention in the encoder finds the relationships between all pairs of the input in the processed sequence during the encoding phase, a self-attention in the decoder finds the

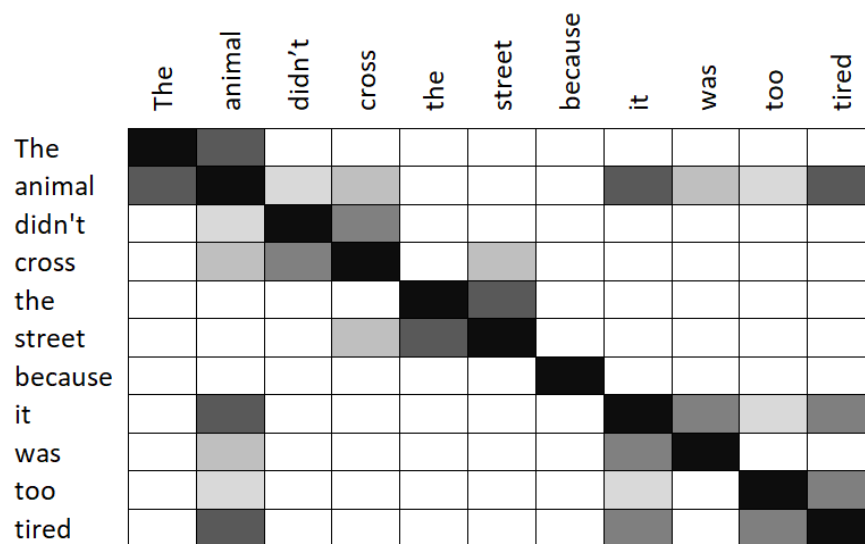


Figure 4: An example of attention applied to the terms of a sentence.

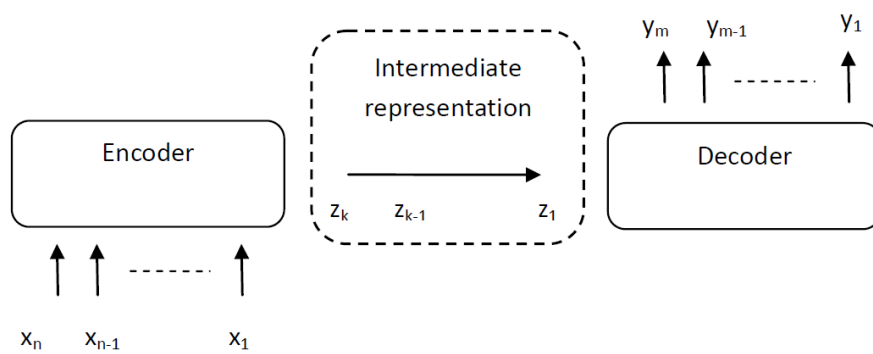


Figure 5: The encoder – decoder architecture.

relationships between the current decoded output and all the previous ones in the sequence, moreover, attention is computed between the encoder and the decoder in order to find the relationships between each decoded element and all the encoded input sequence.

The multi-head attention module performs in parallel several different attention computations. The resulting attention computations represent the relationships among the encoded and decoded flowing information. Moreover, all the representations are persistent during the flow because they are concatenated together. The Positional Encoding module is introduced in order to differentiate the positions in the flow of the embedding input elements, the implemented functions are the sine and cosine.

The multi-head attention module consists of multiple Scaled Dot-Product Attention modules shown in fig. 7 computing the dot products and softmax of the Q , K and V matrices, where V is the matrix of the value vectors, K is the matrix of the key vectors and Q is the matrix of the query vectors. The strength of the transformers algorithm is based on the fact that there is no use of recurrence while the main computational cost is affected by the attention calculus that is based just on matrix multiplications that can be efficiently executed in parallel by GPUs.

BERT. Bidirectional Encoder Representations from Transformers (*BERT*) [14] is a transformer-based neural architecture able to process natural language. It is trained through an algorithm including two main steps, respectively named *Word Masking* and *Next Sentence Prediction* (NSP). In the former step, a percentage of the words composing a sentence is masked, and the model is trained to predict the missing terms by considering the word context, i.e., the terms that precede and follow the masked one. Then, the model is fine-tuned by considering a further task that allows for understanding the relations among the sentences. Different datasets can be used to perform this phase e.g., *SQuAD* [50], *NER* [63] and *MNLI* [68]. An overview of this learning procedure is depicted in figure 8. In particular, tok_i is referred to the generic token extracted from the text, while E_i is its internal representation inside the Bert model, and T_i is its output representation yielded by the model. Basically, given two subsequent sentences, negative examples are created by replacing the second with a random sentence. As regards the architecture, BERT can be figured out as a stack of transformer encoder layers that include multiple self-attention “heads” [14].

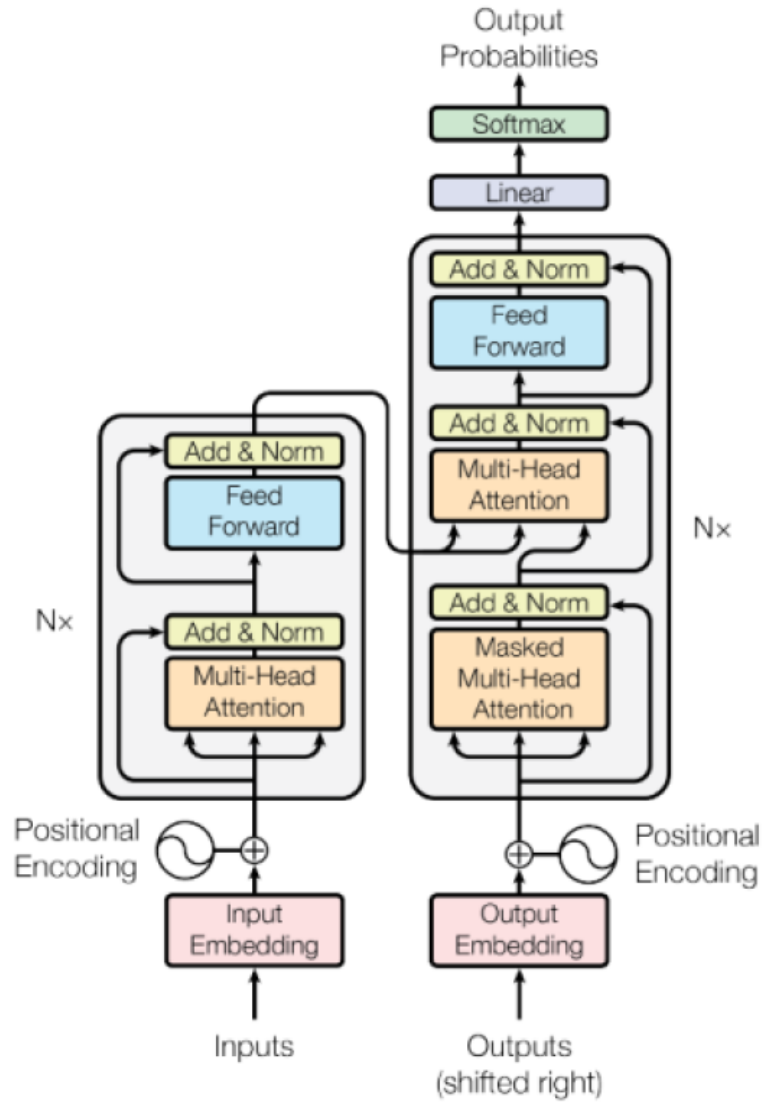


Figure 6: The transformer architecture proposed in [65].

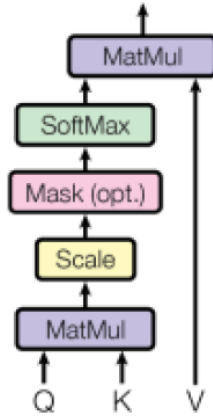


Figure 7: The Scaled Dot-Product Attention module.

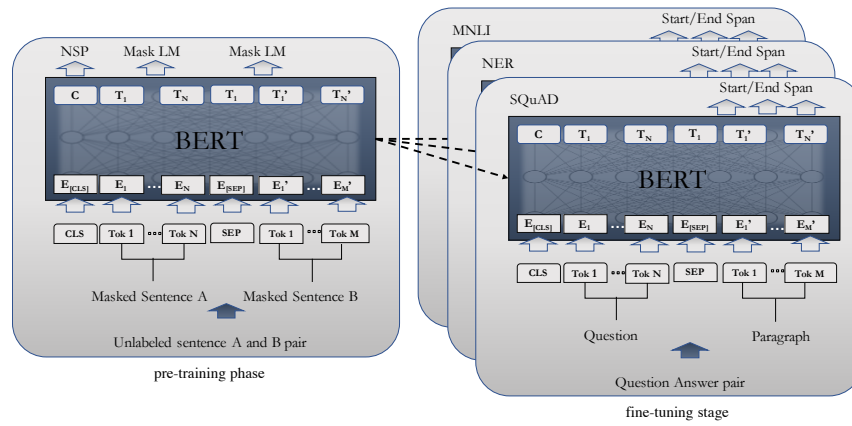


Figure 8: BERT Learning scheme.

2.2 Ensemble Learning

Ensemble learning is a machine learning field focused on improving performances by combining multiple learners. In [15], an ensemble (or committee) of learners is defined as a set of learners whose individual decisions are combined in some way to classify new examples. The main idea behind the concept of ensembles is to build a certain number of (heterogeneous or homogeneous) classifiers trained with subsets (proper or not, disjoint or not) of the original training set and then combine their predictions with some function and then furnish a common decision that takes into account all the base classifiers in different ways. The final model is proved to have improved characteristics with respect to the single base learners in terms of better generalization performances, trying to reduce the bias and the variances among the base learners. Ensemble exploits the diversity of weak learners to build strong learner following the principle that diverse classifiers make different decision errors that can be corrected by the other ones in the combiner (for example, when voting or average is used), thus reducing the risk of choosing the wrong classifier. In fact, a necessary and sufficient condition for an ensemble to be more accurate than any of its individual members is that the classifiers must be accurate (with an error rate better than random guessing) and diverse (the errors made by the classifiers are uncorrelated) as stated in [25]. Moreover, most of the learning algorithms work by performing some form of local search that may get stuck in local optima. An ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers.

In the last decades, the machine learning research community has been developing new approaches to generate, combine and test ensembles of models. The different schemes vary depending on the generation of the base classifiers and the ensemble combination, i.e. the same learning algorithm can be trained on different datasets or/and different algorithms can be trained on the same dataset. The generic scheme of an ensemble classifier is reported in Figure 9, the *Partitioner/Dispatcher* block is aimed to define the strategy for the dispatching of the subsets of the training set to the different classifiers. The *Combiner* block can be a trainable or not trainable function of the base classifier predictions. The ensemble with trainable combiners has the possibility to use the input instances of the training set together with the base classifier predictions in order to learn when each base classifier is supposed to make the correct decision finding a direct correlation between the input instances, the model predictions and the correct classification for

each base model for a more complex ensemble combination of the single base decisions.

Bagging [8], also known as bootstrap aggregating, is one of the standard techniques for generating ensembles based on the method of manipulating the training dataset. In general, the techniques that run a learning algorithm several times, each time with a different subset of the training set works especially well for "unstable" algorithms, like decision trees and neural networks, i.e. those algorithms whose output undergoes major changes in response to minor changes in the training data. The main idea of bagging is to generate a series of independent observations with the same size and distribution as that of the original data. After generating the bagging samples and passing each bag of samples to the base models, the predictions of the multiple predictors are combined in different ways. The majority voting is mostly used for classification problems while the averaging strategy is used in regression problems for generating the ensemble output. A bagging subset is generated by sampling with replacement from the original training set, with several training examples appearing multiple times. Such a transformed training set is called a *bootstrap replicate* of the original dataset, and the technique is called bootstrap aggregation, from which the bagging term is derived. When the training set sampling method uses disjoint subsets of the original training set, the ensemble is called *cross-validated* committee.

Random Forest [7] combines the output of multiple decision trees to reach a single ensemble result, exploiting the bagging strategy for improving the predictions of the base classifiers consisting of decision trees. Decision trees are prone to problems, such as bias and over-fitting; however, when multiple decision trees form an ensemble in the random forest algorithm, they predict more accurate results, particularly when the individual trees are uncorrelated with each other. The random forest extends the bagging method as it utilizes both bagging and feature randomness to create an uncorrelated forest of decision trees. Feature randomness generates a random subset of features, which ensures low correlation among decision trees. The fundamental difference with respect to the traditional decision tree classifier consists in the fact that at each tree split in Random Forest, only a subset of features is randomly selected and considered for splitting, thus promoting the decorrelation of the trees and preventing over-fitting. Thus, two levels of randomness are exploited in Random Forest, the bootstrap sampling consisting in drawing samples from a training set with replacement, and the feature bagging consisting of using random subsets of features.

AdaBoost [19] is another ensemble method based on manipulating the training set for generating multiple hypotheses. AdaBoost maintains a set of

weights over the training examples; at each iteration, the learning algorithm is invoked to minimize the weighted error on the training set and to return a prediction. A weighted error of the hypothesis is computed and applied to update the weights on the training examples. The effect of the change in weights is to place more weight on training examples that were misclassified and less weight on examples that were correctly classified. The final classifier is constructed by a weighted vote of the individual base classifiers; each classifier is weighted according to its accuracy on the weighted training set.

Gradient boosting [20] is another popular boosting ensemble technique formulated as a numerical optimization problem where the objective is to minimize the loss of the model by adding weak learners using a gradient descent-like procedure. Decision trees are used as the weak learner in gradient boosting. Trees are added iteratively one at a time. A gradient descent procedure is used to minimize the loss when adding trees. In the functional gradient descent approach, after calculating the loss, a new opportunely parameterized tree is added to the model in order to reduce the loss and then the gradient at the following iteration. The output of the new tree is then added to the output of the existing sequence of trees in order to correct or improve the final output of the model. The ensemble model training stops when a fixed number of trees are added or when the loss reaches an acceptable level, or no longer improvements are reached on a validation dataset.

Stacking or Stacked Generalization is an ensemble technique that combines the predictions from multiple machine learning models by using another machine learning model that is trained to learn how to better combine the different base models' predictions for achieving the best performances. The main characteristic of stacking is that the combiner is a machine learning algorithm, also called a meta-learner, i.e. a learning algorithm that learns from other learning algorithms. The architecture, as shown in Figure 10, is built on a two-level hierarchy, where the first level consists of the base models, usually diverse machine learning models having different prediction 'skill', making different assumptions about the prediction task, and uncorrelated to each other, while the second level is represented by the meta learner. When using stacking for prediction, while the first level is trained by using the training set, the second level uses the probability predictions generated by the base models for training. Different more or less complex algorithms can be used for implementing the meta-learner; when a simple linear model like Logistic Regression is used, it is also called *blending*.

MOE (Mixture of Experts) was introduced in [30]. This ensemble technique, shown in Figure 11, is based on the divide-and-conquer principle in which the problem space is divided among different experts (learners) oppor-

tunely supervised by a gating network. Different strategies were developed to divide the problem space between the experts. Moreover, different error functions in the learning process are used to localise the base experts in different distributions of data space. The combiner is a gating network aimed at modelling the local competence of the experts in different distributions of data space according to each input data. The gating network allows the mixing proportions of the experts to be determined by learning a partition of input space and trusting one or more expert(s) in each of these partitions.

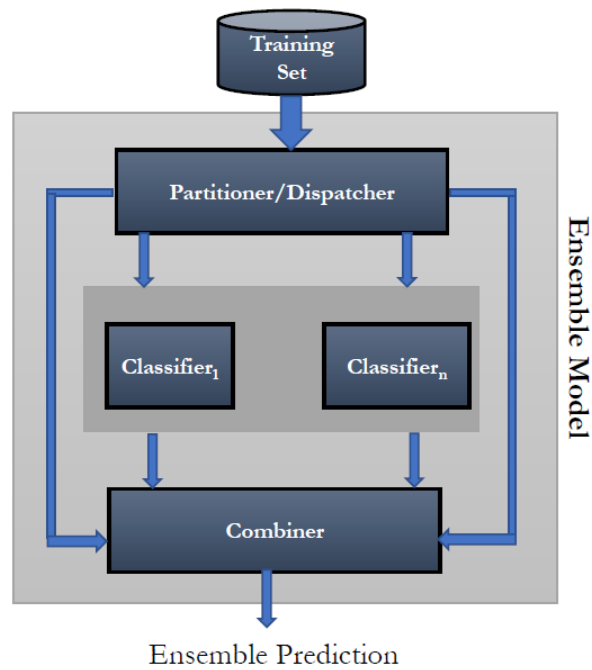


Figure 9: Generic ensemble classifier scheme.

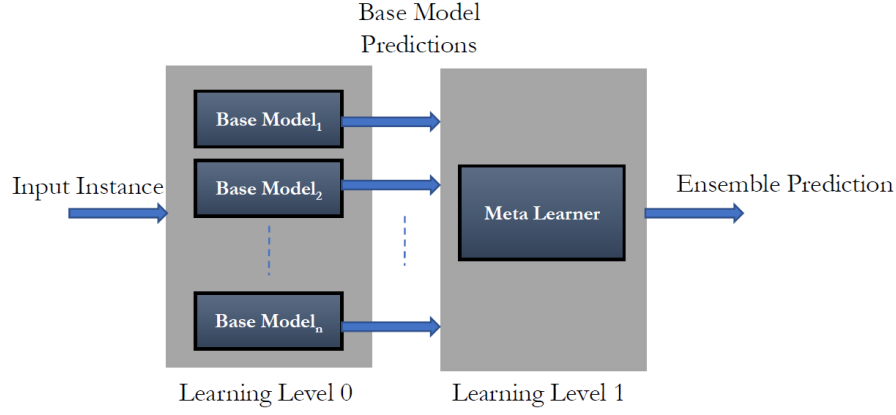


Figure 10: Stacking ensemble.

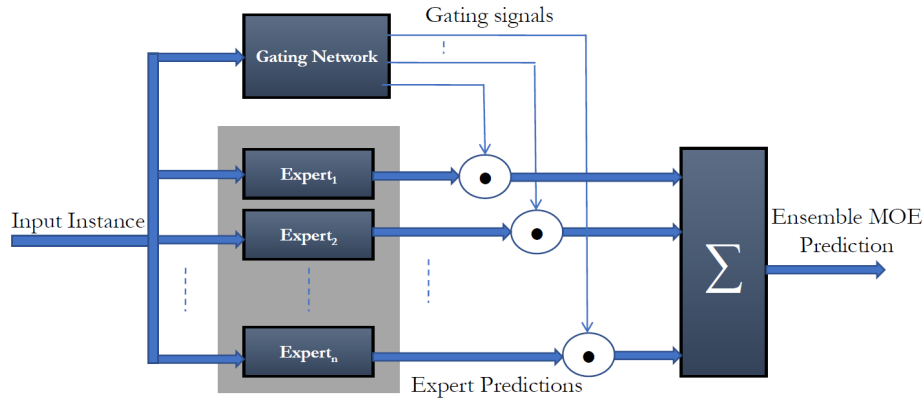


Figure 11: MOE ensemble.

2.3 Ticket Classification

Customer support is a key functional area of modern enterprises, usually supported by some *Customer Relationship Management (CRM)* platform, which has been paid increasing attention in recent years. Nowadays, many companies made up of specialized customer service departments employ dedicated personnel to provide effective support to their customer base.

Tickets, also named *cases* or *issues*, are opened by a customer request through different channels. A multi-channel ticketing system collects all

support tickets from different channels and organizes them in a single view. The most common channels for customer support use *emails*, *phone calls*, *web form*, *live chat*, and now also *social media* like Facebook and Twitter in order to have easy and user-friendly communication with media increasingly used by younger people.

The integration of requests from different channels helps the customer support system to better manage the tickets and offer more personalized assistance.

Even if customers feel more comfortable when issues are solved with human interactions, with kind and helpful agents taking care of them, it implies very high costs to bear for a company that wants to offer this kind of support in an efficient and effective way, above all when the number of customers becomes very high. A great deal of effort is focused in finding the right compromise between cost reduction and the improvement of customer satisfaction in terms of service quality, quick response and resolution of issues with minimal stress.

Ticket Management Systems (TMSs) are a precious technical solution in this area, based on organizing customer support activities in ad hoc business processes, which consists in handling specific *tickets* (a.k.a. *issues*) for customers' requests (coming from e-mails, phone calls, web forms, live or social media chats). Improving these processes (e.g., by prioritizing urgent tickets, and enabling competency-aware ticket allocation) can bring great benefits in terms of cost reduction, customer retention and reputation strengthening.

Some of the main tasks involved in this process include the automatic categorization of tickets, prioritization on the basis of some urgency criteria, identification of the customer and automatic tracking of the his/her history, assignment to the right agent of the customer support team, suggestion to the agent of the best solution among possible solutions selected from a historical archive preserving base knowledge of the previous cases, or eventually generate the automatic answer to be sent back to the customer as a solution of the ticket, generating report and analytic surveys.

Customer satisfaction increases when any request issue is resolved very fast, above all for critical time problems, the proposed solution is right and preferably the best among all the possible ones, customers feel that the company takes care of them during all the steps of acquisition and use of the offered products and services.

Beneficial effects of automation in addressing the issue tickets to the appropriate person or unit in the support team means cuts in issue processing time, reduction of mistakes due to human errors, reduction of company personnel involved in repetitive tasks, freeing precious resources for higher value

works.

This explains why the automatic classification of tickets (where the class to be predicted can correspond to intrinsic hidden properties of the tickets, like the urgency, impact, or to predefined ticket routing and resource allocation schemes) has become a hot field of application for both industrial and academic research in the sectors of *Natural Language Processing (NLP)* and *Machine Learning (ML)*.

Indeed, the usage of ML techniques allows for identifying relevant behavioural patterns which can be exploited to understand best practices, optimize the process and recommend actions with the aim to improve the overall performance of the TMS.

In particular, there is a growing interest in approaches based on *Deep Learning (DL)* for the development of advanced TMSs, due to their ability to learn quite accurate classification models and automatically extract more informative features from low-level raw and/or noisy data, without requiring costly feature engineering steps.

The ever growing business interest in this research area is leading companies to invest in extending their TMSs with intelligent tools, to increase the efficiency and quality of ticket management processes.

As an example, *Uber's Customer Obsession* team define and develops advanced solutions to improve the user experience with Uber services.

Several companies are focusing their business on furnishing advanced solutions for supporting TMS services through the development of software platforms to be customized for the specific requirements of the different customer care uses, like *Zendesk*, *Freshworks*, *Salesforce*.

In more detail, *Zendesk* offers a platform for tracking and organizing tickets through different channels, routing the requests to the right agent based on their availability, workload and expertise for an efficient and personalized solution. The introduction of AI modules, built on billions of real customer service interactions, gives the possibility to better understand customer experience for a more personalized support.

Freshworks is a company offering scalable platforms for supporting the customer care by integrating different channels, with smart business decisions through analytic tools, leveraging data from customer context and behavior in a unified data representation for enabling sales, marketing, and support teams to deliver personalized solutions.

Salesforce offers a cloud-based platform designed to help the connection between the businesses of the companies and their customers. A suite of different products is aimed at integrating sales, services, marketing, commerce, and IT teams furnishing a single, shared view of the customer information.

Moreover, a powerful set of AI-enhanced features was recently added in order to improve the quality of customer services with prediction abilities.

One of the most advanced integrated data and AI platform is offered by *IBM Watson*. Recognizing the profound promise of AI in customer care, the design of virtual agents and chat bots can quickly build natural conversation flows between company apps and users, and deploy scalable and cost effective solutions. The Provider Services Conversational Voice Agent is the IBM AI solution designed to understand the intent of a provider's call, verify if there is the right permission to access the system and member information, and then determine how best to provide the information requested, without the need to speak with a live agent, by using significant speech customization with seven language models and two acoustic models. Advanced NLP systems are employed to analyze text and extract meta-data from content such as concepts, entities, keywords, categories, sentiment, emotion, relations, and semantic roles. Moreover, cognitive search and content analytics engine can be used to support customer applications to identify patterns, trends and actionable insights that drive better decision-making.

2.4 Fake detection

Nowadays, the world of information is very complex, fast, and insidious. Publishing, sharing and accessing information is constantly stimulated by the generalized use of online and social media (e.g., *Twitter*, *Facebook*, *Instagram*, etc.), and huge amounts of news are generated and manipulated every day from the traditional main media, online social systems, and personal broadcasting systems. However, this proliferation of information that circulates so fast through the digital media makes the veracity control and the fact-checking very difficult, making the social media a fertile ground for the spread of unverified and/or false information commonly referred as fake news. People often publish posts or share other people's posts verifying neither the source nor the information validity and reliability. Consequently, malicious users can leverage these unofficial channels to share misleading or false news to manipulate the readers' opinions and make false news viral. Unfortunately, the spreading of false, unverified and not completely true information is considered as one of the most dangerous threats to democracy, justice, journalism, public trust and freedom of expression. As a matter of fact, the fast and widespread propagation of fake news together with the disinformation has become an emergency, considering the suspicion that important society events, like the Brexit referendum and the United States 2016 election, were influenced by malicious misinformation actions. The complexity and the importance of this phenomenon requires to encourage interdisciplinary research, in order to make a joint effort covering all the aspects involved in it. For example, social and psychological factors play a key role in understanding the motivations, the causes and the consequences of this phenomenon resulting in a distortion of the reality. Thus, only a collaborative effort among experts from different fields like computer and information sciences, political science, journalism, social sciences, psychology, and economics will give the possibility to reach the result of curbing this dangerous trend, thus limiting the possible negative effects.

The broad and common definition assimilates fake news to false news, where the term news includes generically articles, claims, statements, speeches, posts, among other types of information related to public figures and organizations, created by journalists and non-journalists. This generic definition is exclusively focused on the information authenticity, making the fact checking the main fundamental aspect to be verified in order to state if a news can be considered true or simply a fake. Obviously, fake news should not be confused with information related to opinions, viewpoints, personal considerations, feelings and all theories and thinking that have a strong com-

ponent of subjectivity. Thus, fake news requires a scientific demonstration of the falsity through the support of objective facts and evidences. A narrow definition of fakes considers the fake news as intentionally false news, emphasizing the deceptive intent for aims that can mislead the opinion and the understanding of the real true facts.

Fake news can take several different forms and shapes in the social media environment creating a variety of terms representing the concept of false news with a multiplicity of different nuances, [6]. Terms like false news, deceptive news, satire news, disinformation, misinformation, rumors and clickbait emphasize differently aspects related to the authenticity, the intention and the form through which the information is represented.

Satire news are written in a satiric style, where techniques, such as exaggeration, humor, and irony make the news content exaggerated and sometimes ridiculous.

Disinformation refers to false information that is spread with the specific intent of misleading or deceiving people, especially adopted in political propaganda. Misinformation is an incorrect or misleading information that is spread regardless of intent to mislead. The main difference between disinformation and misinformation is the intent. Although both words refer to wrong or false information, only disinformation is wrong on purpose, deliberately false. While this distinction may seem simple enough, misinformation and disinformation are similar and sometimes interconnected, and so get used interchangeably.

A rumour is an unverified statement or story put in circulation without being proved true, often related to private aspects of people life.

Clickbait typically refers to the practice of writing sensationalized or misleading headlines and information in order to attract attention and encourage visitors to click on a link to a particular web page often for improving money gain.

For sure, the first question to be asked is why fake news are generated. Motivations are the same as the ones that lead people to lie, but resulting in a high resonance capacity. One of the key main motivations is a pecuniary reason, in fact, fake news often are produced in order to become viral on social media, thus attracting a significant amount of revenue from advertising when users click on the website where the news is being published. This is also the reason of the spread of fake online reviews, in fact, most of the online customers base their purchase decisions on the online consumer reviews. The possibility to artificially change the considerations, the comments and the quality judgements inside fake reviews can unfairly grow the profit of specific companies, brands and products. Political fake news are generated

to attract consensus of voters discrediting the reputation of opposite parties and prominent political figures, in order to take advantage. Sometimes, false news and rumors are deliberately generated with the aim of damaging the reputation of a person or entity. Sometimes, fake news are just generated in order to create confusion, and disorient people.

Regardless of the reasons, with social media platforms, news content can be shared among users with limited or no filtering, fact-checking, or any form of editorial judgment by any authoritative or regulatory third-party, giving the possibility to easily disseminate false information. Fake detection is of fundamental importance in order to discern between true and fake news in order to contrast the dissemination of fakes, thus limiting the negative dangerous effects. This is the reason why digital and social media companies like Facebook and Google are putting efforts to exclude fake news sites from their advertisement platform when contravening policies against misleading content, and to identify and flag false articles with the support of fact checker in order to avoid accidentally propagating.

One of most difficult and challenging task is how to check the authenticity and intention of a given information. Research studies developed across various disciplines such as sociology, economics, psychology, and computer science are reaching important qualitative and quantitative results in the analysis of fake news. These research studies [75], trying to build accurate and explainable models for fake news detection and intervention, are based on two main group of theories, the news-related theories and the user-related theories. News-related theories try to detect and reveal fake news by analyzing only the news, focusing on the contents, the writing style, the quality, the sentiments expressed and the strength used. User-related theories, instead, focus their attention on the analysis of the users involved in fake news activities together with their role in the consciously or unconsciously, intentionally or unintentionally spreading. Profiling the users, their reputation, and analyzing the activities of posting, forwarding, liking, and commenting fake news are the main objectives of these research theories.

The immediate but most expensive way to detect fake news is to delegate this arduous task to domain experts who are able to verify the veracity or falsity by using their knowledge for a demonstrable reasoning deducted by real facts. Given the huge amount and high speed of circulating news, it is unfeasible to relegate the detection and verification of fakes to the only human effort. It is therefore necessary to introduce automatic fake detection methods.

Fake detection is a text classification problem in the Natural Language Processing research field. Recently, machine and Deep Learning methods

are reaching promising results, [28]. However, learning classification models that can reliably detect misleading/false information requires coping with different challenging issues. First, in many social media channels, information is conveyed through short texts, which can be regarded as a form of raw, noisy and sparse data. Moreover, the class distribution is typically unbalanced as the number of legit contents for a specific topic overwhelms the malicious one. Last but not least, since class labels need to be assigned manually performed by domain experts, the amount of labelled data is often scarce [72], so making it unsuitable to adopt a fully supervised learning approach –which is, however, the prevalent one in the field; on the other hand, as observed in [71], the wide diversity of news domains in social media prevents adopting a pure transfer learning solution, based on reusing an existing good fake-news classifier (trained on a corpus containing a sufficient number of labelled data) to detect fake news in a different domain.

2.5 Explanation Techniques

Recent advancements in machine learning techniques have resulted in extensive use of artificial intelligence supporting and influencing most aspects of our lives, often demanding also important and critical decisions to complex machine learning models' predictions with the aim of limiting as far as possible the human intervention and supervision. Nowadays, deep neural networks have a predominant role in this scenario thanks to the high performances, above all in terms of accuracy, that are capable of reaching in several decision tasks. However, such deep models are characterized by having complex architectures implementing non-linear functions that make so often the entire decision process flow so difficult to be interpreted. Thus, deep neural networks are considered black boxes by human operators because of the difficulty of understanding their behaviour. Above all, when deep learning is employed for supporting decision-making in critical application domains, it is of fundamental importance to understand, analyze, interpret and explain the rational reasons for the model behaviour behind its decision in order to trust, validate, check and refine the used model. Moreover, in order to motivate the final user to employ and trust an artificial intelligence model based on machine learning and deep learning, it is of fundamental importance to explain clearly the correlation between each decision result and its corresponding input instance. The explanation is more effective if similar training examples are provided to the user by supporting the decision result. For this reason, in the last few years, a considerable effort has been made in the so-called explainable artificial intelligence (XAI) domain as a special research field devoted to studying in depth the explanation problem.

Summarizing, explanations play an important role in supporting the entire decision process with several purposes:

- guiding users through the decision-making process, thus identifying the main reasons for the decision result;
- furnishing a correlation of cause and effect between the features of an input instance and the result decision, highlighting which features have contributed most to the final result;
- helping the user to understand if it is better to trust or not the prediction and/or the model;
- helping to evaluate the goodness of a model furnishing means to improve and fine-tune the model itself by understanding its failure points;

- extracting new insights and hidden laws of the model;
- identifying modules responsible for incorrect decisions
- giving a human interpretable interface to an artificial and complex system.

Several explanation techniques have been proposed in recent years, [56]. A first major distinction, [69], considers two types of explanation: *ante hoc explanation* and *post hoc explanation*.

In the *ante hoc explanation* techniques, the prediction models incorporate the modules for the explanations into their architecture. The model has the capacity to both predict and explain why it makes that prediction as well, thus furnishing additional information that can provide insights reasoning why such predictions were made. Ante hoc explanation deals with the inner working of models to interpret their results.

Differently, in the entire workflow, including the *post hoc explanation*, there are two separate modules: (1) the prediction module only dedicated to the task of prediction, and (2) the explanation module for its interpretations. The explanation module does not influence the predictions made by the prediction module. Post hoc explanation tends to be more flexible than the ante hoc ones because it can be applied to the models without the necessity to analyze the structure of the classifier that could be considered as a black box, even if it provides less comprehension of the model itself.

The explanation can be classified with respect to the furnished results as visual, textual or example based. Visual explanation is a direct and easy-to-understand way to represent the reasoning of the model behaviour in the classification task through visual representation of the relevant features. Textual reasoning is particularly useful for text classification tasks. While the example-based explanation uses similarity and dissimilarity approaches to explain the decision results of an input instance by putting it in relation with other true labelled examples and counterexamples.

Moreover, the explanation can be done by exploiting: Functional Analysis, Decision Analysis, and Explainability by Design. Functional Analysis tries to capture overall behaviour by investigating the relation between inputs and outputs. On the other hand, Decision Analysis tries to understand internal behaviour by probing internal components. Explainability by Design is a technique that includes explanation into the design of DNNs, thus making the initial effort to create the model architectures and algorithms with explainability in mind, with the advantage of avoiding the expensive and challenging building of the posterior knowledge explanation. Users who

adopt explainable technologies can understand the mechanisms of the systems that they are using and thus make better decisions.

The main properties that characterize the explanation approaches can be summarized as follows:

- *Locality.* In order to be meaningful, an explanation should be locally faithful, i.e. its behaviour should be coherent in the vicinity of the instance being predicted.
- *Globality.* It is the possibility to identify globally faithful explanations that are interpretable for evaluating and trusting the entire model and not only some predictions.
- *Model-agnostic.* Model-agnostic explanations try to interpret the model behaviour by making a correlation between input features and output decisions, considering the model as a black box, without analyzing the functionalities inside the architecture. For this reason, this kind of explanation results in being more flexible and suitable for DNNs. Post hoc explanations are model-agnostic, in fact, they assume to access the model only for querying the classification results on different inputs.
- *Model-specific.* These techniques are only suitable for specific classes of models because they generally focus on an algorithm or inner structure details of the model. Ante hoc explanations are model-specific because the explanation generator and the predictor are the same models with varying degrees of transparency.
- *Interpretability.* The capacity of the explanation to provide a qualitative understanding of the input-output model behaviour for the end users. Thus the explanations should be easy to understand and verify.

More generic properties include reliability, causality, generality, scalability, and accuracy. As regards the accuracy and the explanation, it is worth highlighting that there is a trade-off between these two elements. Simpler predictive algorithms are more easily interpreted by human beings, while they are prone to be less accurate than the complex and advanced methods that are hard to explain. Thus, the explanation, which is an assessment of the model that reflects how easily a model is explained to the users, is often counterbalanced by the model accuracy, which is a quantitative measure of the probability of making correct predictions, usually strictly related to the complexity of the model itself.

LIME. LIME (Local Interpretable Model-Agnostic Explanations) [54] is a well-known model-agnostic explanation technique that explains the predictions of any classifier in an interpretable and faithful manner, focusing on learning the behaviour of the models locally around the predictions. LIME assumes that all complex models are locally linear; hence, it explains predictions by approximating them with an interpretable simple model around several local neighbourhoods. The neighbourhood of a prediction is obtained by permuting original observations and then measuring the similarity between those generated data and the original ones. The permuted new data are classified by the classification model to be explained, then, after fixing a certain number of features k to be used for the explanation, the k describing features with the highest likelihood of the predicted class are selected, and a simple model (e.g. linear model) is fitted with data based on it. The weights of this explanation model are considered a metric of importance in explaining the local behaviour of the complex classifier model.

Formally, the explanation produced by LIME is obtained by minimizing a function L that measures how unfaithful an explanation model g (with $g \in G$, where G is a class of potentially interpretable models, such as linear models, decision trees, etc.) is in approximating the model probability function f in the neighbourhood locality π_x of the input instance x , taking into consideration the measure of complexity (as opposed to interpretability) $\Omega(g)$ of the explanation model g , as in the following:

$$\xi(x) = L(f, g, \pi_x) + \Omega(g) \quad (1)$$

Word cloud. A ‘word cloud’ is a visual representation of the most frequent words in a text. The more commonly the term appears within the text being analysed, the larger the word appears in the image generated. By opportunely pre-processing a text and by setting opportunely the parameters of the word cloud generator tools, it is possible to avoid the inclusion of stop words and common words, and it is possible to group together words with similar meanings and/or having the same lemma. As word cloud tends to focus only on single word frequency, it is unsuitable for context and phrase representations. Word clouds have recently become very popular for a first data visualization analysis of texts, being employed as a simple tool to identify the focus of written material. Largely used in business, education and politics, for example, to visualise the content of political speeches, word clouds furnish an immediate analysis of the content of a written and/or oral text highlighting whether sufficient attention is being given to the words

mainly representing the target concepts that need to be disclosed.

Fig. [12](#) shows the word cloud of this PhD thesis.

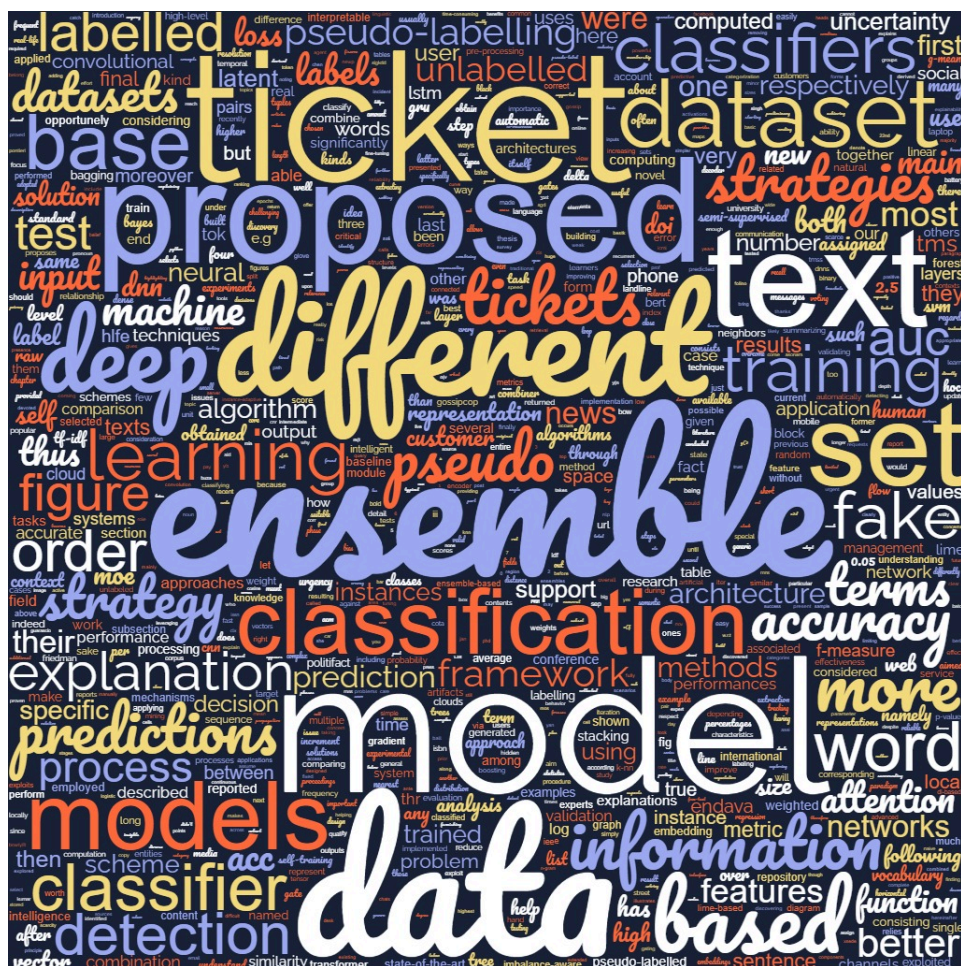


Figure 12: The word cloud of this PhD thesis.

3 Related Works

The research proposed in this thesis has focused on two specific fields of application of the text classification: ticket classification in *Customer Relationship Management* (*CRM*) systems and fake detection. A survey of the state of the art present in literature is here reported.

3.1 Text classification in Ticket Management Systems

The automatic ticket classification and assignment for customer support is a specific field of application of Natural Language Processing and text classification. In literature, many solutions have been proposed in order to improve the capacity of customer support systems in solving ticket issues both in terms of accuracy and efficiency. This section is meant to discuss some major ML approaches to the automated classification of tickets.

Some major approaches developed in the field of Machine Learning (ML) are discussed in what follows, in two separate subsections. More specifically, the first subsection is devoted to illustrating approaches based on traditional (shallow) ML models and training algorithms, whereas the latter subsection focuses on the usage of Deep Neural Networks.

Table I summarizes the main characteristics of both groups of approaches. Before illustrating these approaches in more detail, it is important to pinpoint that only a few existing approaches in the field took advantage of an ensemble learning strategy, and none of them also tried to exploit DL methods to learn the base classifiers and/or an instance-adaptive combiner like ours—all previous ensemble-based approaches to ticket classification employed standard global combination schemes (e.g., voting, averaging). On the other hand, to the best of our knowledge, none of the DL-based approaches proposed in the field tried to complement the returned class predictions with explanatory artifacts, despite the black box nature of deep classifiers.

3.1.1 ML models for TMS

A classification-based help desk system was proposed in [26] to improve the handling of tickets in German Jordanian University, through the provision of several advanced kinds of functionalities (e.g., sending automatic email notifications amongst collaborators for further action, defining business processes for ticket management, monitoring associated performance indicators). Specifically, the system allows for discovering and applying

Table 1: ML and DL-based approaches to ticket classification and intelligent ticket management.

Approach	ML/ DL	Ensemble	Model(s)	Evaluation metrics	Data repre- sentation
[26]	ML	N	SVM	Accuracy, Avg. Ticket Assignment Time, Avg. Ticket Resolution Time	TF-IDF
[49]	ML	Y	SVM, LogR, SGD, NB, DT, RF	Accuracy, Precision, Recall, F1-score	TF-IDF
[2]	ML	N	DT, SVM, NB, KNN	Accuracy	Boolean, TF, TF-IDF
[59]	ML	N	SVM, KNN	Accuracy	TF-IDF
[46]	ML	Y	DT, MNB, SVM, RF, LogR, KNN	Accuracy, Precision, Recall, F1-score	TF-IDF
[53]	ML	N	kNN and variants, DT, NB, LogR, SVM, Quick-SUCCESS	Accuracy, Precision, Recall, F1-score	TF-IDF, Linguistic Features
[60]	ML, DL	N	MNB, SNN	Accuracy	TF-IDF
[45], COTA v1	ML	N	RF, Co-sine Similarity ranking	Accuracy, Hits@3, F1-score	TF-IDF
[45], COTA v2	DL	N	Encoder-Combiner-Decoder (CNN, RNN)	Accuracy, Hits@3, F1-score	Word Embedding
[73]	DL	N	CNN	F1-Score, Precision, Recall, Accuracy	Word Embedding
[74]	DL	N	CNN	F1-Score, Hamming Loss, H-Loss HMC-Loss	Word Embedding

SVM models for classifying tickets based on their title, description and comments. A standard TF-IDF [38] representation (where TF and IDF stand for *Term Frequency* and *Inverse Document Frequency*, respectively) is used to model relevant terms from the tickets, and to turn the latter into vector space tuples suitable for training SVM models. Experimentation has proved the effectiveness of the model in terms of classification accuracy and of many application performance indicators, such as number of tickets per agent, percentage of closed tickets, average resolution time and average assignment time. Tests were executed on a dataset collected from Istanbul Technical University ITU Issue Tracking System, a web application for answering on various requests to different departments within the university. The classification of a ticket consists of two phases (aimed at identifying the ticket category and sub-category, respectively) and is semi-automatic: if the prediction confidence does not overcome a predefined threshold, the ticket class is assigned manually. Classic ML methods (namely, Logistic Regression, SVMs, Naive Bayes, SGD and Random Forests) were also exploited in [49], in order to address three different tasks concerning the handling of ticket messages: spam detection, ticket assignment and sentiment analysis. The discovered classifiers are combined into an ensemble model via majority voting, which is shown to improve the performances of the classifiers.

In [2], it is experimentally demonstrated how the adoption of ML algorithms reduce manual efforts and human errors while ensuring high service levels, and improve end-user satisfaction in Issue Tracking Systems (ITS), i.e., a software system used to capture and keep track of customer issues, and to automatically assign the issue tickets to the relevant person or unit in the support team. Four different supervised ML techniques were tested for performance comparison: Decision Trees (DTs), Support Vector Machines (SVMs), naïve Bayes and k Nearest Neighbor (k -NN) classifiers. Tests were executed on a dataset collected from Istanbul Technical University ITU Issue Tracking System, a web application for answering various requests to different departments within the university.

In order to reduce the time wasted on the incident ticket route and the number of errors in the incident ticket classification, a novel module for automatically classifying incident tickets was proposed in [59], as part of an existing Information Technology Service Management (ITSM) service. The module exploits SVM classification techniques to automatically assign (with an estimated accuracy of 89%) each incident ticket to one of the 10 following categories: application, collaboration, enterprise resource planning (ERP), hosting services, network, security and access, output management, software, workplace and support.

[60] presents an approach to the automatic classification of user tickets in the XSEDE (*Extreme Science and Engineering Discovery Environment*) project, an NSF TeraGrid project providing researchers and students across the country with an integrated network of high-performance computers, data resources and tools, and research facilities. The ever-increasing number of tickets pushed towards the automation of XSEDE user ticket classification. To this end, the authors of [60] resorted to techniques for the discovery of MNB and Softmax Regression Neural Network (SNN) classifiers.

In [46], the application of three popular ensemble learning strategies, namely Bagging, Boosting (Adaboost) and Voting ensembles, is explored in order to combine different ML classifiers. Specifically, the former two strategies were applied to DT, Multinomial Naïve Bayes (MNB), SVM and Random Forest base classifiers. A combination of logistic regression (LR), SVM, MNB and k -NN are used as input to the Voting ensemble classifier. Considerable improvements in the accuracy of ticket classifications are shown for the resulting ensemble classifiers, compared to the underlying base ones.

In [53], a study, aimed to investigate the core design elements of a typical IT ticket text classification pipeline, is presented. Experiments were conducted on two datasets, containing ticket texts supplied by the IT Infrastructure Library (ITIL) Change Management (CHM) ticket processing department of a big enterprise. The aim of these experiments is the comparison of the TF-IDF and linguistic features-based text representations of the tickets together with the comparison of many ML-based classifiers: i.e., Naïve Bayes (NB), some variants of k Nearest Neighbor (k -NN), SVM, and a semi-supervised technique called (QuickSUCCESS), an improved version of the Semi-supervised Classification of Time Series (SUCCESS), proposed in [9]. Their experiments evidence the benefit of using simple (easy to interpret) classification models like decision trees and Naïve Bayes, especially if combined with domain/expert-driven linguistic representations. Similar research results are presented in [52], which also explores link between the explainability of ticket classification and the paradigm of Granular Computing.

3.1.2 DNN models for TMS

Recently, *Deep Learning* (DL) techniques were shown a powerful and convenient solution for extracting accurate ticket classifiers from raw labelled data, without needing heavy manual feature engineering. Basically, a *Deep Neural Network* (DNN) can be regarded as a model capable of learning

a hierarchy of (mainly, non-linear) transformation layers, where each layer yields a more abstract representation of the data than the previous ones. For example, DL techniques are used in the intelligent system *COTA*, integrated into Uber’s TMS and proposed in [45], which supports two main tasks for handling each new ticket: the identification of the ticket type and the recommendation of a suitable resolution and of reply template. In fact, the customer service provides a list of reply templates from which it is possible to select the proper solution to the specific identified ticket type. Two different model implementations are proposed, COTA v1 is a model based on feature engineering, while COTA v2 uses deep learning algorithms in an Encoder-Combiner-Decoder architecture. The system implements DL methods based on an Encoder-Combiner-Decoder DNN architecture, which are shown to improve significantly shallow ML methods, in terms of accuracy and key business metrics (including ticket resolution speed and customer satisfaction level). Another DL approach is proposed in [73], for automatically classifying phone calls to the CRM front-end of a car dealer company, which usually concerns questions on car sales, services, vendors or job opportunities. Specifically, a convolutional DNN is trained to classify the customer calls, suitably transcribed into text documents, into four intent categories (namely, *sales*, *service*, *vendor* and *job*). Experiments executed on the convolutional neural network architecture with 128 filters for different convolutional filter sizes demonstrated the goodness of the approach in terms of accuracy, F1-score, precision and recall in this specific application context.

In [74] the authors proposed a ranking framework called STAR (System for Ticket Analysis and Resolution) for automatic ticket resolution, ticket classification and clustering of incidents in IT service management. Their framework, based on a CNN model, is evaluated against a large real world dataset, and shown able to obtain accurate classifications while capturing semantic relations.

3.2 Fake Detection

The fast and widespread propagation of fake news is becoming an emergency for the correct disclosure of information. Information accessibility has grown exponentially due to the generalized use of online and social media, and huge amounts of news are generated and manipulated every day from the traditional main media, online social systems, and personal broadcasting systems. Therefore, distinguishing the truth and veracity of the news represents a critical task in order to mitigate the diffusion of misinformation and consequent negative effects on society. The main issue is represented by the fact that fakes can be very similar to real news. At first glance, fake news can easily be mistaken for real ones; thus, fake detection represents a relevant and challenging problem. The standard solution that consists in delegating the verification process to trusted professionals and specialists to check claims against evidence based on previously demonstrated or proofed evident facts is time-consuming and expensive and unfeasible for the huge quantity of news shared on the web. The automatic fake news detection of text contents is a critical and challenging Natural Language Processing (NLP) problem.

In the literature, three main types of fake news detection have been proposed: *(i)* knowledge-based, *(ii)* content-based and *(iii)* context-based.

Fake news detection based on knowledge is named *fact checking*, as it adopts the approach of checking the authenticity of news by comparing the information with documents or web resources extracted from the semantic web, linked open data and/or information retrieval. Content-based detection techniques analyse content and writing style to identify fake news and are based on Machine and Deep Learning methods. Finally, context-based detection approaches combine the news content with other information, e.g., the source, the author, the website, the topic, the propagation path and the speed of dissemination.

Content-based approaches to fake news detection constitute the prevalent kind of solutions in the field, due to their broader applicability. Indeed, it is not easy to obtain high-quality integrated information from heterogeneous sources. Even though a large part of the content-based methods proposed so far rely on traditional supervised learning methods, it is important to remark that obtaining appropriate fake-news detection models via supervised learning entails gathering large amounts of reliable (labelled) data, which is time-consuming, expensive and requires specific topic knowledge. Thus, providing fake news detection systems with the ability to also exploit unlabelled data, via semi-supervised learning mechanisms, is necessary to

suitably deal with real-life application scenarios where only small fractions of news documents are provided with a fake/normal class label.

In what follows, some major semi-supervised approaches to the discovery of content-based classification models for fake news detection are surveyed.

Semi-supervised Approaches for Fake News Detection In [55], the authors compare four methods for detecting deceptive and fake opinion reviews: co-training, expectation maximisation, label propagation and spreading, and positive unlabelled learning. Co-training is a technique that exploits different views of the dataset, where each view is a distribution of features representing the data; the basic idea is to train two classifiers on each view and then classify instances on the unlabelled category to enlarge the training set. Expectation maximization consists of two steps: the learning of the algorithm with the conjunction of the labelled and predicted labelled sets (Expectation step) and the prediction of the labels of the unlabelled set (Maximization step). Label propagation and spreading use graph-based algorithms for learning: the graph is constructed by ordering suitable vector features based on a suitable similarity metric, such as Manhattan distance or Euclidean distance, on both labelled and unlabelled nodes; label information is spread across the graph dynamically until all nodes are labelled. Positive unlabelled learning refers to a specific binary classification problem characterised by the constraint that only positive labelled data are available together with unlabelled data, and the classifier has to identify hidden positives from the set of unlabelled examples when negative training data is not supplied or available.

The work in [24] proposes a semi-supervised fake detection classifier consisting of three phases: building the tensor-based embeddings representation of the article text; constructing a k-NN graph of proximal embeddings; and propagating the beliefs by using the *FaBP* (Fast Belief Propagation) algorithm. A similar approach, based on a graph-based semi-supervised fake news detection algorithm is proposed in [5], exploiting document embedding, graph inference for the representation of articles, and a graph neural network-based classifier.

In [41], a semi-supervised temporal ensemble model is learned by using a Convolutional Neural Network (CNN) as a reference architecture for training the base models against the headline and the body of the news. The underlying idea of the temporal ensembling technique [33] is that different prediction outputs of all previous epochs can be aggregated in order to furnish a collaborative prediction which proved to be more accurate and thus

better suitable for inferring pseudo labels. Indeed, the ensemble predictions of unknown labels accumulated in several training epochs perform better than the last epoch prediction.

In [42] the same authors have also proposed a semi-supervised fake news detection technique based on GCN (Graph Convolutional Networks) trained with limited amounts of labelled data. The proposed solution consists of three stages: extracting an embedded representation from the news text by using *GloVe*, constructing a similarity graph using *Word Mover’s Distance* (*WMD*), and finally leveraging a Graph Convolution Network to address the binary classification task in a semi-supervised paradigm.

In [16], the authors introduce a novel deep two-path semi-supervised learning (DTSL) model composed of three convolutional subnets. The first is trained by using a supervised learning scheme, while the second is trained against unlabelled data in an unsupervised fashion. An additional shared CNN is used to propagate low-level features to the two former networks. The loss function is computed by weighting two components: a standard cross-entropy loss function to evaluate the loss for labelled inputs only and the mean squared error of the two output path predictions in order to penalize different predictions for the same training input.

To the best of our knowledge, our current work has been the first attempt to combine (language-model) pre-training and (pseudo-label based) self-training together with an ensemble combining strategy in order to train a powerful (BERT-based) form of deep model to discriminate fake news from genuine ones. As a matter of fact, the idea of resorting to an “old-fashion” pseudo-labeling approach was inspired by the results of the empirical analysis in [10], where it was shown that such an approach can be competitive with state-of-the-art semi-supervised DL methods (leveraging consistency regularization mechanisms), while being more resilient to out-of-distribution samples in the unlabeled set.

4 A Deep Ensemble Framework for Text Classification

In this section, a comprehensive classification framework for the entire processing flow of text classification is proposed. It relies on training a novel kind of ensemble of deep classifiers. Moreover, it proposes different kinds of easy-to-interpret artefacts for explaining and debugging the predictions of the ensemble providing AI-based interpretation methods. Finally, it introduces a novel self-training learning process based on an ensemble pseudo-labelling workflow aimed at overcoming the performance limits of scarce labelled datasets.

The entire framework is first proposed in its general form, while two implemented architectures are then described in more detail as specialized frameworks for the two text classification fields of application treated as use cases: the text classification for Ticket Management Systems and for fake detection.

4.1 The Software Architecture of the Framework

The conceptual architecture of the proposed complete framework aimed at furnishing a complete text classification tool is shown here in detail. The framework relies on training a novel kind of ensemble of deep classifiers, proposing different kinds of easy-to-interpret artefacts for explaining and debugging the predictions, and supporting a pseudo-labelling-based self-training learning process for improving classification performances in case of scarce labelled data.

Fig. 13 illustrates a high-level view of the proposed framework. The *Information Retrieval* block collects raw data from different kinds of sources by listening to different communication channels. Moreover, depending on the specific application, this block transforms the data into a unified semi-structured format (for the TMS ticket data, fields like customer ID, timestamp, urgency, the free-text body of the customer request, etc. are extracted, while for fake detection, the title and the document text are retrieved from the web pages) and, if it is required, it makes the data anonymous for the sake of privacy preservation. These consolidated data are maintained in an ad-hoc repository (named *Raw Data*). With the help of the *Text Encoding* block, texts are converted into a tensor form, suitable for training/undergoing neural-network models, after applying common text processing operations (including lower-case normalization, tokenization, stop-word removal, stemming) to their free-text contents. It is worth not-

ing that all these operations for data preparation represent a crucial step for improving the performances of text classification tasks as deeply investigated in [64], especially when the text is in the form of chat messages like in the TMS ticket use case where spelling and grammatical rules are often neglected. Pre-processed text data are used, by means of the *Deep Ensemble Model Building* block, to train an ensemble-based text classification model. Novel heterogeneous ensemble schemes based on MOE and Stacking architectures consisting of multiple diverse DNNs are proposed in the *Model Building* block implemented for the ticket application, while a self-training ensemble-based pseudo-labelling scheme is proposed for the fake detection application when only a few labels are available as a training set.

The *Pseudo Labelling* block supports a novel ensemble-based self-training approach to be used when the available labelled data are scarce in order to integrate them with unlabelled data furnished with labels artificially generated from the predictions of the classifier models.

The *Explanation Building* block supports the classification system with multiple functions like model correction and refining; it furnishes a comprehension reasoning of the results with respect to the input instance, it gives a valid instrument to the application end users for validating, checking and comparing the results.

The details of the *Ensemble*, *Pseudo Labelling* and *Explanation* blocks are reported in the following sections. Finally, the details of the two architectural implementations of the proposed framework specialized for the TMS and the fake detection applications are also reported.

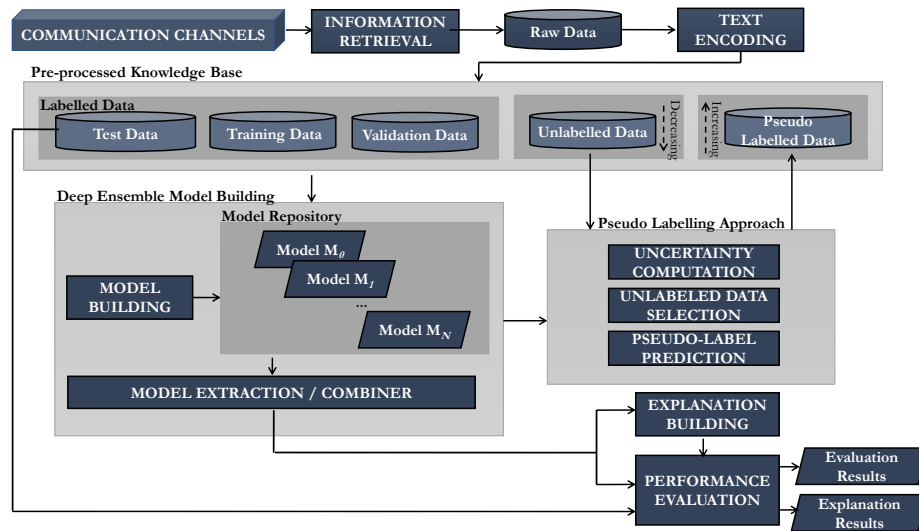


Figure 13: The proposed Framework: Conceptual Architecture.

4.2 The Ensemble Strategies

The ensemble techniques are exploited in order to reach better generalization performances and to reduce the bias and the variances among the base learners. The *Deep Ensemble Model Building* block of the proposed framework is built with multiple heterogeneous classifiers appropriately combined in order to have improved characteristics with respect to the single base learners.

In the proposed framework architecture, an ensemble text classifier is modelled as a neural network that: (i) takes a vector x encoding the text contents of an input sample; (ii) incorporates several different classifiers, each of which instantiates a specific DNN architecture; (iii) includes a component named *High Level Feature Extractor (HLFE)*, extracting a dense representation of x as a whole, which is meant to capture both semantic aspects of the text management domain and their relationships to the classes; (iv) a *combiner* sub-net that is meant to combine the outputs of the base classifiers into an overall class prediction for x , taking as an additional input the high-level representation of x coming out of the HLFE component. Defining the strategy to combine the base models into a collective decision is a critical issue in each ensemble classification model.

As shown in Figure 14, the HLFE simply consists of three layers: (i) a Word Embedding layer, mapping each term of x to a dense vector; (ii) a Global Max Pooling layer [36], which summarizes the contents of the ticket by aggregating the dense vectors of all the terms appearing in x ; (iii) a fully-connected dense layer equipped with *ReLU* activation functions. This simple structure proposed for the HLFE sub-net is meant to facilitate classification explanation and model debugging. Indeed, besides enabling the discovery of instance-adaptive combination schemes, the HLFE allows human operators and analysts to assess whether the high-level representations the ensemble model relies upon are really meaningful and general enough, or they rather focus on incidental aspects of the input text (so undermining the trustworthiness of the model).

Two different ensemble architectures, denoted hereinafter as *Stacking ensemble* and *MOE ensemble*, are proposed in the framework architecture, which implements two alternative strategies for combining the predictions of the base classifiers.

In the *Stacking ensemble* architecture, shown in Fig. 14, a feed-forward sub-net (consisting of one layer with softmax activations in the current implementation) is trained to derive overall class predictions from the predictions (namely, $\tilde{y}^{(1)}, \dots, \tilde{y}^{(n)}$) of the n base learners and the output of the

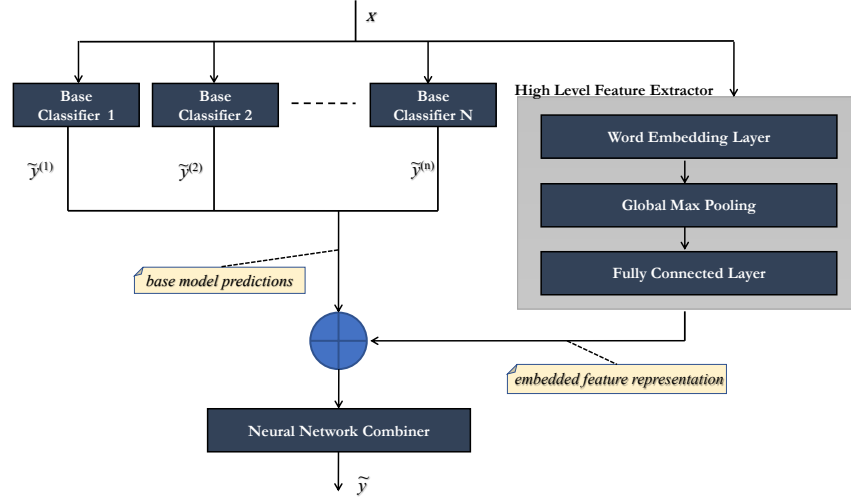


Figure 14: *Stacking ensemble* architecture.

HLFE sub-net.

The *MOE ensemble* architecture (shown in Fig. 15) implements a Mixture-of-Experts (MOE) [39], where the overall prediction for x is obtained as a linear combination of the predictions (namely, $\tilde{y}^{(1)}, \dots, \tilde{y}^{(n)}$) of the n base classifiers. To this end, a *Gate* sub-net (consisting of a single dense layer with softmax activations) is learnt for deriving the weights of this combination out of the latent representation of x returned by the HLFE sub-net (rather than directly from x itself, as in classical MOE schemes). This architecture aims at giving more importance to the decisions of the base models that look more competent in the region of the latent space to which x belongs.

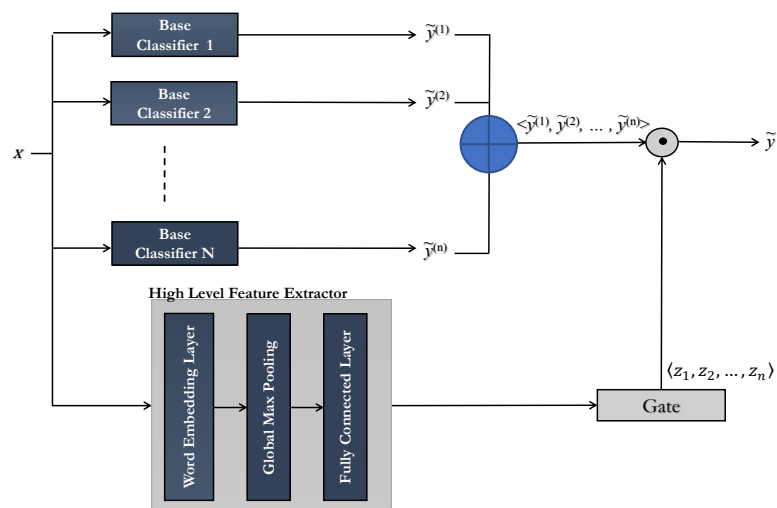


Figure 15: *MOE ensemble* architecture.

4.3 The Explanation Technique

The explanation consists in providing a visual informative representation able to explain the behaviour (i.e., the predictions) of the classification model. As the proposed framework is devoted to classifying data with an ensemble of deep neural networks which by their nature are assimilated to black boxes, it is of primary importance to equip the framework with an explanation tool able to give some kind of correlation information between the classification decision and the input data.

In order to allow human operators to easily evaluate and debug the classification decision obtained for an input instance x , with some deep ensemble-based classifier M , the proposed framework allows for computing two kinds of explanation artefacts: (1) a local linear classifier computed with method LIME, capturing which words impacted the most (and how much) on the classification decision of M for x ; (2) multiple word clouds summarizing the contents of different groups of examples texts that fall in a neighbourhood of x , in the embedding space associated with M . The user can also inspect each of these groups of neighbours.

For the sake of interpretability, both artefacts (1) and (2) are derived from the *Bag-Of-Word* (BOW) representation of the instances that result from applying the text processing operations (namely, tokenization, stop-world removal and stemming). Precisely, let V be the reference vocabulary, and D be a given set of examples (viewed each as a plain-text document and associated with a ground-truth class label), the BOW representation of any text instance y , denoted as $bow(y)$, is a vector containing as many components as the terms in V , such that: for each term $t \in V$, the respective component $bow(y)[t]$ of the vector stores how many times t occurs in y .

In the remainder of this section, technical details on the two kinds of explanation mechanisms and artefacts produced in the proposed approach will be provided: LIME-based post-hoc explanations and latent neighbours and associated word clouds. Then the workflow that the user is expected to follow for interactive requesting and exploring these kinds of explanation artefacts will be treated in more detail.

4.3.1 LIME-based Classification Explanations

The proposed framework integrates a public Python implementation of the post-hoc explanation method *LIME* (Locally Interpretable Model-agnostic Explanations) proposed in [54]. For the sake of interpretability, in this implementation, the contents of every text y is represented as a binary

vector where the component associated with each term t is 1 iff t appears in y (i.e., if $\text{bow}(y)[t] > 0$).

Essentially, in order to approximate the behaviour of M around the instance x , for which we want an explanation of the classification yield by M , a (sparse) linear classification model is trained via Ridge regression on a number (namely, 5000) of artificial instances, computed by perturbing x ; each of these instances is labelled with the class that M predicts for it, and it is weighted according to its proximity to x (measured on the basis of cosine similarity), in order to make the model pay more attention to the instances closer to x (the closer the instance, the higher the misclassification error/loss). Specifically, for each prediction class, a one-vs-all ridge classifier is computed over a certain number n of features (i.e., binarized versions of the instance terms) most correlated to the class, with n opportunely selected depending on how many features are considered useful for most representing the entire instance.

The resulting set of linear classifiers allows the user to understand how the selected terms impacted on the decision that M took for x . These classifiers are presented in the form of visual artefacts through a bar diagram, where each vertical bar corresponds to one of the prediction classes and shows the coefficients of the top n terms considered for the class. For example, if in the linear sub-model of a certain class cl , a term t is associated with a weight w , it means that when this term is removed from the instance, the membership probability of class cl would decrease by w .

4.3.2 The Neighbor-based Word Clouds in the Latent-Space

The LIME explanations described before help study the behaviour of a classifier M locally to a test instance x , but do not allow for assessing whether the text embedding function learnt by M (and used by the latter for combining the predictions of the base models) is really meaningful. Indeed, looking at concrete examples of neighbours of x in this latent space can act as an additional complementary means for evaluating the quality and reliability of M in the portion of the instance space surrounding x .

The ensemble models proposed for classification incorporate an embedding subnet HLFE that maps the text in a low-dimensional space, where instances featuring similar contents and class-related patterns should appear close to one another. It is worth noting that, when providing the proposed ensemble models with a text instance y , the output of the HLFE subnet is a non-linear transformation of a max-based aggregate representation of the embeddings of the terms in y .

This simple embedding-combination scheme mainly serves the purpose of complementing predictions with easy-to-interpret explanations. Indeed, the resulting latent-space text representations exhibit two interesting properties: (i) they capture the subset of features of an instance that impacts the most on classification, and (ii) they are correlated to the flat BOW representation upon which our explanation artefacts (i.e., the local models obtained with LIME and the word clouds illustrated below) are based.

For any text instance y , let $f_M(y)$ denote the extended representation of y that results from concatenating this latent-space representation of y (coming out from the HLFE sub-net of model M) and the predictions $\tilde{y}^{(1)}, \dots, \tilde{y}^{(n)}$ that are returned for y by the n base classifiers incorporated in M —each of these predictions is a discrete probability distribution over the classes.

For each ensemble classifier M , a metric-tree index \mathcal{I}_M is built on a given set D of labelled text instance by adopting the following cosine-based function $d_M(x, y)$ (introduced in [17]) to compute the distance between two instances x and y (in the extended latent space associated with M):

$$d_M(x, y) = \begin{cases} c2d\left(\frac{f_M(x) \cdot f_M(y)}{\|f_M(x)\| \times \|f_M(y)\|}\right), & \text{if } \|f_M(x)\| \times \|f_M(y)\| \neq 0 \\ c2d(-1) = 0, & \text{otherwise} \end{cases} \quad (2)$$

where $c2d(\phi) \equiv \sqrt{(1 - \phi)/2}$, for any $\phi \in \mathbb{R}$. The elements in the index \mathcal{I}_M point to the contents of another data structure, storing, for each example y , the text of y , its BOW representation $bow(y)$ and its class label.

After receiving a prediction by M for a novel instance x , the user can pose a k -NN query against \mathcal{I}_M , by simply specifying the number k of neighbors that must be retrieved for x . For notation convenience, let us denote by $\mathcal{N}_M^k(x)$ the result of this query, i.e. the set containing the k nearest neighbors of x in the HLFE-transformed version of dataset D , along with their associated distances from x —for the sake of presentation, let us just denote this set as a set of text instances rather than a set of pairs of the form $(y, d_M(x, y))$ with y denoting a text identifier. Moreover, for each class c , let $\mathcal{N}_M^k(x)|_c$ be the k nearest neighbors assigned to c (i.e., $\mathcal{N}_M^k(x)|_c = \{y \in \mathcal{N}_M^k(x) \mid \text{label}(y) = c\}$, where $\text{label}(y)$ is meant to denote the class label associated with any instance y).

Neighbor-based word clouds To provide the user with a compact view of these neighbourhoods of x , a word cloud is produced for each set S chosen among $\mathcal{N}_M^k(x)$ and all class-wise subsets $\mathcal{N}_M^k(x)|_c$; the word cloud depicts each of the q most relevant terms in S with a size that is proportional to the

relevance of the term –in our framework prototype system, q is set to 100 by default. The relevance of each term t for a neighbor set S , is computed as a weighted average of the TF-IDF scores of t over the neighbors in S , where each neighbour is assigned a weight that depends on how close it is to x in the extended latent space (the lower the distance, the higher the weight). Precisely, the relevance score $rs(t)$ of t is computed as follows:

$$rs(t) = \sum_{y \in S} bow(y)[t] \times idf(t, D) \times (1 - d_M(x, y))^\beta \quad (3)$$

where $\beta \in \mathbb{R}$ is a scaling factor that allows for controlling the degree of sensitivity of the relevance score to inter-text distance, while $idf(t, D)$ is a variant of the *inverse document frequency* of t in D defined as $idf(t, D) = \log(|D| + 1) - \log(|\{d \in D \mid bow(d)[t] > 0\}|) + 1$.

Two examples of such word clouds are reported in the experimental section for a specific case of real-life ticket message

4.4 The Pseudo Labelling Strategy

A pseudo-labelling self-training ensemble-based module is proposed here in order to improve classification performances when very few labelled data characterize the available dataset. The scarce labelled data are split into three subsets: the initial training set, the validation set and the test set. In order to increase the training set, unlabelled data are labelled by the model itself by opportunely exploiting the classification prediction and the uncertainty of the prediction.

Figure 16 shows the detailed pseudo-labelling self-training process. Initially, the classifier model is trained by using only the labelled data, i.e. the *Initial Training Set* and the *Validation Set*. After training, the model is used to predict the unlabelled data and compute the uncertainty of prediction in order to select a subset of the unlabelled data most suitable to be pseudo-labelled and included in the training set. The process is iteratively repeated until no more unlabelled data are available or when the selection strategy terminates or the prediction uncertainty does not satisfy the minimum criterion to add new pseudo labels.

The self-training process generates a trained model at each iteration that is saved in a model repository together with its performances with respect to the validation set.

Figure 17 shows the ensemble of pseudo-labelling self-trained models. In fact, the generated final classifier model is obtained by opportunely combining the several models trained during the pseudo labelling self-training process steps by the *Ensemble Combiner*. The latter implements different ensemble strategies to combine in different ways the output of the base learners.

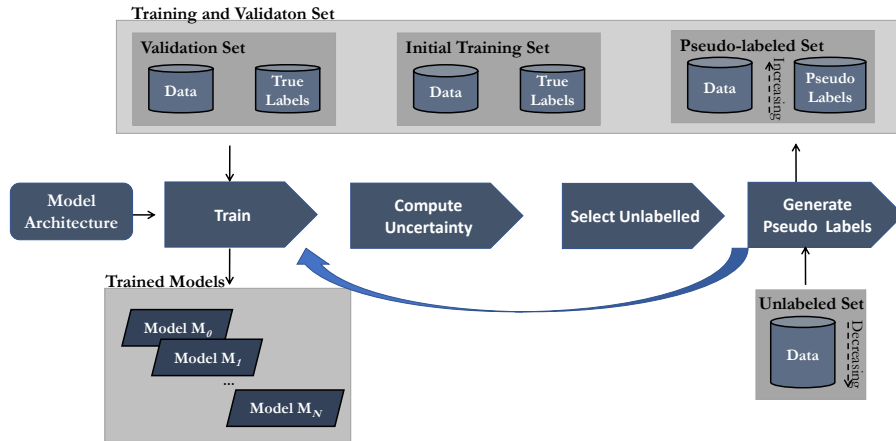


Figure 16: The pseudo labelling self-training process.

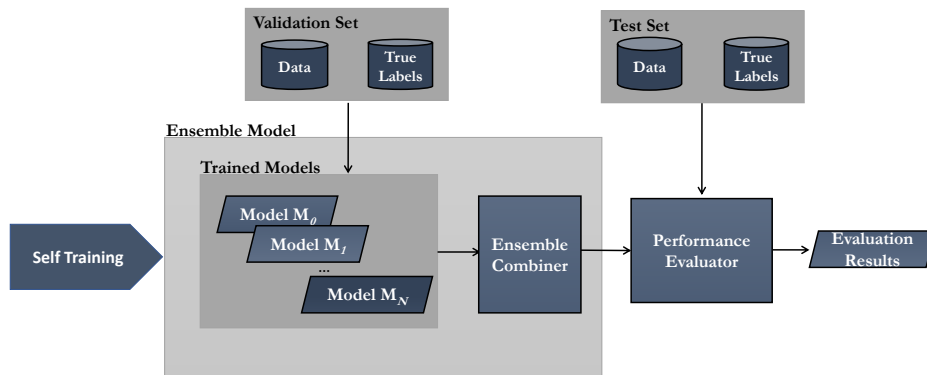


Figure 17: The ensemble of pseudo labelling self-trained models.

5 Case studies: Ticket Classification and Fake Detection

An overview of the implemented architectures of the proposed framework opportunely specialized for the two real scenarios are reported below: the intelligent TMS (Ticket Management System) text classifier and the fake detection system.

5.1 The Ensemble-based System for Ticket Classification

The TMS classification system is an implementation of the proposed framework specialized for the categorization of tickets in customer support systems, essentially relying on combining powerful (deep) ensemble classifiers with ad hoc explanation mechanisms.

The following main concepts will be illustrated in the next paragraphs:

- the high-level conceptual architecture of the TMS framework;
- the core intelligent deep ensemble ticket classification scheme enabled by the *Model Building* block of the TMS framework;
- the human-in-the-loop scheme enabled by the explanation module implemented in the proposed TMS framework architecture in order to support the discovery, application, validation and improvement of reliable ticket classifiers;
- the explanation-based scheme for analyzing the wrong predictions made by the ticket classification models for helping to detect and analyse the weakness of the models themselves.

5.1.1 The Software architecture

The high-level view of the proposed framework specialized for TMS, which supports a layer-wise processing flow of raw ticket data, is described here in detail. Fig. 18 illustrates the overall conceptual architecture. The *Information Retrieval* block is devoted to: (i) collecting such data from different kinds of sources, including e-mail/chat messages, web forms, and TMS applications in the *Ticket Gathering* block; (ii) transforming the data into a unified semi-structured format with fields like customer ID, timestamp, urgency, free-text body of the customer request, etc. in the *Ticket Data Wrapping*; (iii) making the data anonymous in order to make the analysis

and investigation process immune from the privacy problems in the *Ticket Anonymization* block.

The collected, uniformed and anonymized data are stored in the *Raw Data* repository for being pre-processed in the *Ticket Text Pre-processing* block and encoded in a tensor form in the *Tensor Generator* block inside the *Ticket Encoder*.

Pre-processed tickets fill the D dataset used for training, validating and testing the entire classification system.

Inside the *Model Building* block, the *Deep Classifier Discovery* block is devoted to investigating and exploring the potentialities of the proposed ensemble learning models, while the *Latent Space Base Explanation Index Building* block creates the internal structure for supporting the explanation of the classification results from a latent space representation of the ticket data. For this purpose, the proposed ensemble models include a *High-Level Feature Extractor* block mapping the (pre-processed) tickets onto a low-dimensional “latent” space, useful to capture class-related relevant aspects of the tickets. Finally, the *Application and Validation Model* block furnishes the end user with a complete instrument for classifying new tickets together with an explanation module fundamental for validating and improving the performance of the classification system through the aid of human feedback.

5.1.2 Exploiting Novel Efficient Deep Ensemble Classification Models for Ticket Classification

The proposed approach to the problem of predicting the class of a new ticket relies on discovering a special kind of classification model (or *ticket classifier*), which takes the form of an ensemble of deep *base classifiers* featuring different DNN architectures. As confirmed by the empirical study reported in the experimental results, this ensemble model is expected to improve the performance of all the base classifiers (thanks to its higher expressiveness and robustness to overfitting and class-imbalance risks). More specifically, a multi-representation ensemble approach exploiting weak models based on different architectures is devised here.

In order to promote the diversity of the single prediction, in the proposed solution a set of Neural Networks, DNN_1, \dots, DNN_n , based on different deep architectures, are used to yield different weak predictions, which are the input for subsequent combination strategy. In the current implementation of the framework, four different DNN architectures are used to build the base models in the ensemble: (i) *LSTM*, consisting of a stack of Long Short-Term Memory layers; (ii) *CNN*, a Convolutional Neural Network ar-

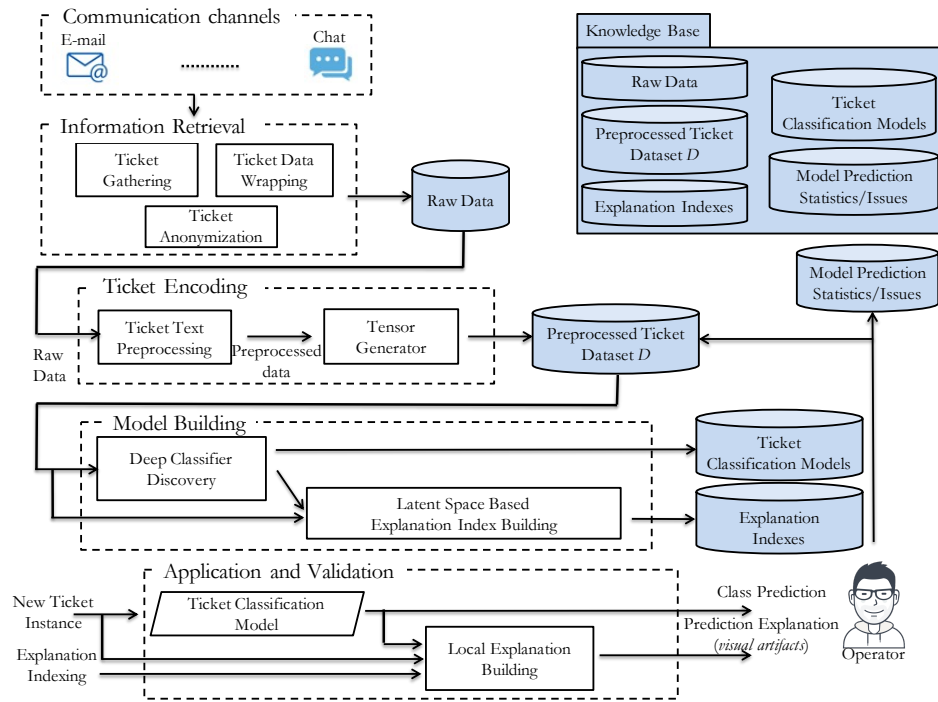


Figure 18: Intelligent TMS Framework: Conceptual Architecture.

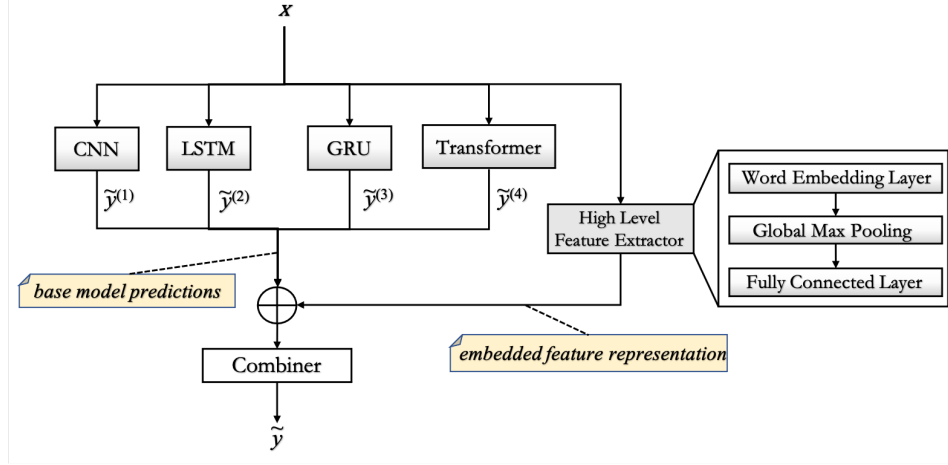


Figure 19: *Stacking ensemble* architecture.

architecture (including Global Max Pooling for down-sampling the convolution results), (iii) *GRU*, composed of a stack of Gated Recurrent Unit layers; and (iv) *Transformer*, corresponding to the encoder part of standard Transformer architecture [66]. More details on the configuration of these architectures are given in the experiment chapter.

All these architectures take, as input, a vectorial encoding x of the sequence of terms appearing in a ticket (also denoted as x in the following, with some abuse of notation), and include a trainable word embedding layer for mapping each of these terms to a dense low-dimensional representation of the term –before applying the transformations encoded by the above-mentioned DNNs. Moreover, each architecture incorporates batch-normalization and dropout mechanisms for the sake of higher stability and robustness to over-fitting, respectively.

The implemented base classifiers are then used in the two proposed ensemble architectures deeply analyzed in the previous chapter: the *MOE ensemble* and the *Stacking ensemble* ensembles that are reported in Figures 20 and 19 respectively.

To help the proposed ensemble classifiers pay attention to rare classes (in case of class imbalance), a class-weighted version of standard categorical cross-entropy loss function in the training process is used. Precisely, the loss on each training instance x is here associated with a weight cl_weight_x linked to the frequency of the class of x (the higher the frequency, the lower the weight), computed as follows: $cl_weight_x = n_sample_cl_x / all_samples$,

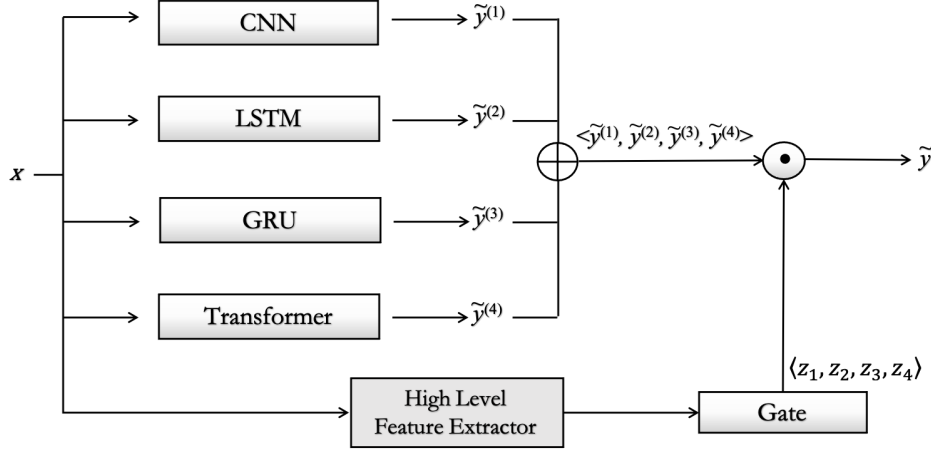


Figure 20: *MOE ensemble* architecture.

where $n_sample_cl_x$ is the number of training examples with the same class as x , while $all_samples$ is the total number of training examples.

5.1.3 Supporting ticket classifiers with a novel Core Human-In-The-Loop scheme

The human-in-the-loop scheme enabled by the proposed framework is described here, it is aimed at supporting the discovery, application, validation and improvement of reliable ticket classifiers. As mentioned above, this scheme exploits a novel combination of advanced (deep ensemble) ticket classification and ad hoc explanation mechanisms.

Figure 21 illustrates the detailed logical architecture. For any new ticket x that is to be handled in the TMS; first, an estimate of the different class-membership probabilities is supplied to the user. These probabilities are computed with the help of an ensemble-based classification model, denoted hereinafter by M , preliminary trained against a given set D of the labelled ticket —specifically, we assume model M to be an ensemble of DNN-based ticket classifiers.

If x appears to clearly belong to one of the classes, the interactive analysis of x ends directly with assigning this class to x . In many real-life application contexts, where decision-makers have limited resources, this helps speed up the ticket classification process, and obtain more accurate classifications.

By contrast, if the class-membership probabilities returned by the ensem-

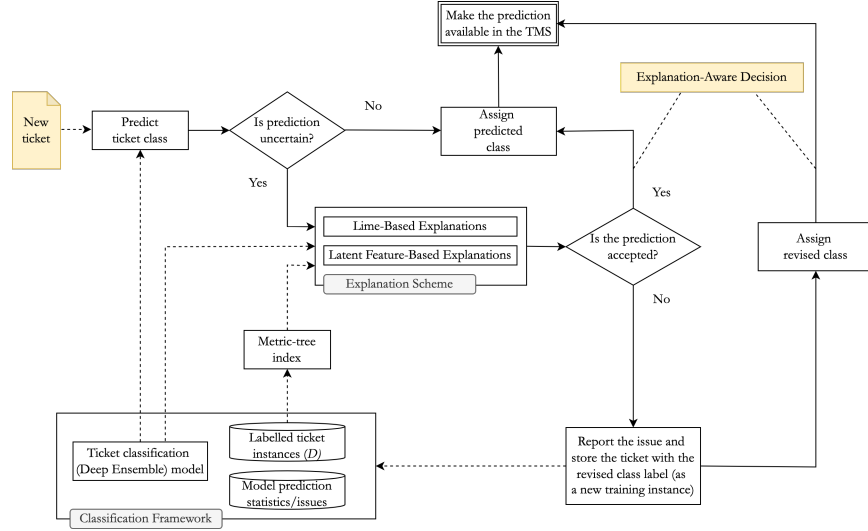


Figure 21: Proposed human-in-the-loop scheme for intelligent ticket classification

ble model M for a new ticket x is uncertain/unconvincing¹, the user can ask for two different kinds of explanation artefacts: LIME-based post-hoc explanations (providing hints on how different parts of the ticket contents may have influenced the prediction results) and summary/detailed information on the neighbours of x in the latent space learnt by (the HLF component of) model M . The details on these explanation artefacts and the way they are computed and presented interactively to the user are treated in the Explanation paragraph.

By shedding light on the behaviour of M on x and on its surrounding region of the instance space, the above-mentioned explanation artefacts allow the operator to classify x in a more conscious way and to revise the suggestion made by the ticket-classification model.

In particular, these artefacts can help discover that a classification model is not trustworthy in some regions of the instance space, likely as a consequence of the fact that the model was not trained with an adequate number of examples of that region so that it tends to rely on spurious (i.e., incidental) classification patterns when classifying new instances in that region. In

¹As every classifier M learnt in our framework is a DNN including dropout layers, the uncertainty of M in classifying a ticket x can be quantified via MC Dropout [21], i.e., by making multiple predictions for x with different (dropout-based) random variants of M , and computing suitable variability/entropy measures over these predictions.

such a case, an issue is reported to the analysts, who may eventually decide to retrain the model, as explained in the final part of the next subsection.

5.1.4 Introducing an Explanation-based Analysis for better interpreting classification errors

Data-driven explanation mechanisms are a valid aid in the core process of classifying a new ticket. In particular, in case of uncertainty, the obtained explanations are expected to help the user make more conscious decisions.

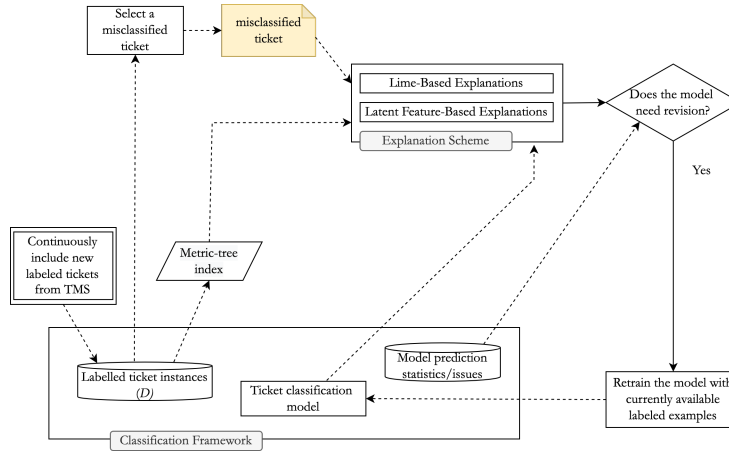


Figure 22: Error analysis use case: exploring explanations for misclassified tickets.

Figure 22 reports the sketch of a different, complementary use case for the explanation methods, where these methods are exploited to investigate errors made by the ticket classification model.

In this use case, the user/analyst requires and explores explanations for some ticket that was misclassified by the model. Clearly, this entails that the ground-truth class label is known with certainty, as in the case of the labelled tickets in the training set or of novel tickets that were assigned a class automatically, but this class was subsequently found to be wrong —think, e.g., of an urgent ticket that is classified as non-urgent, so that a complaint will eventually come from the customer or from another employee for the delay accumulated in handling the ticket. In both cases, explanation mechanisms can be exploited to shed light on the reported inference mismatches and on the factors that could have misled the classification model.

Clearly, this analysis can allow for detecting situations where the ticket

classification model is not working in an accurate and reliable way. To alleviate this problem and improve the competence of the model over the misclassified instances, the analyst can start a re-training procedure for the model by using an extended set of training examples—including novel tickets that have been already classified (starting from the time in which the classification model was trained), and for which the ground-truth class labels are known (i.e. the assigned class labels were validated/revised manually by an expert or by an informed process stakeholder). In this re-training step, one could leverage cost-based learning approaches (assigning higher weights to the training losses of misclassified examples) and possibly discard obsolete training examples (in case the model is suffering from performance degradation caused by some concept drift phenomenon).

In order to support such a continuous improvement of the ticket-classification models, the analysis workflow includes an optional step that permits the storage of two kinds of information in an ad-hoc repository: (i) per-model performance statistics and misclassification reports and (ii) new labelled ticket instances (associated with a manually revised, reliable class label).

5.1.5 The proposed explanation workflow with a neighbour-based artifacts representation

The flowchart scheme in Figure 23 sketches how the user can interactively and incrementally exploit the explanation capabilities described so far for a given ticket x under analysis.

LIME charts are the first concise form of explanation artefact that is presented to the user in case she wants to investigate which properties of the ticket under classification were likely to influence the prediction made on x by the classification model.

If the user feels that more investigation is needed, she can ask to be provided with summary information on the per-class k -neighborhoods of x , presented in the form of word clouds. Algorithm 1 reports a step-by-step description of how these word clouds are computed for a ticket x .

For the sake of deeper inspection, the user can ask to be shown the detailed (ranked) list of the k -nearest neighbours of x for each of the classes.

The exploration of the neighbours of x can be refined iteratively and interactively by the user, who can change the neighbourhood size k at her will.

It is worth pinpointing that the flexible and interactive explanation process described above is led by the user, based on her level of uncertainty about the classification model's predictions.

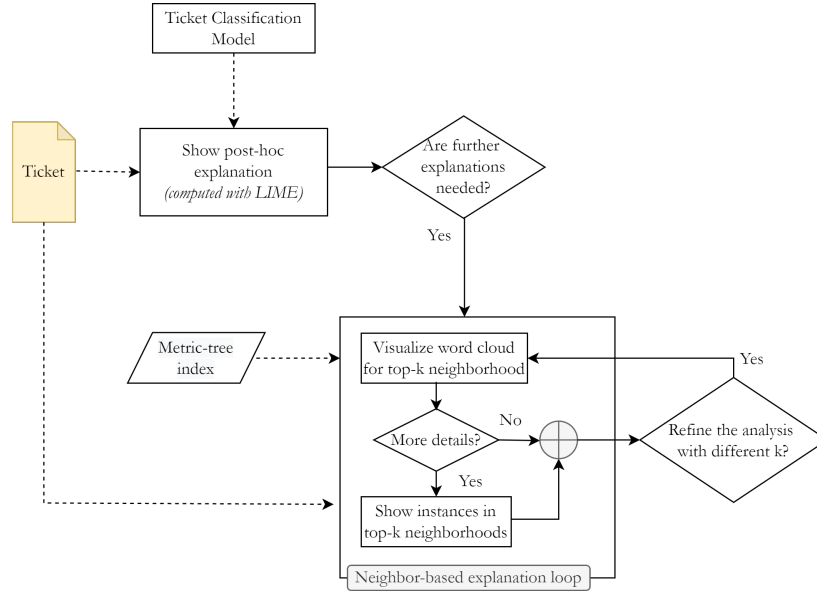


Figure 23: Detailed flow of the explanation process.

5.1.6 Complexity analysis of the Approach

Before illustrating the time complexity of Algorithm 1, a discussion on the implementation of its metric-tree index parameter \mathcal{I}_M and on the cost of initializing it prior to executing the algorithm is here provided. And then an asymptotic study of the time taken by every run of the the algorithm, when provided with such a kind of data index, will be reported.

Time complexity of constructing the metric-tree index In the proposed framework, after training each ensemble-based ticket-classification model M , an index \mathcal{I}_M is built for M (by using the method described in Section 4.3.2), in order to maintain the latent representations (produced by the HLF layer of M) of the labelled instances in the training set D . For the sake of convenience, it is assumed that this index takes the form of a *ball tree* [37], a popular special kind of binary tree where each node represents a hyper-sphere, named *ball*, in the D -dimensional space.² Specifically, every

²It is worth noting that other kinds of data structures (possibly not relying on metric/search-tree structure) could be used in our approach as well, including data structures that only provide approximate, but faster, answers to k -NN queries. In fact, useful explanations for who is classifying a ticket x can be obtained by finding a representative sample of k labelled instances staying close to x (but do not necessarily are its k nearest

Algorithm 1: Pseudo-code for computing neighbour-based per-class explanations in the form of word clouds.

Input : A new ticket x being classified with a ticket classification model M ;
A metric-tree index \mathcal{I}_M , built on the basis of (the
HLFE
subnet of) M and pointing to the labeled tickets in
 D .
The neighborhood size $k \in \mathbb{N}$;
Parameters: The list $[c_1, \dots, c_m]$ of classes defined over the
tickets;
The max number of relevant terms $q \in \mathbb{N}$;
A term relevance function rs (defined as in Equation
3).
Output : A list \mathcal{L} of word clouds.

```

1  $\mathcal{L} = []$ 
2 foreach  $c \in [c_1, \dots, c_m, \perp]$  do
3   //label  $\perp$  here stands for 'whatever class'
4    $N_c = \text{findNeighbors}(\tau, \mathcal{I}_M, k, c)$  //extracts, by using  $\mathcal{I}_M$ , the
   nearest  $k$  neighbors of  $\tau$  that belong to class  $c$  if  $c \neq \perp$  (in this
   case  $N_c = \mathcal{N}_M^k(x)|_c$ ), or to whatever class if  $c = \perp$  (i.e.,
    $N_\perp = \mathcal{N}_M^k(x)$ )
5    $\mathcal{L}_{WT} = \text{deriveTermList}(N_c, \mathcal{I}_M, rs)$  //produce a list of pairs
    $(t, rs(t))$ , for each term  $t$  appearing in the tickets of  $N_c$ , where
    $rs(t)$  is the relevance score of  $t$ 
6    $C_c = \text{buildWordCloud}(\mathcal{L}_{WT}, q)$  //generate a word cloud showing
   the top- $q$  relevant terms in neighborhood  $N_c$ 
7    $\mathcal{L} \stackrel{+}{\leftarrow} C_c$  // append the current word cloud to the list
8 end
9 return  $\mathcal{L}$ 
```

node keeps information on the centre and radius of its associated ball. In particular, the leaves store disjoint subsets of the instances, while any internal node represents a ball that includes those of its children. Data instances are assigned to nodes based on their distance from the node centres.

Since similar instances are expected to fall in the same ball or in close balls, a ball-tree index can enable efficient nearest-neighbour searches. In

neighbours.

particular, in the best and average cases (assuming that the tree is balanced), a k -NN query can be answered in $O(f \times k \times \log n)$ [37], where n is the number of data instances stored in the index, and f is the dimension of their representation (i.e., the number of latent features per data instance).

In order to express the computation-time complexity of building \mathcal{I}_M for a DNN-ensemble classifier M , let n denote the number of instances in the training set D and let f denote the dimension of the data instance representations produced by the HLFE layer in M . Then, by resorting to the first general algorithm described in [37], in our framework, a ball-tree index \mathcal{I}_M for M can be constructed in $O(f \times n \times (\log n)^2)$ steps.

Time complexity of Algorithm 1 For each ticket-group label c in c_1, \dots, c_m, \perp , representing the ticket classes plus an additional group embracing all kinds of tickets, three main computation steps are performed in Algorithm 1: (A) retrieving the neighbors of the given test instance x falling in group c , (B) extracting a list of *(term, relevance)* pairs from the tickets retrieved in the former step; and (C) constructing a word cloud summarizing this list of relevance-weighted terms.

Under the assumptions mentioned in the previous paragraph, step A (Line 4 in the algorithm) can be done in $O(f \times k \times \log n)$ time, where f is the dimension of the latent representations and n is the number of example tickets in D .

The core task in Step B (Line 5) amounts to calculating the relevance scores of each of the terms occurring in the tickets found for group c . This can be done efficiently by exploiting a dictionary (i.e., an associative map) maintaining the (term, score) pairs. If using a binary search tree, the dictionary can be initialized in $O(v \times \log v)$, while retrieving/updating an element in it takes $O(\log v)$ time.

Let v be the total number of distinct terms appearing in the tickets in D (i.e., let v be the size of the vocabulary, built in the model training phase). In $O(k \times v \times \log v)$ time, through a single scan of the k neighbour tickets, it is possible to update the dictionary entries of all the terms in these tickets, according to Equation 3. Indeed, the distance from the query point x was already returned along with each neighbour's tickets, say x' , by the metric tree in Step A (remember, in the algorithm we represented the result as just a set of ticket identifiers for the sake of presentation), while the BOW representation of x' can be retrieved from a separate data structure (if the tickets in D are identified through progressive positive integers, this can be done in constant time). At the end of this scan, the term-score elements in

the dictionary are sorted according to decreasing score values, and eventually put into an ordered list \mathcal{L}_{WT} , in $O(v \times \log v)$.

Finally, Step C just amounts to extracting the top-most q elements from the ordered list, which can be done in $O(q \times \log v) = O(v \times \log v)$ (since $q < v$).

Then, each iteration of the loop takes $O(k \times (f \times \log n + v \times \log v))$ time, so that the total computation time of the algorithm is $O(m \times k \times (f \times \log n + v \times \log v))$.

Thus, the time required to execute the algorithm is expected to be logarithmic (and anyway ensured to be at most linear) in the size of the input dataset D —if regarding parameters m (number of ticket classes plus 1), k (number of neighbours per test instance and ticket group), f (dimension of the latent ticket representations) and v (vocabulary size) as minor constant terms.

5.2 The Ensemble-based self-trained Fake Detection Classifier

A novel fake detection classifier is here described as a specialization of the proposed text classification framework for the discovery, application and evaluation of deep ensemble neural networks aimed at limiting the spreading of fake news, and, specifically devised to face the problem that only a small portion of the available examples of news data is associated with a class label so most of the examples are unlabelled.

Figure 24 illustrates the conceptual fake detection architecture. News coming from different communication channels like news web sources or social networks are collected by the *Data Gathering* block inside the *Information Retrieval* module and wrapped by the *Data Wrapper* block in different fields like news text, title, pictures and eventually tags, web source, owner of the post, or other available information. All these data form the *Knowledge Base* opportunely split in labelled training, validation and test datasets and unlabelled data, i. e. data without labels assigned by a specialist of the field. The proposed *Fake Detection* architecture is designed to overcome the problem of having only a few labelled data, a case that often arises considering that labelling data is a task very expensive in terms of time and money. This problem is overcome by progressively enriching the given labelled data instances with novel pseudo-labelled tuples. At the very beginning of this iterative learning process, a preliminary classification model M_0 is built (by the *DNN model learning* module), by reusing a pre-trained instance of BERT as a backbone. This fine-tuning task is performed by only using the given labelled news data instances, split as usual into a training set and a validation set.

Afterwards, an iterative process is followed, which consists of two phases. In the first phase, the generated model is exploited (by the *Pseudo Labeling Approach* module) to estimate the class of unlabelled data instances, and to assign an artificial class label to some of them eventually; the latter data instances are selected (by the *unlabelled data selection* module) according to different strategies described in detail in the following subsection. In the second phase, the batch of (pseudo-) labelled data instances obtained in the former phase are added to the training set, and exploited, together with those already available before, to train a new version of the classification model (e.g., M_1 at iteration 1, M_2 at iteration 2 and so on), which is stored in the *Detection Model Repository*. These phases are iterated until no new element of the pseudo-labelling set meets the constraints defined by the selection strategy (e.g., until the probability of the model correctly predicting

a tuple goes under a given threshold) or until there are no more available unlabelled instances.

Finally, the *Model Extraction/Combiner* module generates the final classifier for detecting incoming fake news opportunely extracting or combining the models generated during the different self-training iterations.

In the current implementation, different strategies are explored. One of the extraction strategies proposed consists in selecting the model and getting the best AUC score on the validation set as the final model. Several ensemble strategies are also proposed to get higher performances. The three ensemble proposed strategies are the *pseudo-avg-ens*, *avg-ens* and *ps-ens*. In the *pseudo-avg-ens*, the output predictions of the intermediate models are averaged in order to obtain an ensemble prediction that equally takes into account the decisions of each base model. The *avg-ens* and the *ps-ens* are ensemble models based on a weighted average combiner, but the weights are computed differently, as explained in detail in the next section.

The *Performance Evaluation* module returns different evaluation metrics used in the experimental section.

The final *Ensemble Model* module is used to classify the news as fakes or true thus supporting the reader with an important instrument for verifying the veracity of the news massively generated on the web.

After each iteration of pseudo labelling generation, the model parameters are not saved from previous training but reinitialised randomly, thus reducing possible consequences of confirmation bias due to the overfitting to incorrect pseudo-labels used for training the model in the previous steps.

5.2.1 The ensemble strategy for effective training with few labelled data

The pseudo-code in Algorithm 2 explains in detail how the proposed entire pseudo-labelling self-training process works.

Given as input an initial Training Set (*ITrS*), a Validation Set (*VS*) and a set of unlabelled instances (*ULS*) coming from a stream of news, an initial model *M* is trained using the two sets *ITrS* and *VS*. The learning process goes on through multiple training rounds, using both the manually-labelled data initially stored in *ITrS* and *VS*, and the pseudo-labelled data added to *PsS* (i.e. data without true labels that are labelled based on the predictions returned by the model obtained at former iterations). More precisely, the following operations are performed until no new pseudo-labelled instances are added to the current training set (*TrS*).

First, for each instance of the unlabelled set, an uncertainty score *U* is

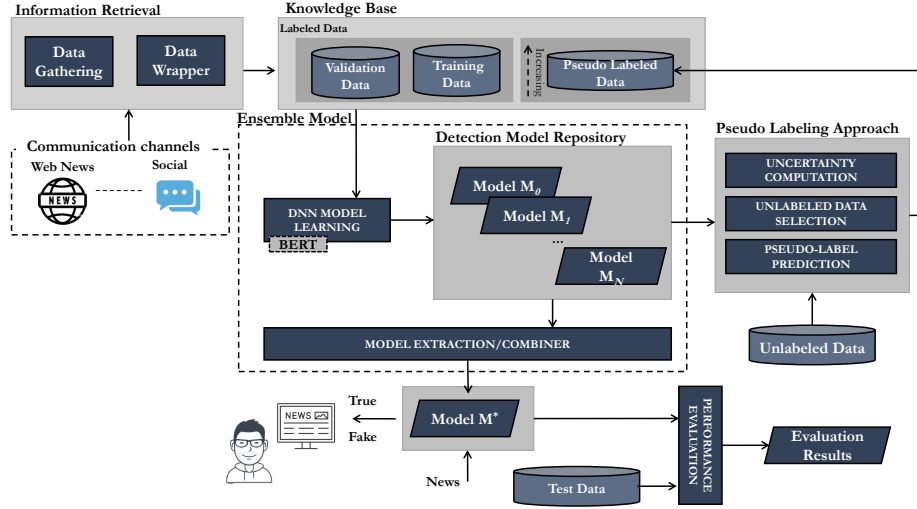


Figure 24: The Fake Detection architecture.

computed, which is meant to estimate how much the prediction returned by model M for x is uncertain. In principle, different uncertainty estimation methods [1] could be adopted for this aim. In the implementation of the framework that was employed in the experimental analysis, the uncertainty scores are simply derived from the highest class membership probability returned by M for x (the closer this probability is to 0.5 the higher the uncertainty degree).

Then, a subset X of instances taken from ULS are selected for being pseudo-labelled by preferring those ones on which the current model M seems to be less uncertain. Two different strategies can be adopted to make this selection step (described later on), and they can be controlled through the parameter *strategy* of Algorithm 1.

The selected instances in X are automatically assigned a pseudo label with the help of the current model M , and put into a new temporary (Pseudo-Label) set PsS . All of these pseudo-labelled instances are added to the current training set TrS , while removing all the instances of X from the set ULS of unlabelled data instances.

Ensemble and combiner strategies In order to take into account the fact that the pseudo-labels are not true labels (i.e. labels assigned by experts and then to be considered true for sure) but labels assigned by the

currently trained model by opportunely evaluating its prediction and the corresponding certainty, for each pseudo-labelled instance, a weight is computed to be considered in the loss function during the subsequent training iterations. In fact, minimizing the loss function during the training means that the model learns how to minimize the differences between each prediction and each target. The idea of using different weights in the loss function is supported by the consideration that pseudo labels are probabilistic labels; thus, in computing the difference between the prediction and the target, the training has to take into consideration that there is an uncertainty on the target itself. Thus, for each training instance, the model has to learn how to generate an output as close as possible to the target for the true label, while this difference has to be scaled with the certainty level of the target for the generated pseudo-labels (in fact, the target could also be not correct). The labels of the instances in the initial Training Set (*ITrS*) are assigned a weight of one by default because they are true labels. For the pseudo-labelled instances, simply the prediction probability is considered as the weight.

Finally, a new model M is trained from scratch over the (augmented) training set TrS , still using VS as a validation set, after initializing the weights of all the layers of M but the last (which is initialised randomly³) with the weights of the pre-trained BERT model. The newly generated model is then saved in the repository.

At the end of the self-training loop, a final classification model M^* is extracted. Several model extraction/combination schemes are devised by exploiting all the models obtained at the end of each self-training iteration in different ways. In more detail, the explored model extraction schemes consider two selection approaches, one that simply takes the last generated self-trained model and the other that selects the model that performed the best (precisely, the one achieving the highest AUC score) on the validation set VS . More complex ensemble approaches that combine in different ways the output of the trained models have been explored to achieve better performances.

Selection strategy Several alternative strategies can be adopted for selecting which unlabelled data are promoted to pseudo-labelled ones, in or-

³To curb the risk of confirmation bias and of concept drifts, we do not adopt a sort of incremental training scheme where M is initialised with a copy of the model obtained at the previous iteration of the self-training loop (Steps 6-18 of Algorithm 1). Indeed, restarting model parameters before each self-training cycle was identified in [10] as a key to the success of pseudo-labelling approaches to the discovery of deep models.

der to obtain an improved version of the fake news classifier. In more detail, two strategies are here considered. One strategy (chosen when setting *strategy* = *str_thr* in Algorithm 1), simply consists in comparing the uncertainty score of an unlabelled data instance to a given maximal threshold *thr*. The subset of samples in the unlabelled set (*ULS*) to be included in the Training set (*TrS*) is built by selecting the instances for which the lastly trained model *M* returns a prediction with an uncertainty score lower than *thr*.

The second strategy (chosen when setting *strategy* = *str_bestK* in Algorithm 1) consists of the ranking of the instances in *ULS* based on their associated prediction-uncertainty scores and eventually selecting the *k* ones of them achieving the lowest scores.

In both cases, each instance *x*, among those selected as described so far, is artificially assigned a (pseudo-)label that refers to the class for which the model returned the highest class membership probability on *x*.

Extraction/Combiner model strategy Different strategies have been tested for extracting the final classification model from the ones obtained during the different self-training steps of the pseudo-labelling learning process. The first extraction strategy (*str_last_iter*) is a simple selection approach that selects the last trained model in the self-training iteration process. Another extraction strategy (*str_best_auc_val*), by testing the intermediate models on the validation set *VS*, selects the model achieving the best performances (more precisely, the AUC score) as the final classifier. Ensemble-based schemes are also proposed here in order to combine the predictions from the intermediate-trained models in a more complex way. Three ensemble strategies are proposed: *str_avg_ens*, *str_avg_acc_ens* and *str_avg_ps_ens*. The *str_avg_ens* adopts the simplest combiner strategy; in fact, the output predictions of the intermediate models are averaged in order to obtain an ensemble prediction that equally takes into account the decisions of each base model. The *str_avg_acc_ens* is an ensemble model that surpasses the idea of considering all the base models equal to each other, by differentiating the predictions furnished by the different base models depending on the performances (precisely, the ACC score) reached by tests on the validation set *VS*. Thus, the final ensemble model predicts the new instances by computing a weighted average of the base models' output predictions, with each prediction weighted by the accuracy of the corresponding predicting base model. Finally, the *str_avg_ps_ens* is an ensemble model also based on a weighted average combiner, but the weights are computed differ-

ently, taking into consideration the 'goodness' of each model, not on the base of its performances computed on the validation set, but on the base of the 'goodness' of the training set used in its learning phase. In fact, the models are trained with incrementally generated datasets obtained by using the initial training set $I\text{Tr}S$ containing true labels, incremented with the generated pseudo-labels assigned to the selected (by the chosen selection strategy) unlabelled instances. In more detail, the $str_avg_ps_ens$ exploits the level of uncertainty or, more precisely, the level of certainty of the pseudo-labels used to train each model to weigh the average in the ensemble combiner prediction. The weight assigned to each base model is computed as the average of the probability predictions of the pseudo-labels introduced in the training set TrS used to train the base model itself.

Algorithm 2: Pseudo-code for self-training the model with the Pseudo-Labeling algorithm.

```

Input      : Initial Training Set ( $ITrS$ );
               Validation Set ( $VS$ );
               Unlabelled Set ( $ULS$ ); // Unlabelled instances, for which one can
produce pseudo labels
Parameters:  $strategy \in \{str\_bestK, str\_thr\}$ ;
                $thr$ ; // maximal uncertainty threshold (to be used when setting
 $strategy = str\_thr$ )
                $k$ ; // maximum number of instances (to be used when setting
 $strategy = str\_bestK$ )
                $combiner \in$ 
 $\{str\_last\_iter, str\_best\_auc\_val, str\_avg\_ens, str\_avg\_acc\_ens, str\_avg\_ps\_ens\}$ ;
Output    : The model  $M^*$  for classifying the tuples
1  $Train(M, ITrS, VS)$  // train the classifier  $M$  using  $ITrS$  and  $VS$ 
2  $TrS = ITrS$  // Current Training Set
3  $PsS = \emptyset$  // Pseudo labelled Set
4  $MS = \emptyset$  // Trained Model Set with the respective validation performance
5  $newPseudoLabel = True$ 
6 while  $|ULS| > 0$  AND  $newPseudoLabel$  do
7    $newPseudoLabel = False$ 
8    $U = ComputeUncertaintyScores(M, ULS)$  //  $U$  is an ordered list of pairs
of the form  $(x, u)$  such that  $x \in ULS$  and  $u \in \mathbb{R}^+$  is a score quantifying
how much  $M$  is uncertain in making a prediction for  $x$  (with the pairs
ordered based on the score values)
9    $X = SelectUnlabelled(ULS, U, strategy, k, thr)$  // Select the unlabelled
data to be pseudo labelled
10  if  $|X| > 0$  then
11     $newPseudoLabel = True$ 
12     $PsS = \{(x, M(x)) \mid x \in X\}$  // Generate a bunch of pseudo-labelled
instances, by assigning a predicted label to each of the selected
unlabeled instances in  $X$ 
13     $ULS = ULS \setminus X$ 
14     $TrS = TrS \cup PsS$ 
15     $W = GenerateWeights(X, M(X))$  // Generate the weights for the
pseudo-labelled instances to be used in the loss function during the
training phase
16     $Train(M, TrS, W, VS)$  // Train the model  $M$  from scratch using the tr.
set  $TrS$  and the val. set  $VS$  by applying the weights  $W$  to the loss
function
17     $MS = MS \cup (M, Evaluate(M, VS))$  // Save the trained Model and its
performance on  $VS$ 
18  end
19 end
20  $M^* = GenerateModel(MS, combiner)$  // Generate a final classification model
from a combination of the ones computed in all the self-training iterations
21 return  $M^*$ 

```

6 Experimental results

The proposed text classification framework in the two specialized architectures opportunely designed for the ticket management system and for the fake detection have been tested extensively in order to evaluate the performances in the real application scenarios.

6.1 Automatic Ticket Classification Experiments

In this section, the experimental tests conducted over the two real-life ticket datasets, to assess the effectiveness and the usefulness of the proposed intelligent ticket classification framework, are described. In particular, first, a statistical analysis aimed to analyze the characteristic of these datasets is here conducted. Then, the effect of the imbalance-aware loss function is verified, and the comparison of the proposed approach with both the baseline DNN architectures and with some state-of-the-art classifiers is reported. In the next two subsections, the datasets and the evaluation procedure used in the tests are illustrated in detail.

6.1.1 Datasets: description and statistics

Before illustrating the results of the experimentation conducted in this work, in this subsection, the two main datasets used, referred to as *Phone* and *Endava* hereinafter, are described, while showing a number of statistics capturing some major characteristics of these datasets.

Phone dataset Though the classification and explanation framework proposed in this work can be reused in disparate ticket management scenarios, it was originally devised as a solution for improving the handling of CRM tickets concerning requests/issues posed by the customers of an Italian phone company.

The text content of these tickets comes from two channels: SMS messages and Facebook chats. Given the nature of these channels, the length of the messages is usually very short, making the task of classifying them harder.

Each customer request can pertain to one among two categories of products/services: mobile phones and landline phones. Independently from the product/service category, a ticket can be assigned to either special or ordinary support staff.

The combinations of these two orthogonal schemes of ticket categorization gives rise to four classes, namely + *Mobile Ordinary support*, + *Mobile*

Table 2: Main features of the *Phone* and *Endava* dataset.

Dataset	# Tickets	# Classes	Vocabulary# size	Words per ticket			
				Avg.	Median	Q1	Q3
Endava	48,549	4	12,259	39.7	23	13	41
<i>Phone</i>	33,439	4	5,890	9.3	7	3	14

Special support, + *Landline Ordinary support* and + *Landline Special support*, having the following relative frequencies in the dataset: 38.56%, 13.20%, 46.43% and 1.81%, respectively. In the experiments conducted over this dataset, we considered the problem of predicting to which of these classes a ticket belongs based on the text content of the ticket. It can be easily noted that this dataset suffers from a high level of class imbalance, as is often the case in many real-life ticket classification applications.

Endava dataset The so-named *Endava* dataset is a publicly available dataset containing about 50K tickets, submitted via email by the customers of the company Endava to the helpdesk. This dataset and associated information can be found at

<https://github.com/karolzak/support-tickets-classification>.

This dataset does not contain the original free-text descriptions of the tickets, but a pre-processed version, was obtained after applying some text-processing steps (including punctuation/stopword removal and case normalization). Each ticket in this dataset is assigned several labels, which concern different dimensions (e.g., the urgency or the impact of the ticket). In the proposed tests, only the urgency label is considered as the class label to be predicted. (based on the sole text contents of a ticket while disregarding all the other labels associated with the ticket).

The resulting classes and the respective frequencies are the following: *High Urgency* (3.40%), *Medium Urgency* (13.90%), *Low Urgency* (11.39%) and *No Urgency* (71.31%). Thus, also this dataset is imbalanced.

Statistical analysis of the datasets Before studying the application of the proposed approach to the two datasets, results from a preliminary statistical analysis of the datasets are shown here. In fact, such analysis allows to capture some major characteristics of the two datasets and was a valid help for better devising the pre-processing of data and the configuration of the classification methods.

Table 10 shows the overall number of tickets, the vocabulary size and some

statistics on the number of words per ticket (i.e., average, median, first and third quartile). From this table, it is evident that the number of tuples for the two datasets is very similar; however, while the *Phone* dataset presents very short messages, the *Endava* dataset consists of a little longer messages (about 9 words and about 40 words for a ticket on average, respectively). The size of the vocabulary and the median and the quartiles are in line with the different consistency of the two datasets.

Table 3: Class statistics computed on the *Phone* dataset (*MSS* = Mobile Special, *MOS* = Mobile Ordinary, *LOS* = Landline Ordinary, *LSS* = Landline Special Support) and the *Endava* dataset (class: *Urgency*).

	Class	# Tickets	% Tickets	# Words per ticket			
				Avg.	Median	Q1	Q3
Phone	MSS	4,414	13.20	8	6	3	13
	MOS	12,894	38.56	8	7	3	13
	LOS	15,527	46.43	9	8	4	14
	LSS	604	1.81	9	7	4	14
	Overall	33,439	-	9	7	3	14
Endava	High	1,652	3.40	52	31	20	56
	Medium	6,748	13.90	53	32	21	56
	Low	5,528	11.39	50	31	20	52
	No	34,621	71.31	34	19	12	35
	Overall	48,549	-	39	23	13	41

A more detailed analysis of the characteristics and distributions of the classes (and of the overall dataset) is reported in Table 3. Looking at this table, it is possible to note the unbalanced nature of both datasets (the minority class is about 1.8% for *Phone* and about 3.4% for *Endava*). Moreover, it is clear that the distribution of the words for each class is quite uniform (in terms of average, median and quartiles), if excluding the *No Urgency* class in *Endava* (34 words for tickets vs about 50 words in the other classes). The different distribution of class *No Urgency* is likely due to the fact there is no need for so many words to describe a problem that is not urgent at all. Anyway, from this analysis, it does not seem to emerge any further remarkable characteristics that could be exploited in better tuning learning/classification steps.

Figures 25 and 26 illustrate the ticket length distribution for the four classes in the datasets *Phone* and *Endava*, respectively. It is important to observe that the values on the vertical axis are reported on a logarithmic scale for the sake of readability and easier comparison. For these datasets, however, the differences in the distribution of the word lengths reflect the average distribution in the classes, as there seem to be no other remarkable features.

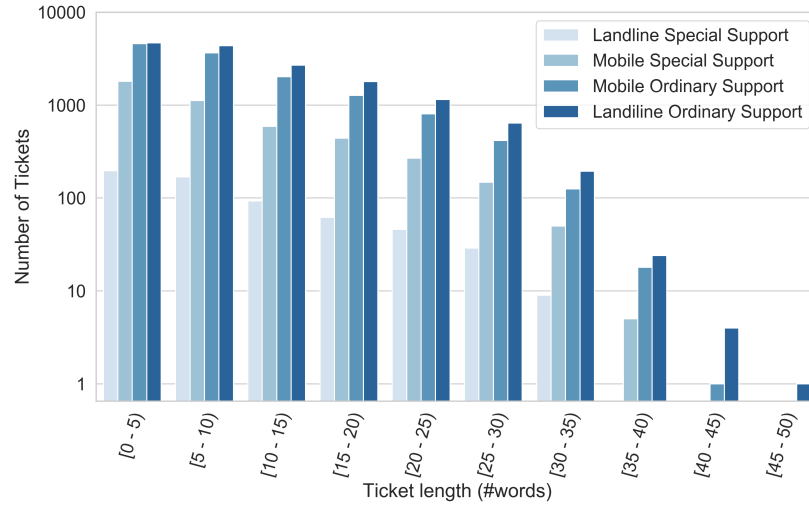


Figure 25: Ticket length distribution w.r.t. the four classes for the Phone dataset.

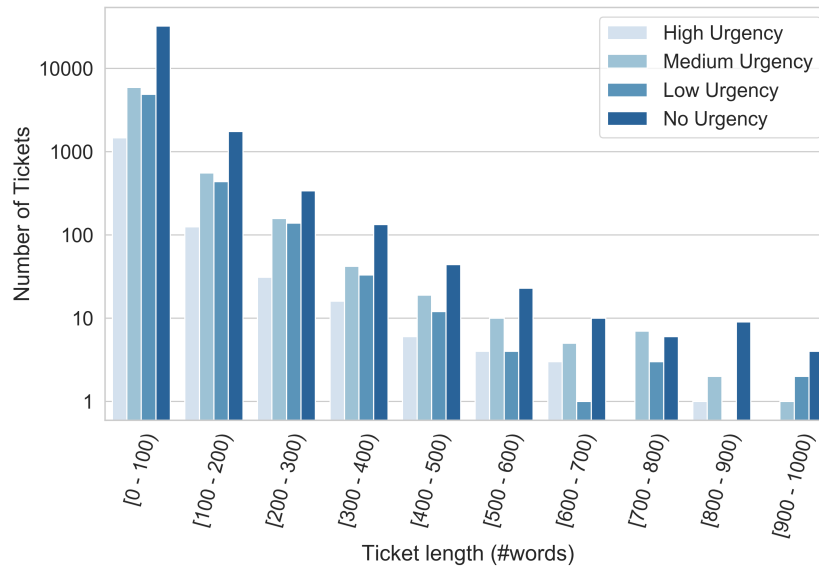


Figure 26: Ticket length distribution w.r.t. the four classes for the Endava Dataset.

Figures 27 and 28 show the 25 most frequent words (entities are reported in angle brackets) in the datasets *Phone* and in *Endava*, respectively. In the former, it is possible to notice the relatively high frequency of some entities (i.e., the mobile and landline phone number, the company name, the date) and of certain domain-specific terms (i.e., request, pay, activate, operator, line).

In the *Endava* dataset, the most frequent terms concern certain temporal dimensions (namely, the day of the week and the month) and TMS-related concepts (e.g., issue, log, form, error).

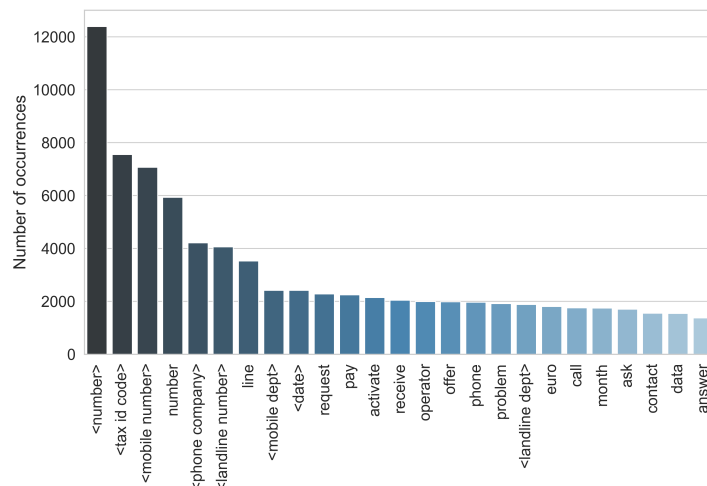


Figure 27: Top 25 most frequent words in dataset *Phone* (entities are reported in angle brackets).

6.1.2 Implementation, configuration, and test procedure

A Python prototype implementing the framework in Section 4 was developed by leveraging libraries Keras and Tensorflow. The source code for test replicating the ticket-classification experiments can be found at <https://github.com/PaoloZicari/IntelligentTicketManagement>

The DNN architectures of the base classifiers have been configured as follows. In the *CNN* architecture, a stack of five convolutional layers was used, all equipped with ReLU activations, and consisting of 64 neurons. In both *LSTM* and *GRU* architectures, each recurrent layer consists of 256 units (equipped with `tanh` activations and `sigmoid` recurrent-step activations),

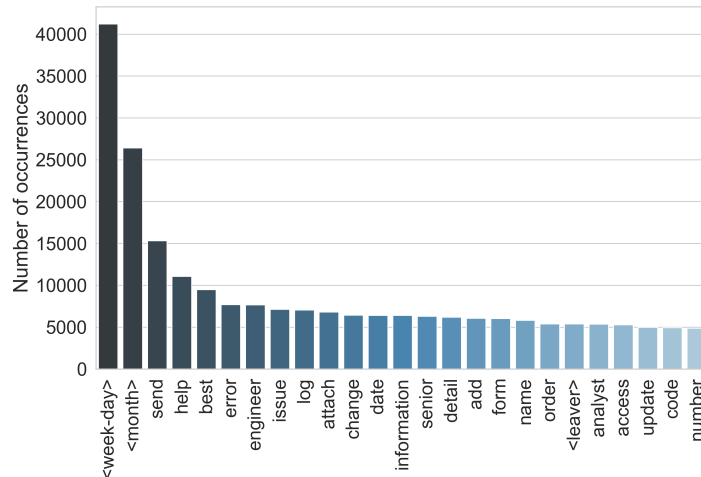


Figure 28: Top 25 most frequent words in dataset *Endava* (entities are reported in angle brackets).

and fixed the dropout percentage to 0.25. Two attention heads of size 32 were used in the *Transformer* architecture. In all the HLFs sub-nets, the output sizes of the Word Embedding and Fully Connected layers were fixed to 128 and 64, respectively.

In each test run, the data were split into a training set (70%) and a test set (30%) in a stratified fashion in order to preserve the original distribution of the class labels. All the DNN-based classifiers were trained for 32 epochs, using a batch size of 64, and Adam optimizer. The performance of each discovered classifier were measured against the test set through three different metrics: AUC (Area Under the Curve), G-Mean and F-Measure metrics. Accuracy, precision and recall metrics have not been considered in this specific context of experimental test measurements because providing misleading results on imbalanced data. An aggregate evaluation score is presented in the following: for each discovery method M and each evaluation metric ϕ , computed by averaging the scores assigned by metric ϕ to the classifiers that were found with the help of method M in 20 different runs.

6.1.3 Effect of the imbalance-aware loss function

The experiments described in this subsection were aimed at assessing whether the proposed classification methods *Stacking ensemble* and *MOE ensemble*

Table 4: Comparing the proposed deep ensemble learning algorithms with and without the imbalance-aware loss function for the *Phone* dataset (the values in bold are significantly better than the others).

Algorithm	Balancing	AUC	G-mean	F-measure
<i>Stacking ensemble</i>	No	0.809 ± 0.007	0.618 ± 0.009	0.434 ± 0.021
	Yes	0.811 ± 0.006	0.659 ± 0.008	0.471 ± 0.003
<i>MOE ensemble</i>	No	0.809 ± 0.007	0.616 ± 0.010	0.430 ± 0.021
	Yes	0.810 ± 0.009	0.659 ± 0.009	0.467 ± 0.007

Table 5: Comparing the proposed deep ensemble learning algorithms with and without the imbalance-aware loss function for the *Endava* dataset (the values in bold are significantly better than the others).

Algorithm	Balancing	AUC	G-mean	F-measure
<i>Stacking ensemble</i>	No	0.953 ± 0.001	0.739 ± 0.005	0.568 ± 0.011
	Yes	0.954 ± 0.001	0.779 ± 0.004	0.620 ± 0.006
<i>MOE ensemble</i>	No	0.952 ± 0.001	0.734 ± 0.008	0.556 ± 0.021
	Yes	0.953 ± 0.001	0.778 ± 0.005	0.613 ± 0.007

benefit from the imbalance-aware loss function defined in this work. Table 4 and 5 report the results obtained on the datasets by both kinds of deep-ensemble classifiers with and without using the imbalance-aware loss function against the datasets *Phone* and *Endava*, respectively. The *Stacking ensemble* and the *MOE ensemble* algorithms perform significantly better when configured with the weighted loss function in terms of both G-mean and F-measure scores, over both datasets. Minor differences between the versions of the classification method are observed when evaluating the AUC metric; as a matter of fact, this behaviour was expected since this metric does not fully take into account the unbalanced nature of the data.

6.1.4 Comparison of the proposed approach with the baselines

This subsection compares the two ensemble methods proposed here with the base DNN classifiers they are based upon, in order to establish whether the adoption of an ensemble paradigm really leads to better predictions.

Table 6 and 7 report the results obtained by both the proposed deep ensemble approaches and the DNN baselines against the *Phone* and *Endava* datasets, respectively. It is evident that, for all the evaluation metrics and datasets, our ensemble-based methods perform significantly better than the baselines. Only for the F-measure metric on dataset *Endava*, *Stacking ensemble* outperforms *MOE ensemble*, but the differences are not significant; in all the other cases, the results of these methods do not differ

Table 6: Comparing the proposed deep ensemble learning methods with the four deep learning baselines on the *Phone* dataset (the values in bold are significantly better than the others).

Algorithm	AUC	G-mean	F-measure
<i>LSTM</i>	0.796 ± 0.006	0.649 ± 0.007	0.447 ± 0.005
<i>CNN</i>	0.787 ± 0.013	0.636 ± 0.012	0.435 ± 0.008
<i>GRU</i>	0.798 ± 0.005	0.646 ± 0.008	0.444 ± 0.013
<i>Transformer</i>	0.793 ± 0.013	0.631 ± 0.006	0.429 ± 0.015
<i>Stacking ensemble</i>	0.811 ± 0.006	0.659 ± 0.008	0.471 ± 0.003
<i>MOE ensemble</i>	0.810 ± 0.009	0.659 ± 0.009	0.467 ± 0.007

Table 7: Comparing the proposed deep ensemble learning methods with the four deep learning baselines on dataset *Endava* (the values in bold are significantly better than the others).

Algorithm	AUC	G-mean	F-measure
<i>LSTM</i>	0.948 ± 0.001	0.765 ± 0.007	0.598 ± 0.012
<i>CNN</i>	0.943 ± 0.003	0.755 ± 0.008	0.573 ± 0.010
<i>GRU</i>	0.949 ± 0.001	0.757 ± 0.022	0.575 ± 0.033
<i>Transformer</i>	0.944 ± 0.003	0.757 ± 0.012	0.584 ± 0.026
<i>Stacking ensemble</i>	0.954 ± 0.001	0.779 ± 0.004	0.620 ± 0.006
<i>MOE ensemble</i>	0.953 ± 0.001	0.778 ± 0.005	0.613 ± 0.007

substantially. Among the baselines, the best achievements are obtained by *LSTM* and *GRU*.

6.1.5 Comparison of our approach with state-of-the-art algorithms

In this section, the two proposed ensemble-based methods are compared with: (i) two state-of-the-art ensemble-based classification algorithm, namely *Random Forest* and *Gradient boosting*, and with (ii) several standard classification techniques that were recently shown to perform well in some TMS datasets [52, 53], namely Naive Bayes, Decision Tree, K-Nearest-Neighbors and Support Vector Machine (SVM) classifiers. For all the competitors (i.e., Naive Bayes, Decision Tree, K-Nearest-Neighbors, SVM, Gradient Boosting and Random Forest), we resorted to the respective implementations available in popular machine-learning library *scikit-learn* [47], without performing any kind of parameter tuning where not specified. In more detail, the Complement Naive Bayes (CNB) version of Naive Bayes was used in the experiments, since CNB was shown in [51] to fit better the nature of text data and to ensure more stable weight estimates and better classification performances on such data.

Table 8: Comparing the proposed deep ensemble learning methods with state-of-the-art Machine Learning algorithms, on dataset *Phone* (the values in bold are significantly better than the others).

Algorithm	AUC	G-mean	F-measure
<i>Naive Bayes</i>	0.779 ± 0.006	0.648 ± 0.006	0.462 ± 0.005
<i>Decision Tree</i>	0.722 ± 0.007	0.579 ± 0.004	0.399 ± 0.006
<i>K-Nearest Neighbors</i>	0.616 ± 0.005	0.535 ± 0.011	0.351 ± 0.016
<i>Support Vector Machine</i>	0.763 ± 0.005	0.603 ± 0.005	0.426 ± 0.005
<i>Gradient Boosting</i>	0.789 ± 0.009	0.627 ± 0.006	0.467 ± 0.009
<i>Random Forest</i>	0.799 ± 0.009	0.609 ± 0.004	0.428 ± 0.006
<i>Stacking ensemble</i>	0.811 ± 0.006	0.659 ± 0.008	0.471 ± 0.003
<i>MOE ensemble</i>	0.810 ± 0.009	0.659 ± 0.009	0.467 ± 0.007

Table 9: Comparing the proposed deep ensemble learning methods with state-of-the-art Machine Learning algorithms, on dataset *Endava* (the values in bold are significantly better than the others).

Algorithm	AUC	G-mean	F-measure
<i>Naive Bayes</i>	0.928 ± 0.001	0.728 ± 0.005	0.566 ± 0.007
<i>Decision Tree</i>	0.900 ± 0.003	0.717 ± 0.004	0.563 ± 0.006
<i>K-Nearest Neighbors</i>	0.698 ± 0.003	0.598 ± 0.026	0.424 ± 0.007
<i>Support Vector Machine</i>	0.940 ± 0.001	0.660 ± 0.004	0.522 ± 0.007
<i>Gradient Boosting</i>	0.945 ± 0.001	0.728 ± 0.005	0.587 ± 0.008
<i>Random Forest</i>	0.945 ± 0.001	0.696 ± 0.004	0.536 ± 0.004
<i>Stacking ensemble</i>	0.954 ± 0.001	0.779 ± 0.004	0.620 ± 0.006
<i>MOE ensemble</i>	0.953 ± 0.001	0.778 ± 0.005	0.613 ± 0.007

For the Decision Tree method, we fixed the maximum depth to 10. For K-Nearest-Neighbors we used a neighborhood of 3 and set parameter *weight* to “distance” (in order to weight points by the inverse of their distance). As to SVM Classifier, we specifically resorted to the version embodied by algorithm *ν -Support Vector Classification* (ν -SVC) [11], with $\nu = 0.09$ and a polynomial kernel of degree 3 —parameter ν allows for directly controlling the number of support vectors and the margin errors (an upper bound on the fraction of margin errors and a lower bound of the fraction of support vectors).

All the above-described settings were chosen after performing a grid-search procedure.

Table 8 and 9 report the results obtained by the above-cited competitors, compared with those of ensemble-based methods, for the *Phone* and *Endava* datasets, respectively. It is evident that, for all the evaluation metrics and datasets, our ensemble-based methods perform significantly better than all the other algorithms. Only for the F-measure metric on *Phone*,

Gradient Boosting obtains the same performance as our ensemble-based methods. Among the other algorithms, the *Naive Bayes* one obtains results comparable with the ensemble-based competitors (especially on dataset *Endava*), while *Decision Tree* and *K-Nearest Neighbors* tend to stay under the performance of the other algorithms —with the exception of *Decision Tree* on dataset *Endava*.

6.1.6 Analysis of statistical significance

In order to assess the significance of the differences observed between the different methods considered in the experimentation, the popular *Friedman-Nemenyi* statistical procedure [13, 22], a widely used in the evaluation of classifiers, was adopted. This procedure essentially consists of two phases. First, the Friedman test is employed to possibly reject the null hypothesis H_0 that the different populations of results (produced each by a distinguished classifier-induction method) have the same mean, so that they can be considered as statistically different ($\alpha = 0.05$). Then, if H_0 is rejected, a post-hoc Nemenyi test with a significance level of 0.05 is used (as post-hoc test) to detect all the pairs of methods that are significantly different from one another: if a pair of methods is assigned a p -value under 0.05, these methods are eventually deemed as significantly different from a statistical viewpoint.

For the sake of presentation, in this statistical-significance study the focus will be given only to a subset of the classification methods considered so far. Specifically, among all the six different competitors, only the two best-performing ones, i.e. state-of-the-art ensemble-based methods *Gradient Boosting* and *Random Forest* will be considered here.

Figure 29, 30 and 31 shows the *critical difference* (CD) diagram [13, 22], respectively for the metrics of AUC, G-mean and F-measure, obtained by using the Friedman-Nemenyi procedure described above. In this diagram, two methods are connected through a horizontal line iff they resulted not significantly different from one another.

It is evident from the figures that *Stacking ensemble* and *MOE ensemble* are not significantly different for all the metrics. However, for both AUC and G-mean, the two ensembles proposed here are significantly better than all the base algorithms and all the ensemble-based competitors. Only for the F-measure metric, *MOE ensemble* does not perform significantly differently from *Gradient Boosting*, but the latter looks also similar to the other two methods (namely, *LSTM* and *GRU*) that are neatly overcome by *MOE ensemble*. Anyway, also with F-measure, *Stacking ensemble* confirms

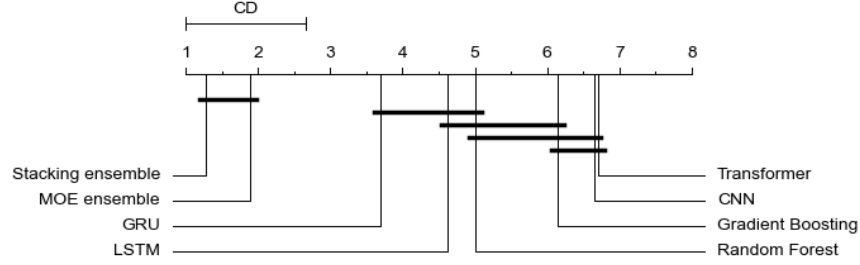


Figure 29: Critical difference (CD) diagram for AUC scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a $p\text{-value} \geq 0.05$) are connected through a horizontal line.

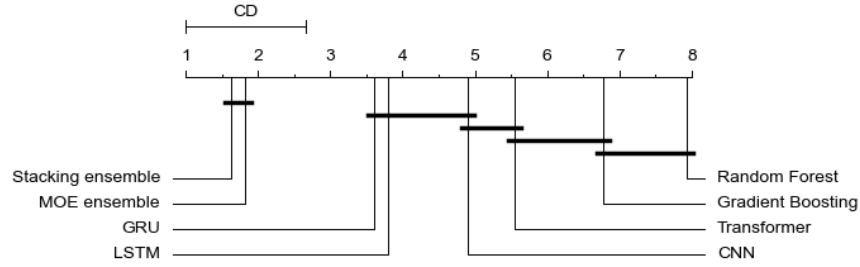


Figure 30: Critical difference (CD) diagram for G-mean scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a $p\text{-value} \geq 0.05$) are connected through a horizontal line.

its superiority in comparison with all the other competitors.

6.1.7 Explanation results

Two examples are here shown illustrating the exploitation of the explanation capabilities of the proposed framework in a simulation scenario where the ensemble classifier discovered with method *Stacking ensemble*, named hereinafter M , is used to predict the class of two tickets from dataset *Endava*. These tickets, named hereinafter x_1 and x_2 , were both associated with the highest urgency class, referred to as either *High Urgency* or class 0 in what follows.

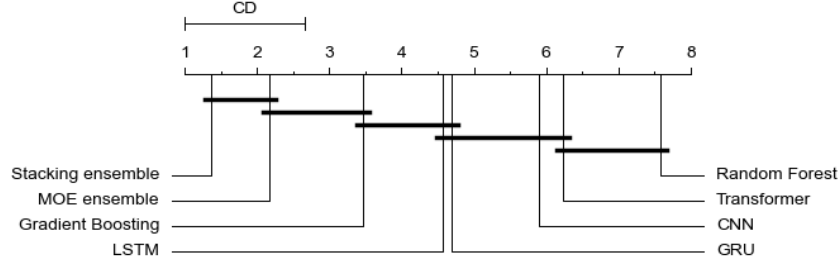


Figure 31: Critical difference (CD) diagram for F-measure scores (Friedman test + Nemenyi test, $\alpha = 0.05$). All (and only) the pairs of methods resulting not significantly different according to the test (i.e., receiving a p -value ≥ 0.05) are connected through a horizontal line.

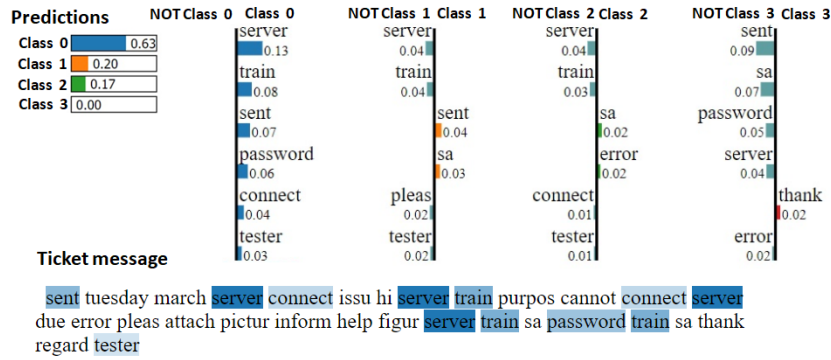


Figure 32: LIME-based explanation obtained for test ticket x_1 .

The text of the ticket x_1 (related to problems with server connection password notified by a tester in a follow-up message) is shown in Fig. 32, which also reports the prediction returned for x_1 by the proposed classifier M and the respective local explanation obtained with the proposed LIME-based procedure. Clearly, M guesses the real class (i.e., class 0) of this ticket –which is given a (relatively high) membership probability of 0.63. The terms that LIME deems most influential for this prediction decision (highlighted in blue in the ticket message) confirm that the HLFN subnet of M focuses on concepts (namely, *server*, *connection*, *password*, *tester*, *sent*) that are really relevant for deciding the urgency class of x_1 . The relevance of these terms was confirmed by the word clouds derived from the $k = 100$ nearest neighbors of x_1 in the latent space of M , omitted here for brevity.

All these explanation artefacts support the prediction of M for x_1 .

The text of ticket x_2 is shown in Fig. 33, along with the class predictions and LIME-based artefacts returned by our framework. Differently from the previous case, the ensemble classifier is uncertain on the class of this ticket (the membership probabilities of classes 1 and 2 are similar).

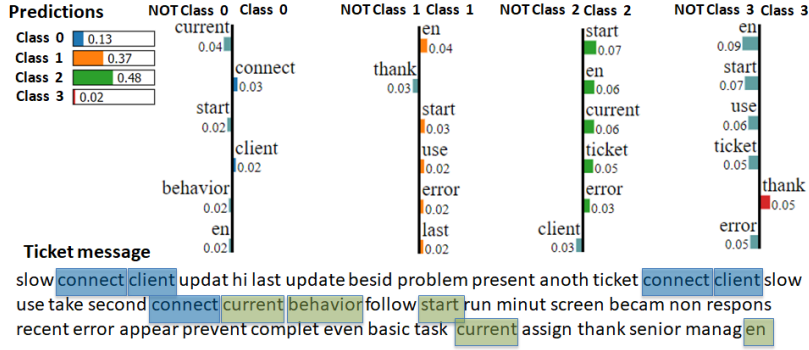


Figure 33: LIME-based explanation obtained for test ticket x_2 .

In fact, the explanation obtained with LIME for this prediction is not convincing, for it relies on terms (namely, *start*, *current*, *en*) that fail to capture the semantic of x_2 . Indeed, by looking at the text of x_2 , a human can easily understand that it regards a serious problem (non-responsive screen, and the impossibility of performing basic tasks) that should be assigned a high level of urgency.

In fact, after performing a k -NN search (with $k = 100$) in the latent space associated with the ensemble classifier, we discover that the two nearest example tickets of class 0 share some semantic similarity with x_2 , in that they also seem to be related to blocking malfunctions (affecting a battery and an interface device, respectively, instead of a server).

The (pre-processed) text contents of these example tickets (lying at a distance of 0.539 and 0.367, respectively, from x_2 in the latent space) are reported below:

- “battery en laptop broke suddenly hi based have using laptop inventory en since today laptop battery seems have failed completely out sudden if remove charger connector leaving laptop run battery laptop shuts down immediately even if battery charging percentage widows had laptop connected charger throughout if try start up laptop with charger disconnected won start looks like battery drained please could you look into makes laptop use extremely thanks regards senior manager”

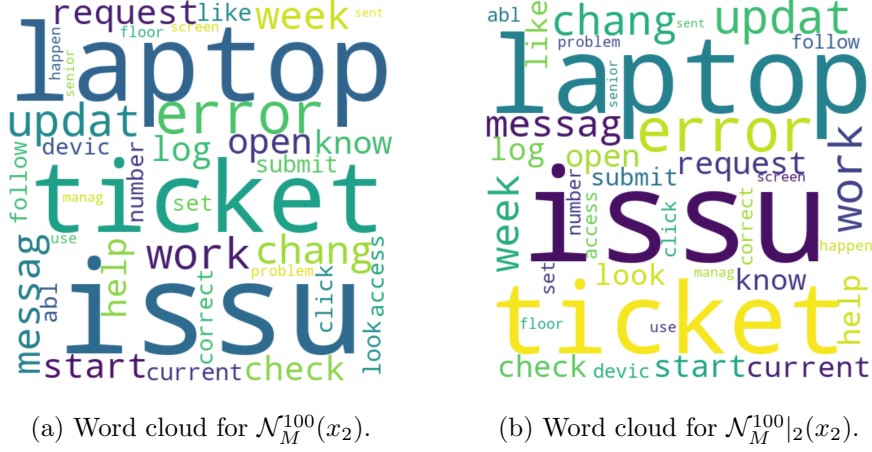


Figure 34: Word clouds summarizing the neighbour sets $\mathcal{N}_M^{100}(x_2)$ and $\mathcal{N}_M^{100}|_2(x_2)$ returned by the framework for test ticket x_2 —the latter word cloud summarizes the example tickets that are both in the neighborhood of x_2 and have the class label (namely, class=2, i.e. *Medium Urgency*) as the one predicted for x_2 .

- “issues hello during course today we have experiencing intermittent issues with interface stops responding various stages interface buttons can longer be pressed best regards click here find out more about experience en address blvd th floor district please refer section our for list entities”

However, by analyzing the word clouds of the 100 nearest neighbours of x_2 (in the latent space) in Fig. 34, a human operator can easily reckon that the ensemble model is not working appropriately in the proximity of x_2 within the latent space, and thus, it needs to be refined/retrained. Indeed, these word clouds unveil that, based on the ticket embedding learnt by the model, some semantically-relevant terms (e.g., *responsive*, *slow*, *basic*), which help reckon x_2 as a severe and urgent case, do not appear at all in the tickets that the model is perceiving as neighbours of x_2 .

6.2 Fake Detection Experiments

The proposed semi-supervised deep learning ensemble-based framework devised to effectively detect fake news by coping with the data scarcity problem has been tested for evaluating its performance.

Experiments conducted on two public datasets confirmed the quality of the approach in generating accurate models, also when a limited number of training examples are available in the early stages of the proposed semi-supervised method.

6.2.1 Datasets and Parameters

This subsection describes the parameters used in the proposed framework and the datasets used to assess its effectiveness in detecting fake news.

The learning model employed in the performed tests is based on a *BERT* layer followed by a *Dropout* layer for regularisation and a final dense layer with a sigmoid activation layer. The *BERT* implementation presents a vector of hidden size of 768, and 12 attention heads. The used model is pre-trained for the English language on *Wikipedia* and *BooksCorpus*, after a normalisation phase.

The following parameters were used in BERT: Number of Epoch = 30; Batch size = 32; Learning Rate = $3e - 5$; Dropout = 0.1; the Binary Cross entropy as loss function and the chosen optimiser was AdamW, a stochastic optimisation method that modifies the typical implementation of weight decay in Adam, by decoupling weight decay from the gradient update.

It is worth recalling that several strategies of extraction/combination can be employed in the proposed framework that differently exploit the trained models built at the different steps of the self-training process. For the extraction process, two strategies can be adopted, the *str_best_auc_val* and the *str_last_iter*. The first extraction strategy selects the trained model with the best AUC performances obtained with the validation set, while the second strategy selects just the last trained model in the iteration process. As regards, the combination strategies, three ensemble combiners can be performed by the framework, *str_avg_ens*, *str_avg_acc_ens* and *str_avg_ps_ens*. They are all based on the average combination of the output of the trained models, considered as base learners of the ensemble, obtained at different steps of the self training process. The *str_avg_ens* computes the average of the base learners' output, while *str_avg_acc_ens* and *str_avg_ps_ens* compute a weighted average. The *str_avg_acc_ens* combiner strategy uses the accuracy of the base learner model as the weight, while the *str_avg_ps_ens* combiner

Table 10: Main features of the *PolitiFact* and *GossipCop* dataset.

Dataset	#Articles	#Classes	Vocabulary size	# Words per article			
				Avg.	Median	Q1	Q3
<i>PolitiFact</i>	814	2	60,870	817.5	199.5	66.5	530.7
<i>GossipCop</i>	4,719	2	149,557	349.0	205.0	109.5	323.0

strategy uses the average of the probability predictions of the pseudo-labels introduced in the training set used to train the base models.

Moreover, it is also worth recalling that two strategies can be employed in the proposed framework for iteratively selecting unlabelled data to be pseudo-labelled: strategy *str_thr* (which relies on filtering candidates through a maximal uncertainty threshold) and strategy *str_bestK* (which extracts the “bottom-k” tuples with the lowest prediction uncertainty).

A grid search was performed to choose the probability prediction *thr* in the case of the *str_thr* strategy and the number *k* of the best-*k* unlabelled data to be pseudo-labelled at each self training iteration for the *str_bestK* algorithm. Respectively, the values of *thr* = 0.4 and *k* = 100, and the values of *thr* = 0.3 and *k* = 200 were chosen for the *PolitiFact* and for the *GossipCop* dataset.

All the experiments of the next subsection were averaged over 30 runs. The validation set is used in the training process for selecting the best model during the different epochs, and at the end for selecting the final model among the models trained at the different self-training iterations.

The two datasets used for the experiments come from the *FakeNewsNet* data repository [58] [57]. They respectively concern political and gossip news obtained by two fact-checking websites: *PolitiFact*⁴ and *GossipCop*⁵.

Table 10 reports the main characteristic of the two datasets: the overall number of articles, the vocabulary size and some statistics on the number of words per article (i.e., average, median, first and third quartile).

The performance of the proposed methods and of the baseline is evaluated against the test set through three different metrics: the largely used Accuracy metric and some measures more appropriate for evaluating unbalanced datasets, i.e., AUC (Area Under the Curve) and F-Measure.

⁴<https://www.politifact.com/>

⁵<https://www.gossipcop.com/>

6.2.2 Experimental validation of the pseudo-labelled based self-training proposed model

This subsection reports the evaluation of the proposed pseudo-labelling strategies in comparison with the baseline when different percentages of labelled data (training set and validation set) are considered (2.5%, 5%, 10% and 20%), in order to consider the situation in which a few (costly) labelled data are available.

The traditional method consisting in fine-tuning the same pre-trained BERT model in a fully-supervised against the sole labelled data is considered as the baseline for all the tests. After evaluating some preliminary experiments, it was decided to make extensive testing considering only some of the possible strategies selectable in the framework for the pseudo-labelling-based self-training. More specifically, among the two pseudo-label selection strategies (the *pseudo_thr* and the *pseudo_k*), only the *pseudo_thr* was reported because of its better performance. The preliminary experiments have also shown insignificant performance variations between the two extraction strategies (*str_best_auc_val* and *str_last_iter*), thus leading to choose the second strategy (*str_last_iter*) for completing the experimental results, that is also the simplest solution which selects the last trained model, i.e. the model that completed the entire pseudo-labelled based self-training algorithm. As regards the ensemble strategies, just the *str_avg_acc_ens*, and the *str_avg_ps_ens* combiners have been reported because the *str_avg_ens* strategy did not bring any particular improvements.

Thus summarizing, the proposed experiment results report the comparison of the baseline with three pseudo-labelling-based self-trained models here named: *pseudo*, *avg_ens* and *ps_ens*. They all use the threshold based (*str_thr*) uncertainty pseudo-label selection strategy. However, the *pseudo* uses the *str_last_iter* extraction strategy, while the *avg_ens* and *ps_ens* use the *str_avg_acc_ens* and the *str_avg_ps_ens* ensemble combiner strategies, respectively.

Tables 11 and 12 report the results of the comparison among the baseline and the three variants of the proposed approach, *pseudo*, *avg_ens* and *ps_ens*, in terms of Accuracy, AUC, and F-measure performance metrics for the *PolitiFact* and the *GossipCop* datasets, respectively.

Tables 13 and 14 show the performance improvements in terms of percentage increment for the different metrics of the proposed self-trained models with respect to the baseline, when the *PolitiFact* and the *GossipCop* datasets are tested, respectively.

The increment performance results in tables 13 and 14 allow for a better

understanding of the behaviour of the different proposed strategies when the percentages of available labelled data vary.

Results highlight that all the proposed pseudo-labelling strategies obtain a substantial increment for all the metrics considered and for both datasets. The improvements are more evident for the *PolitiFact* dataset, which is characterised by a smaller number of samples (814), probably because the proposed self-training strategy is more efficient when the labelled data are very scarce. The other interesting discovery, more evident for the *PolitiFact* dataset, is that the proposed pseudo-labelling based self-trained approaches perform very well, with a very high-performance increment, for very low percentages of labelled tuples. In fact, when only 2.5% of data are labelled, the *avg_ens* is able to reach an improvement of the performances with respect to the baseline of 31.91%, 34.33% and 25.17% for the *PolitiFact* dataset and of 15.54%, 15.50% and 13.50% for the *GossipCop* dataset, relatively to accuracy, AUC and F1, respectively.

The increment of performances when using the proposed self-training solutions decreases when increasing the percentage of available labelled data. In fact, it is supposed that when the labelled data increase, the original BERT model can exploit enough labelled tuples in order to be trained for reaching high enough performances, thus confirming that using true, and therefore all correct, labels are always preferable to the use of pseudo, and therefore artificial and not always correct, labels, unless the true labelled data are not enough to guarantee a good training of the model.

In order to highlight how the introduction of the ensemble strategies affects the performance of the self-trained pseudo-labelling proposed solution, Tables 15 and 16 are here reported. These tables, together with Figures 35 and 36, introduced for better visual comprehension, show, for the different metrics, the performance improvements in terms of percentage increment of the proposed self-trained ensemble models (*avg_ens* and *ps_ens*) with respect to the same strategy but without ensemble (*pseudo* which selects just the last trained model), for the *PolitiFact* and the *GossipCop* datasets, respectively. The use of the ensemble techniques still is confirmed to be the preferable choice, as it is possible to notice mainly in the *PolitiFact* dataset where the *avg_ens* is able to improve the pseudo-labelling solution without the ensemble of maximum 6.33%, 5.59% and 9.33% when only 2.5% of labelled data are available for the accuracy, AUC and F1 metrics, respectively. In the *GossipCop* dataset, the use of the ensemble furnishes an almost constant performance increment that remains in the interval between 0.93% and 1.54% for the accuracy metric, between 0.71% and 1.26% for the AUC metric, and between 1.07% and 1.89% for the F1 metric.

A further consideration can be given by comparing the two proposed ensemble solutions. It is worth recalling that the *avg_ens* uses the accuracy of the model as weight thus considering the “reliability” of the model calculated for the validation set, while the *ps_ens* uses the average probability of the generated pseudo labels thus considering the “reliability” of the labels used during the training phase. From the experimental results, it is possible to conclude that, even if the two ensembles differ very much in the techniques of weighing the base models, substantially, the performances are very similar to each other, with a slightly better average performance improvement offered by the *avg_ens* solution.

Table 11: Comparison of the pseudo-labelling strategies for the *PolitiFact* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	Baseline	0.58 ± 0.09	0.65 ± 0.08	0.75 ± 0.05	0.79 ± 0.05
	<i>pseudo</i>	0.72 ± 0.08	0.74 ± 0.08	0.80 ± 0.04	0.85 ± 0.04
	<i>avg_ens</i>	0.77 ± 0.06	0.77 ± 0.05	0.83 ± 0.05	0.87 ± 0.03
	<i>ps_ens</i>	0.76 ± 0.04	0.76 ± 0.05	0.83 ± 0.04	0.86 ± 0.03
AUC	Baseline	0.57 ± 0.08	0.65 ± 0.08	0.74 ± 0.05	0.78 ± 0.05
	<i>pseudo</i>	0.72 ± 0.07	0.73 ± 0.09	0.80 ± 0.06	0.85 ± 0.04
	<i>avg_ens</i>	0.77 ± 0.06	0.76 ± 0.05	0.83 ± 0.05	0.87 ± 0.03
	<i>ps_ens</i>	0.76 ± 0.04	0.76 ± 0.05	0.82 ± 0.04	0.86 ± 0.03
F1	Baseline	0.62 ± 0.17	0.68 ± 0.12	0.77 ± 0.04	0.81 ± 0.05
	<i>pseudo</i>	0.72 ± 0.12	0.77 ± 0.06	0.81 ± 0.09	0.86 ± 0.04
	<i>avg_ens</i>	0.78 ± 0.08	0.79 ± 0.06	0.85 ± 0.04	0.88 ± 0.03
	<i>ps_ens</i>	0.78 ± 0.06	0.79 ± 0.06	0.84 ± 0.04	0.88 ± 0.03

Table 12: Comparison of the pseudo-labelling strategies for the *GossipCop* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5 %, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	Baseline	0.61 ± 0.04	0.66 ± 0.03	0.68 ± 0.02	0.70 ± 0.03
	<i>pseudo</i>	0.69 ± 0.03	0.72 ± 0.02	0.75 ± 0.02	0.76 ± 0.02
	<i>avg_ens</i>	0.71 ± 0.02	0.73 ± 0.01	0.76 ± 0.02	0.77 ± 0.01
	<i>ps_ens</i>	0.71 ± 0.03	0.73 ± 0.01	0.76 ± 0.02	0.77 ± 0.02
AUC	Baseline	0.60 ± 0.04	0.66 ± 0.02	0.68 ± 0.02	0.70 ± 0.03
	<i>pseudo</i>	0.69 ± 0.02	0.72 ± 0.02	0.74 ± 0.02	0.75 ± 0.02
	<i>avg_ens</i>	0.70 ± 0.02	0.72 ± 0.01	0.75 ± 0.02	0.76 ± 0.02
	<i>ps_ens</i>	0.70 ± 0.02	0.72 ± 0.01	0.75 ± 0.02	0.76 ± 0.02
F1	Baseline	0.65 ± 0.05	0.69 ± 0.04	0.71 ± 0.03	0.72 ± 0.05
	<i>pseudo</i>	0.73 ± 0.05	0.75 ± 0.03	0.78 ± 0.01	0.78 ± 0.02
	<i>avg_ens</i>	0.74 ± 0.04	0.76 ± 0.02	0.79 ± 0.01	0.79 ± 0.01
	<i>ps_ens</i>	0.74 ± 0.05	0.76 ± 0.02	0.79 ± 0.01	0.79 ± 0.01

Table 13: Delta increment of the pseudo-labelling strategies in comparison with the baseline for the *PolitiFact* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	<i>pseudo</i>	24.05%	13.28%	7.14%	7.51%
	<i>avg_ens</i>	31.90%	17.44%	11.61%	10.25%
	<i>ps_ens</i>	31.14%	16.76%	10.78%	9.47%
AUC	<i>pseudo</i>	27.22%	13.32%	7.65%	7.95%
	<i>avg_ens</i>	34.33%	17.58%	11.83%	10.58%
	<i>ps_ens</i>	33.44%	16.90%	11.02%	9.84%
F1	<i>pseudo</i>	14.49%	13.03%	4.52%	6.02%
	<i>avg_ens</i>	25.17%	16.04%	9.87%	8.68%
	<i>ps_ens</i>	25.08%	15.46%	9.08%	7.95%

Table 14: Delta increment of the pseudo-labelling strategies in comparison with the baseline for the pseudo-labelling strategies for the *GossipCop* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	<i>pseudo</i>	13.79%	9.42%	9.88%	8.72%
	<i>avg_ens</i>	15.54%	10.43%	11.09%	10.09%
	<i>ps_ens</i>	15.46%	10.54%	11.04%	9.98%
AUC	<i>pseudo</i>	14.06%	9.08%	8.91%	8.3%
	<i>avg_ens</i>	15.50%	9.84%	10.01%	9.53%
	<i>ps_ens</i>	15.45%	9.90%	9.95%	9.45%
F1	<i>pseudo</i>	11.39%	9.23%	10.88%	9.24%
	<i>avg_ens</i>	13.50%	10.74%	12.08%	10.83%
	<i>ps_ens</i>	13.37%	10.87%	12.07%	10.72%

Table 15: Delta increment of the ensemble pseudo-labelling strategies in comparison with the pseudo-labelling solution without ensemble for the *PolitiFact* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	<i>avg_ens</i>	6.33%	3.67%	4.18%	2.54%
	<i>ps_ens</i>	5.72%	3.08%	3.41%	1.83%
AUC	<i>avg_ens</i>	5.59%	3.76%	3.88%	2.43%
	<i>ps_ens</i>	4.88%	3.16%	3.13%	1.75%
F1	<i>avg_ens</i>	9.33%	2.67%	5.11%	2.50%
	<i>ps_ens</i>	9.24%	2.15%	4.36%	1.82%

Table 16: Delta increment of the ensemble pseudo-labelling strategies in comparison with the pseudo-labelling solution without ensemble for the *GossipCop* dataset: Accuracy, AUC and F-measure for different percentages of the training set (2.5%, 5%, 10% and 20%).

Metric	Algorithm	2.5%	5%	10%	20%
Accuracy	<i>avg_ens</i>	1.54%	0.93%	1.11%	1.28%
	<i>ps_ens</i>	1.47%	1.02%	1.06%	1.17%
AUC	<i>avg_ens</i>	1.26%	0.71%	1.01%	1.14%
	<i>ps_ens</i>	1.22%	0.77%	0.96%	1.06%
F1	<i>avg_ens</i>	1.89%	1.38%	1.08%	1.46%
	<i>ps_ens</i>	1.78%	1.5%	1.07%	1.35%

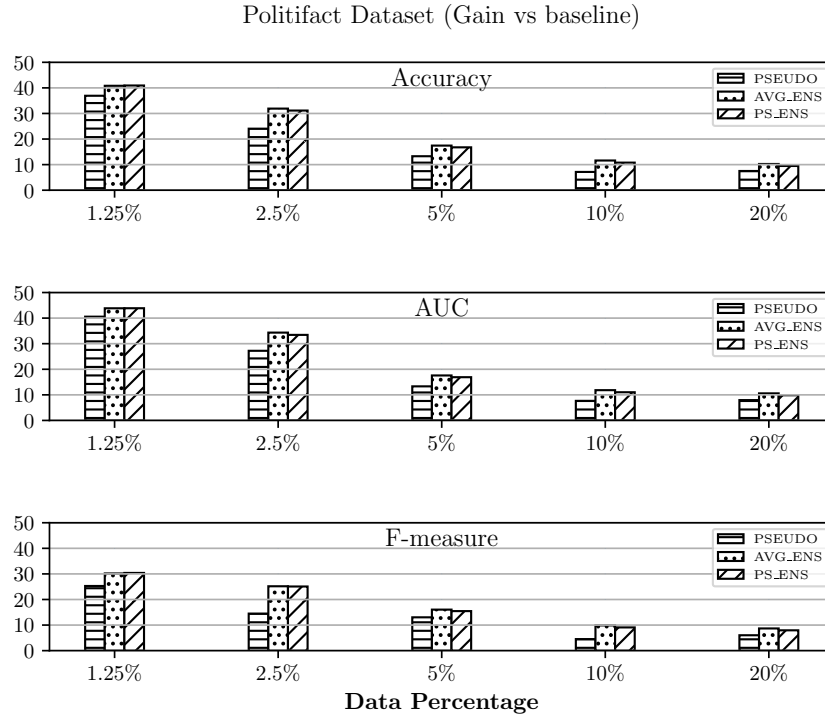


Figure 35: Delta increment of the different strategies in comparison with the baseline solution for the *PolitiFact* dataset: Accuracy, AUC and F-measure for different percentages of the training set (1.25%, 2.5%, 5%, 10%, and 20%)

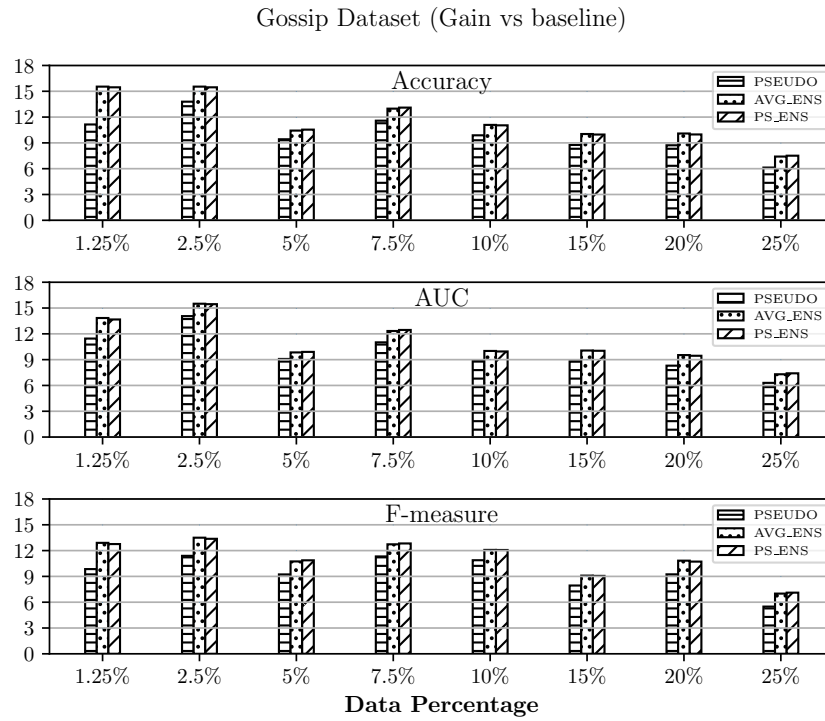


Figure 36: Delta increment of the different strategies in comparison with the baseline solution for the *Gossip* dataset: Accuracy, AUC and F-measure for different percentages of the training set (1.25%, 2.5%, 5%, 7.5%, 10%, 15%, 20% and 25%)

7 Conclusions and future works

In this thesis, a general framework architecture for the text classification problem based on the ensemble paradigm is proposed.

The proposed deep ensemble classification framework integrates different types of DNN architectures as base classifiers of the ensemble model in order to increase the levels of expressivity and promote diversity. Although the idea of combining heterogeneous models is not new itself, a novel idea was introduced by including a high-level representation of the features that is exploited for a dual purpose, to provide more accurate classifications when extending two state-of-the-art ensemble combination strategies and to create a “latent space” representation of ticket texts to be used together with a similarity metric and neighbour space search strategies in the explanation task. Based on the concept of combining the classifier model trained at different steps of the self-training process in the presence of scarce labelled data, a novel ensemble-based strategy of pseudo-labelling was proposed.

The proposed framework was designed in order to furnish a valid instrument for the study and analysis of the state of the art in the research fields of NLP, ML and Ensemble of DNNs and explanation; the study of the specific context peculiarities of the automatic ticket classification and the automatic fake detection; the design of novel classification and explanation EDN models; the proposal of a novel pseudo labelling strategy for improving classifiers in the presence of a small number of labelled data; the implementation of a complete framework for the entire text processing flow for performance tests.

Several innovative research contributions have been proposed through this framework that can be synthesized as in the following. Two novel ensemble deep neural network models have been introduced. An imbalance-aware training loss strategy has been adopted to improve performances when imbalance class issues occur in ticket classification. A novel human-in-the-loop explanation scheme for intelligent text classification has been introduced for providing two kinds of artefacts, LIME (Locally Interpretable Model agnostic Explanations) based on local explanations capturing the importance of text words in the class predictions, and local word clouds, summarizing the contents of example texts looking most similar to the predicted text instance in the latent space learnt by the HLFE sub-net of the model. A novel ensemble based pseudo-labelling strategy has been proposed in order to improve the performances of the classifier in case of scarcity of labelled data.

The solutions proposed by this thesis represent a general framework based

on an ensemble paradigm designed for the entire process of text classification. From the general structure, the framework has been customized for two application contexts used for validating the proposed strategies in real-life scenarios: ticket classification for customer support and fake detection.

Experiments conducted on real data confirmed the accuracy of the ensemble classifiers discovered with this framework and the usefulness of the explanation artefacts. The experimental results proved that the ability to provide both accurate predictions along with their interpretable explanations is a valuable line of research with a high potential impact on the quality of processes and services in all the application contexts, as demonstrated for the intelligent Ticket Management Systems (TMSs). This ability enables an effective scheme of cooperation between the system and human users with mutual benefits: on the one hand, predictions and explanations allow users to make decisions in a more conscious and risk-aware fashion; on the other hand, human feedback can help detect wrong predictions (and prevent wrong decisions), as well as to trigger the revision/re-discovery of biased/inaccurate classifiers. The experimental tests also demonstrated the effectiveness of the proposed innovative strategy of using the ensemble idea for increasing performance in the pseudo-labelling self-training process. The proposed strategy is a valid instrument to reduce the costs and difficulties of labelling, a problem that has become increasingly significant with the huge amount of data every day generated on the web, like for the fake detection problem.

As concerns future work, several possible topics that would be interesting to explore are here identified as an extension of the proposed framework: (i) *Deep Active Learning* mechanisms [40], in order to allow human experts to provide new informative example tickets that can help improve the predictive performance of a ticket classifier (especially in regions of the instance space where it is performing worse or it is more uncertain), (ii) visual explanation paradigms (e.g., based on saliency/attribution maps [62, 70] and, in particular, attention mechanisms for recurrent NNs [67]), complementing the interpretable explanations provided by the local surrogate models discovered with LIME; and (iii) more expressive methods for estimating the epistemic uncertainty affecting each prediction returned by an ensemble classifier (using measures of disagreement over the predictions of the base classifiers, plus uncertainty for the combiner, possibly obtained with Monte Carlo dropout methods [1]).

References

- [1] Moloud Abdar et al. “A review of uncertainty quantification in Deep Learning: techniques, applications and challenges”. In: *Information Fusion* 76 (2021), pp. 243–297.
- [2] M. Altintas and A. Tantug. “MACHINE LEARNING BASED TICKET CLASSIFICATION IN ISSUE TRACKING SYSTEMS”. In: *AICS e-Journal of Artificial Intelligence and Computer Science* 2 (2014), pp. 33–44.
- [3] Mucahit Altintas and Ahmet Cuneyd Tantug. “Machine Learning Based Ticket Classification in Issue Tracking Systems”. In: *Journal of King Saud University - Computer and Information Sciences* (2014).
- [4] Eric Arazo et al. “Pseudo-Labeling and Confirmation Bias in Deep Semi-Supervised Learning”. In: *2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020*. IEEE, 2020, pp. 1–8. DOI: [10.1109/IJCNN48605.2020.9207304](https://doi.org/10.1109/IJCNN48605.2020.9207304). URL: <https://doi.org/10.1109/IJCNN48605.2020.9207304>.
- [5] Adrien Benamira et al. “Semi-supervised learning and graph neural networks for fake news detection”. In: *ASONAM '19: International Conference on Advances in Social Networks Analysis and Mining, Vancouver, British Columbia, Canada, 27-30 August, 2019*. Ed. by Francesca Spezzano, Wei Chen, and Xiaokui Xiao. ACM, 2019, pp. 568–569. DOI: [10.1145/3341161.3342958](https://doi.org/10.1145/3341161.3342958). URL: <https://doi.org/10.1145/3341161.3342958>.
- [6] Alessandro Bondielli and Francesco Marcelloni. “A Survey on Fake News and Rumour Detection Techniques”. In: *Information Sciences* 497 (May 2019). DOI: [10.1016/j.ins.2019.05.035](https://doi.org/10.1016/j.ins.2019.05.035).
- [7] L Breiman. “Random Forests”. In: *Machine Learning* 45 (Oct. 2001), pp. 5–32. DOI: [10.1023/A:1010950718922](https://doi.org/10.1023/A:1010950718922).
- [8] L. Breiman. “Bagging predictors”. In: *Machine Learning* 24 (2004), pp. 123–140.
- [9] Krisztian Buza and Aleksandra Revina. “Speeding up the SUCCESS Approach for Massive Industrial Datasets”. In: *2020 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*. 2020, pp. 1–6. DOI: [10.1109/INISTA49547.2020.9194656](https://doi.org/10.1109/INISTA49547.2020.9194656).

- [10] Paola Cascante-Bonilla et al. “Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 8. 2021, pp. 6912–6920.
- [11] Hong-Gunn Chew, Robert E Bogner, and Cheng-Chew Lim. “Dual *nu*-support vector machine with error rate and training size biasing”. In: *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing*. Vol. 2. IEEE. 2001, pp. 1269–1272.
- [12] Kyunghyun Cho et al. *On the Properties of Neural Machine Translation: Encoder-Decoder Approaches*. 2014. DOI: [10.48550/ARXIV.1409.1259](https://arxiv.org/abs/1409.1259). URL: <https://arxiv.org/abs/1409.1259>.
- [13] Janez Demsar. “Statistical Comparisons of Classifiers over Multiple Data Sets”. In: *Journal of Machine Learning Research* 7 (2006), pp. 1–30.
- [14] J. Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *NAACL-HLT*. Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: [10.18653/v1/N19-1423](https://arxiv.org/abs/1910.1423).
- [15] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: *Multiple Classifier Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–15. ISBN: 978-3-540-45014-6.
- [16] Xishuang Dong et al. “Deep Two-path Semi-supervised Learning for Fake News Detection”. In: *CoRR* abs/1906.05659 (2019). arXiv: [1906.05659](http://arxiv.org/abs/1906.05659). URL: <http://arxiv.org/abs/1906.05659>.
- [17] Stijn van Dongen and Anton J. Enright. “Metric distances derived from cosine similarity and Pearson and Spearman correlations”. In: *CoRR* abs/1208.3145 (2012). URL: <http://arxiv.org/abs/1208.3145>.
- [18] Maha Elbayad, Laurent Besacier, and Jakob Verbeek. “Pervasive Attention: 2D Convolutional Neural Networks for Sequence-to-Sequence Prediction”. In: *CoNLL*. 2018.
- [19] Yoav Freund and Robert E. Schapire. “Experiments with a New Boosting Algorithm”. In: *International Conference on Machine Learning*. 1996.
- [20] Jerome Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *The Annals of Statistics* 29 (Nov. 2000). DOI: [10.1214/aos/1013203451](https://arxiv.org/abs/1013.2034).

- [21] Yarin Gal and Zoubin Ghahramani. “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *International Conference on Machine Learning*. PMLR. 2016, pp. 1050–1059.
- [22] Salvador García and Francisco Herrera. “An Extension on “Statistical Comparisons of Classifiers over Multiple Data Sets” for all Pairwise Comparisons”. In: *Journal of Machine Learning Research* 9 (2009), pp. 2677–2694.
- [23] Jonas Gehring et al. “Convolutional Sequence to Sequence Learning”. In: ICML’17. Sydney, NSW, Australia: JMLR.org, 2017, pp. 1243–1252.
- [24] Gisel Bastidas Guacho et al. “Semi-Supervised Content-Based Detection of Misinformation via Tensor Embeddings”. In: *Proceedings of the 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. ASONAM ’18. Barcelona, Spain: IEEE Press, 2020, pp. 322–325. ISBN: 9781538660515.
- [25] L.K. Hansen and P. Salamon. “Neural network ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990), pp. 993–1001. DOI: [10.1109/34.58871](https://doi.org/10.1109/34.58871).
- [26] Feras Al-Hawari and Hala Barham. “A machine learning based help desk system for IT service management”. In: *Journal of King Saud University - Computer and Information Sciences* 33.6 (2021), pp. 702–718.
- [27] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (Nov. 1997), pp. 1735–1780. ISSN: 0899-7667. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735), URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [28] Linmei Hu et al. “Deep learning for fake news detection: A comprehensive survey”. In: *AI Open* 3 (2022), pp. 133–155. ISSN: 2666-6510. DOI: <https://doi.org/10.1016/j.aiopen.2022.09.001>, URL: <https://www.sciencedirect.com/science/article/pii/S2666651022000134>.
- [29] Ahmet Iscen et al. “Label Propagation for Deep Semi-Supervised Learning”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5065–5074. DOI: [10.1109/CVPR.2019.00521](https://doi.org/10.1109/CVPR.2019.00521).

- [30] Robert A. Jacobs et al. “Adaptive Mixtures of Local Experts”. In: *Neural Computation* 3.1 (1991), pp. 79–87. DOI: [10.1162/neco.1991.3.1.79](https://doi.org/10.1162/neco.1991.3.1.79).
- [31] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A Convolutional Neural Network for Modelling Sentences”. In: *ArXiv abs/1404.2188* (2014).
- [32] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *Conference on Empirical Methods in Natural Language Processing*. 2014.
- [33] Samuli Laine and Timo Aila. “Temporal ensembling for semi-supervised learning”. In: *arXiv preprint arXiv:1610.02242* (2016).
- [34] Dong-Hyun Lee. “Pseudo-Label : The Simple and Efficient Semi-Supervised Learning Method for Deep Neural Networks”. In: *ICML 2013 Workshop : Challenges in Representation Learning (WREPL)* (July 2013).
- [35] Qian Li et al. “A Survey on Text Classification: From Traditional to Deep Learning”. In: *ACM Trans. Intell. Syst. Technol.* 13.2 (2022), 31:1–31:41. DOI: [10.1145/3495162](https://doi.org/10.1145/3495162), URL: <https://doi.org/10.1145/3495162>.
- [36] Min Lin, Qiang Chen, and Shuicheng Yan. *Network In Network*. 2014. arXiv: [1312.4400 \[cs.NE\]](https://arxiv.org/abs/1312.4400).
- [37] Ting Liu et al. “New algorithms for efficient high-dimensional non-parametric classification.” In: *Journal of Machine Learning Research* 7.6 (2006).
- [38] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. USA: Cambridge University Press, 2008. ISBN: 0521865719.
- [39] S. Masoudnia and R. Ebrahimpour. “Mixture of experts: a literature survey”. In: *Artificial Intelligence Review* 42.2 (2014), pp. 275–293.
- [40] Kayo Matsushita, Kayo Matsushita, and Hasebe. *Deep active learning*. Springer, 2018.
- [41] Priyanka Meel and Dinesh Kumar Vishwakarma. “A temporal ensembling based semi-supervised ConvNet for the detection of fake news articles”. In: *Expert Systems with Applications* 177 (2021), p. 115002. ISSN: 0957-4174. DOI: <https://doi.org/10.1016/j.eswa.2021.115002>, URL: <https://www.sciencedirect.com/science/article/pii/S0957417421004437>.

- [42] Priyanka Meel and Dinesh Kumar Vishwakarma. “Fake News Detection using Semi-Supervised Graph Convolutional Network”. In: *CoRR* abs/2109.13476 (2021). arXiv: [2109.13476](https://arxiv.org/abs/2109.13476). URL: <https://arxiv.org/abs/2109.13476>.
- [43] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *Advances in Neural Information Processing Systems* 26 (Oct. 2013).
- [44] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *Proceedings of Workshop at ICLR 2013* (Jan. 2013).
- [45] Piero Molino, Huaixiu Zheng, and Yi-Chia Wang. “COTA: Improving the Speed and Accuracy of Customer Support through Ranking and Deep Networks”. In: *KDD '18: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2018, pp. 586–595.
- [46] S.P. Paramesh, C. Ramya, and K.S. Shreedhara. “Classifying the Unstructured IT Service Desk Tickets Using Ensemble of Classifiers”. In: *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*. 2018, pp. 221–227. DOI: [10.1109/CSITSS.2018.8768734](https://doi.org/10.1109/CSITSS.2018.8768734).
- [47] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [48] Jeffrey Pennington, Richard Socher, and Christopher Manning. “Glove: Global Vectors for Word Representation”. In: vol. 14. Jan. 2014, pp. 1532–1543. DOI: [10.3115/v1/D14-1162](https://doi.org/10.3115/v1/D14-1162).
- [49] Ruanda Qamili, Shaban Shabani, and Johannes Schneider. “An Intelligent Framework for Issue Ticketing System Based on Machine Learning”. In: *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*. Oct. 2018, pp. 79–86.
- [50] Pranav Rajpurkar, Robin Jia, and Percy Liang. “Know What You Don’t Know: Unanswerable Questions for SQuAD”. In: Jan. 2018, pp. 784–789. DOI: [10.18653/v1/P18-2124](https://doi.org/10.18653/v1/P18-2124).
- [51] Jason D. M. Rennie et al. “Tackling the Poor Assumptions of Naive Bayes Text Classifiers”. In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML’03. Washington, DC, USA: AAAI Press, 2003, pp. 616–623. ISBN: 1577351894.

- [52] Aleksandra Revina, Krisztian Buza, and Vera G. Meister. “Designing Explainable Text Classification Pipelines: Insights from IT Ticket Complexity Prediction Case Study”. In: *Interpretable Artificial Intelligence: A Perspective of Granular Computing*. Ed. by Witold Pedrycz and Shyi-Ming Chen. Cham: Springer International Publishing, 2021, pp. 293–332. ISBN: 978-3-030-64949-4. DOI: [10.1007/978-3-030-64949-4_10](https://doi.org/10.1007/978-3-030-64949-4_10). URL: https://doi.org/10.1007/978-3-030-64949-4_10.
- [53] Aleksandra Revina, Krisztian Buza, and Vera G. Meister. “IT Ticket Classification: The Simpler, the Better”. In: *IEEE Access* 8 (2020), pp. 193380–193395. DOI: [10.1109/ACCESS.2020.3032840](https://doi.org/10.1109/ACCESS.2020.3032840).
- [54] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier”. In: *KDD ’16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 1135–1144.
- [55] Jitendra Kumar Rout et al. “Revisiting Semi-Supervised Learning for Online Deceptive Review Detection”. In: *IEEE Access* 5 (2017), pp. 1319–1327. DOI: [10.1109/ACCESS.2017.2655032](https://doi.org/10.1109/ACCESS.2017.2655032).
- [56] Atefeh Shahroudnejad. “A Survey on Understanding, Visualizations, and Explanation of Deep Neural Networks”. In: *ArXiv abs/2102.01792* (2021).
- [57] Kai Shu et al. “Fake News Detection on Social Media: A Data Mining Perspective”. In: *ACM SIGKDD Explorations Newsletter* 19.1 (2017), pp. 22–36.
- [58] Kai Shu et al. “FakeNewsNet: A Data Repository with News Content, Social Context and Dynamic Information for Studying Fake News on Social Media”. In: *arXiv preprint arXiv:1809.01286* (2018).
- [59] S. Silva, R. Pereira, and R. Ribeiro. “Machine learning in incident categorization automation”. In: *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)* (2018), pp. 1–6.
- [60] Gwang Son, Victor Hazlewood, and Gregory Dean Peterson. “On Automating XSEDE User Ticket Classification”. In: *XSEDE ’14: Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment*. Atlanta, GA, USA: Association for Computing Machinery, 2014, pp. 1–7. ISBN: 9781450328937.

- [61] Sanskar Soni, Satyendra Singh Chouhan, and Santosh Singh Rathore. “TextConvoNet: a convolutional neural network based architecture for text classification”. In: *Applied Intelligence (Dordrecht, Netherlands)* (2022), pp. 1–20.
- [62] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. “Axiomatic Attribution for Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. JMLR.org, 2017, pp. 3319–3328.
- [63] Erik F. Tjong Kim Sang and Fien De Meulder. “Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition”. In: *Proceedings of CoNLL-2003*. Ed. by Walter Daelemans and Miles Osborne. Edmonton, Canada, 2003, pp. 142–147.
- [64] Alper Kursat Uysal and Serkan Gunal. “The impact of preprocessing on text classification”. In: *Information processing & management* 50.1 (2014), pp. 104–112.
- [65] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.
- [66] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Long Beach, California, USA, 2017, pp. 6000–6010.
- [67] Sarah Wiegrefe and Yuval Pinter. “Attention is not not explanation”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 11–20.
- [68] Adina Williams, Nikita Nangia, and Samuel Bowman. “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference”. In: Jan. 2018, pp. 1112–1122. DOI: [10.18653/v1/N18-1101](https://doi.org/10.18653/v1/N18-1101).
- [69] Feixue Yan et al. “Explainable machine learning in cybersecurity: A survey”. In: *International Journal of Intelligent Systems* 37 (Nov. 2022). DOI: [10.1002/int.23088](https://doi.org/10.1002/int.23088).
- [70] Hao Yuan et al. “Interpreting deep models for text analysis via optimization and regularization methods”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (2019), pp. 5717–5724.

- [71] Hua Yuan et al. “Improving fake news detection with domain-adversarial and graph-attention neural network”. In: *Decision Support Systems* 151 (2021), p. 113633. ISSN: 0167-9236. DOI: <https://doi.org/10.1016/j.dss.2021.113633> URL: <https://www.sciencedirect.com/science/article/pii/S0167923621001433>.
- [72] Xichen Zhang and Ali A. Ghorbani. “An overview of online fake news: Characterization, detection, and discussion”. In: *Information Processing & Management* 57.2 (2020), p. 102025. ISSN: 0306-4573.
- [73] Junmei Zhong and William Li. “Predicting Customer Call Intent by Analyzing Phone Call Transcripts Based on CNN for Multi-Class Classification”. In: *8th International Conference on Soft Computing, Artificial Intelligence and Applications* (2019), pp. 13–25.
- [74] Wubai Zhou et al. “STAR: A System for Ticket Analysis and Resolution”. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017), pp. 2181–2190.
- [75] Xinyi Zhou and Reza Zafarani. “A survey of fake news: Fundamental theories, detection methods, and opportunities”. In: *ACM Computing Surveys (CSUR)* 53.5 (2020), pp. 1–40.