Expert Systems With Applications 225 (2023) 119984

Contents lists available at ScienceDirect

Expert Systems With Applications

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

Review

Ticket automation: An insight into current research with applications to multi-level classification scenarios

Alessandro Zangari[a,1], Matteo Marcuzzo[a,*,1], Michele Schiavinato[a,1], Andrea Gasparetto[b], Andrea Albarelli[a]

[a] Ca' Foscari Department of Environmental Sciences, Informatics and Statistics, Via Torino 155, Mestre (VE), 30172, Italy
[b] Ca' Foscari Department of Management, Cannaregio 873, Fondamenta San Giobbe, Venice, 30121, Italy

ARTICLE INFO

Keywords:
Ticket automation
Natural language processing (NLP)
Multi-level classification
Helpdesk
Support ticket
Word embeddings

ABSTRACT

Modern service providers often have to deal with large amounts of customer requests, which they need to act upon in a swift and effective manner to ensure adequate support is provided. In this context, machine learning algorithms are fundamental in streamlining support ticket processing workflows. However, a large part of current approaches is still based on traditional Natural Language Processing approaches without fully exploiting the latest advancements in this field. In this work, we aim to provide an overview of support Ticket Automation, what recent proposals are being made in this field, and how well some of these methods can generalize to new scenarios and datasets. We list the most recent proposals for these tasks and examine in detail the ones related to Ticket Classification, the most prevalent of them. We analyze commonly utilized datasets and experiment on two of them, both characterized by a two-level hierarchy of labels, which are descriptive of the ticket's topic at different levels of granularity. The first is a collection of 20,000 customer complaints, and the second comprises 35,000 issues crawled from a bug reporting website. Using this data, we focus on topically classifying tickets using a pre-trained BERT language model. The experimental section of this work has two objectives. First, we demonstrate the impact of different document representation strategies on classification performance. Secondly, we showcase an effective way to boost classification by injecting information from the hierarchical structure of the labels into the classifier. Our findings show that the choice of the embedding strategy for ticket embeddings considerably impacts classification metrics on our datasets: the best method improves by more than 28% in F_1-score over the standard strategy. We also showcase the effectiveness of hierarchical information injection, which further improves the results. In the bugs dataset, one of our multi-level models (`ML-BERT`) outperforms the best baseline by up to 5.7% in F_1-score and 5.4% in accuracy.

1. Introduction

The term *support ticket* describes a request for help from a customer to a service provider's support team. These include service tickets, customer complaints, and incident reports, and are fundamental tools for any modern company when it comes to managing their relationship with customers (Al-Hawari & Barham, 2021). Tickets represent the most valuable point of contact between the users and the staff responsible for the management of a service, allowing for the resolution of any issue or incident related to it. These types of interactions are ubiquitous across practically any industry field. Though the most common examples include IT-related support requests and bug reports (Mani, Sankaran, & Aralikatte, 2019), these services are also used in domains such as healthcare (Young, Luz, & Lone, 2019) and governmental institutions (Powell, Rotz, & O'Malley, 2020).

What we refer to with the generic term of "tickets" are most commonly messages presented in textual form, often written directly by customers or technicians, therefore comprising mainly of natural language (though it should be mentioned that it is also common for them to be created automatically by a computational agent in response to a fault or bug). The most prominent channels from which tickets originate include emails, phone calls, specialized web forms, live chats, and, as of late, social media platforms (Zicari, Folino, Guarascio, & Pontieri, 2021). They are most frequently composed of a short title

* Corresponding author.
E-mail addresses: alessandro.zangari@unive.it (A. Zangari), matteo.marcuzzo@unive.it (M. Marcuzzo), michele.schiavinato@unive.it (M. Schiavinato), andrea.gasparetto@unive.it (A. Gasparetto), albarelli@unive.it (A. Albarelli).
[1] Authors contributed equally to this work.

https://doi.org/10.1016/j.eswa.2023.119984
Received 20 September 2022; Received in revised form 14 March 2023; Accepted 25 March 2023
Available online 6 April 2023

and a description that recounts the issue or request by the client and are usually very noisy and concise. In some cases, along with the textual help request, tickets may contain additional context data (*e.g.*, a screenshot) (Mandal, Agarwal et al., 2019).

When a ticket is produced, its categorization and routing to resolving experts are tasks of the utmost importance. A swift resolution ensures customer satisfaction, high productivity, and compliance with Service-Level Agreements (SLAs) — which often dictate that issues be solved within a specific time frame (Gupta & Sengupta, 2012). Conversely, improper routing of tickets may result in wasteful reassignment and unnecessary resource utilization, with adverse financial consequences for customers and service providers both (Paramesh, Ramya, & Shreedhara, 2018; Paramesh & Shreedhara, 2019). In this context, *Ticket Automation* (TA) can be defined as the collection of automated systems that aim to reduce the number of steps between the submission of a ticket and its resolution.

Among TA tasks, accurately classifying incoming tickets with a descriptive label is among the most intuitive and widely studied, as well as being of particular importance to ensure that customers have their requests complied with rapidly. Indeed, as the volume of support tickets has significantly grown (especially in IT companies) (Ali Zaidi, Fraz, Shahzad, & Khan, 2022; Fuchs, Drieschner, & Wittges, 2022), the need for automated systems able to expedite the ticket resolution process has only become more prevalent.

1.1. Contributions

In this article, we will provide an overview of the TA landscape, exploring its most common sub-tasks and listing the most recent developments that have been applied to this field. Then, we will explore in more detail the most common framing of TA, *i.e.*, the automatic categorization of a service request within a shallow hierarchy of topics. The task is therefore that of text classification; as such, a preliminary step is that of learning semantically meaningful representations from the bodies of text of which support tickets are constituted. In this regard, we seek to explore the most recent developments in the field of Natural Language Processing (NLP) concerning text representation, mainly contextualized Language Models (LMs) (Devlin, Chang, Lee, & Toutanova, 2019; Radford, Narasimhan, Salimans, & Sutskever, 2018). These neural approaches based on the Transformer architecture (Vaswani et al., 2017) have obtained outstanding results in all NLP-related tasks and are now the *de-facto* standard approach to NLP transfer learning. As of now, new Transformer-based LMs are constantly being proposed, often with radical changes with respect to the original architecture. Nevertheless, the core attention-based foundation (Bahdanau, Cho, & Bengio, 2015; Luong, Pham, & Manning, 2015) remains the same. Still, despite their recent popularity in NLP applications, there is a lack of work that leverages these models for the classification of ticket-related datasets.

For the experimental section of this work, we will examine how a LM such as BERT (Devlin et al., 2019) – one of the most well-studied Transformer-based LMs – can be utilized in the context of support tickets. First, we explore different document embedding summarization strategies derived from its composing word embeddings. While a few works have addressed this topic in the past, we believe it would be interesting to showcase how different strategies behave in the ticket domain, which contains text that is by nature noisy and conversational. Then, as a second contribution, we devise a specialized multi-level model, able to extract hierarchical information by combining the embeddings of the documents as fine-tuned on different levels of the hierarchy. In this contribution, we tie notions derived from Hierarchical Text Classification (HTC) and apply them to the Ticket Classification (TC) environment. Finally, we compare the results with a set of baselines, including traditional approaches as well as more recent proposals.

The main contributions of this research can be summarized as follows:

- **Ticket automation overview** — We overview different approaches to TA and provide an analysis of recent contributions to this task. Moreover, we supply an up-to-date list of recent methods, framing them within four different TA tasks they aim to solve, as well as a comprehensive list of public datasets in the customer care domain;
- **Document embedding strategies** — We showcase how several strategies for producing document embeddings from a BERT LM can impact the model's performance on document-level classification;
- **Multi-level classification** — We propose a novel global approach to the TC sub-task, which exploits the hierarchical structure of the labels;

Our work is among the first to utilize a pre-trained Transformer-based LM for the classification of support tickets, which we demonstrate on two public datasets. Despite the noisy nature of the data, we show that these LMs can perform better than more traditional (*i.e.*, non-deep learning) methods, often still proposed in current literature. As such, we hope the insights provided in this work can help researchers to consider the usage of pre-trained LMs for industrial applications in the TA domain. We publicly share all the code and datasets used in our experiments.[2]

1.2. Structure of the article

The rest of the article is organized as follows. Section 2 provides a brief introduction to text representation concepts, fundamental to any approach that aims to solve a downstream task in NLP. In Section 3 we formalize the TA task, describing its similarities with HTC and describing the identified sub-tasks. We then present the results of our literature review about the applications of Machine Learning (ML) algorithms to the automation of ticket-related tasks, such as topic classification. We additionally include a list of public datasets suitable for research purposes, two of which are being leveraged in this work. Section 4 describes our contributions, which consist of the analysis of different summarization strategies for documents, as well as multiple multi-level classifiers for ticket categorization. Our experimental procedures are detailed in Section 5, along with the adopted metrics, preprocessing choices, and the baseline algorithms we implemented. This section also contains the results of our experiments, which are then discussed in detail and compared to other baselines in Section 6. Finally, Section 7 concludes our work and summarizes our main contributions and achievements.

2. Background: Text representation in NLP

A fundamental step for any machine learning algorithm that deals with text is its representation in a machine-digestible form. In this section, we provide a brief introduction to text representation techniques, highlighting both traditional approaches and the most recent advancements.

Text representation procedures have evolved enormously in recent years. These techniques have been revolutionized by the introduction of Deep Learning, allowing for semantically and syntactically meaningful embedding (*i.e.*, vectorial representations) of words and sentences. As we are interested in employing some of the latest language modeling techniques, we briefly introduce recent developments in deep learning methods for the representation of text, which constitute the major drive for improvement for text classification methods and NLP in general. This overview is necessarily superficial, and a much more in-depth exploration of the recent evolution of NLP and text classification procedures can be found in Gasparetto, Marcuzzo, Zangari, and Albarelli (2022), Kowsari et al. (2019), Li, Peng et al. (2020), Minaee et al. (2021).

[2] https://gitlab.com/distration/dsi-nlp-publib/-/tree/main/ticket-automation-survey-app-22

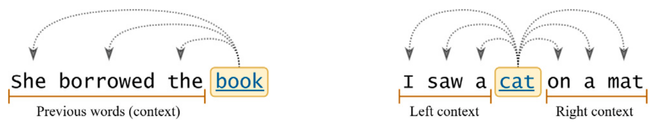


Fig. 1. Exemplification of the two most common objectives in language modeling tasks. The highlighted word is being predicted based on the surrounding (context) words.

2.1. *Weighted word counts*

Traditional methods for text classification are based on general-purpose classifiers, including methods such as Decision Trees (Ho, 1995; Safavian & Landgrebe, 1991), k-Nearest Neighbors (Cover & Hart, 1967; Li, Yu, & Lu, 2003), Probabilistic graphical models (Sutton & McCallum, 2012; Torsello, Gasparetto, Rossi, Bai, & Hancock, 2014; van den Bosch, 2017; Xu, Li, & Wang, 2017), and Support Vector Machines (Boser, Guyon, & Vapnik, 1992; Cortes & Vapnik, 1995).

These classifiers require a numerical input, thus necessitating text to be translated into some kind of vectorial form. At a very high level, text representation techniques practically always begin by indexing different words and creating a vocabulary by which words can be referenced by their index. Before the advent of neural approaches, bodies of text were then transformed into vectors by utilizing relatively simple statistical depictions, the most popularly used being that of Bag-of-Words (BoW). This technique essentially amounts to an unordered word count for vocabulary terms within a document. Most often, these counts are then weighted utilizing frequency terms based on word occurrence statistics, such as Term Frequency (TF) and Inverse Term Frequency (IDF) (Jones, 1972). Through these operations, a single, vectorized representation for each document can be obtained. However, these representations do not contain any real syntactic or semantic information — all sentence ordering information is lost, and the vectors do not encapsulate any real meaning of what they represent.

2.2. *Word embeddings*

A substantial turning point in text representation has been the development of *word embeddings*, a feature extraction technique able to learn semantically and syntactically meaningful vectorial representations of text. Seminal works such as Word2Vec (Mikolov, Chen, Corrado, & Dean, 2013; Mikolov, Sutskever, Chen, Corrado, & Dean, 2013) and GloVe (Pennington, Socher, & Manning, 2014) proposed language modeling approaches from which these embeddings could be extracted through shallow neural networks. The authors have since been able to prove that these vectors indeed encapsulate word meaning — for instance, these representations allow for vector arithmetic such as $\vec{king} - \vec{man} + \vec{woman} \approx \vec{queen}$, which showcase a deeper understanding of word semantics by the model. Moreover, these representations have since allowed for immense benefits on downstream task performances, such as with classification.

Word embedding models are based on the aforementioned concept of language modeling (Jurafsky & Martin, 2020). Language models themselves are probability distributions, usually obtained through a next-word prediction task, even though many possible variations have been proposed in practice. In the training process of embedding methods, models are usually tasked to infer a word given its context (*i.e.*, surrounding words). While originally LMs would only infer words given their left context (previous words), modern word embedding models most commonly utilize both left and right context (Fig. 1).

While much could be said about word embedding techniques, an influential approach worth mentioning is that of FastText (Bojanowski, Grave, Joulin, & Mikolov, 2017). Very briefly, the core difference between FastText embeddings and other earlier representations is the usage of character n-grams, *i.e.*, fragments of words. This way, word embeddings can be seen as a composition of multiple n-gram embeddings, which allows better generalization over rare or unknown words.

2.3. *Contextualized language models*

Word embeddings have been utilized in a broad range of approaches, both traditional and neural-based. For many tasks such as text classification, Recurrent Neural Networks (RNNs) (Sutskever, Vinyals, & Le, 2014) have long been the go-to model, as they are effective in dealing with sequentially structured information. RNNs have been widely utilized with word embeddings, both as classifiers that use them as input, and as part of the embedding training process. Convolutional Neural Networks (CNNs) (Gasparetto et al., 2018; Kim, 2014) have also been utilized, though to a lesser extent.

However, transfer learning in NLP – which word embeddings can be understood as – has had its second turning point in the development of *contextualized word embeddings*. Indeed, earlier word embeddings were unable to discern context, and therefore incapable of properly representing *polysemous words* (*i.e.*, words with multiple meanings). The introduction of context in these representations has allowed to solve this issue; this had been explored with RNNs with methods such as ELMO (Peters et al., 2018), but found greater success with the advent of the Transformer architecture (Vaswani et al., 2017). Among the advantages of Transformer-based architectures, which are entirely based on the *attention* mechanism (Bahdanau et al., 2015; Luong et al., 2015), stand out the capability for greater parallelism (because of the absence of recurrence), as well as favorable scaling with network depth (Kaplan et al., 2020). Indeed, a crucial advantage of Transformer-based approaches is that they scale very well in terms of performance with an increased number of parameters, which is most commonly achieved by adding more layers to the architecture (Bender, Gebru, McMillan-Major, & Shmitchell, 2021).

2.3.1. *Latest advancements*

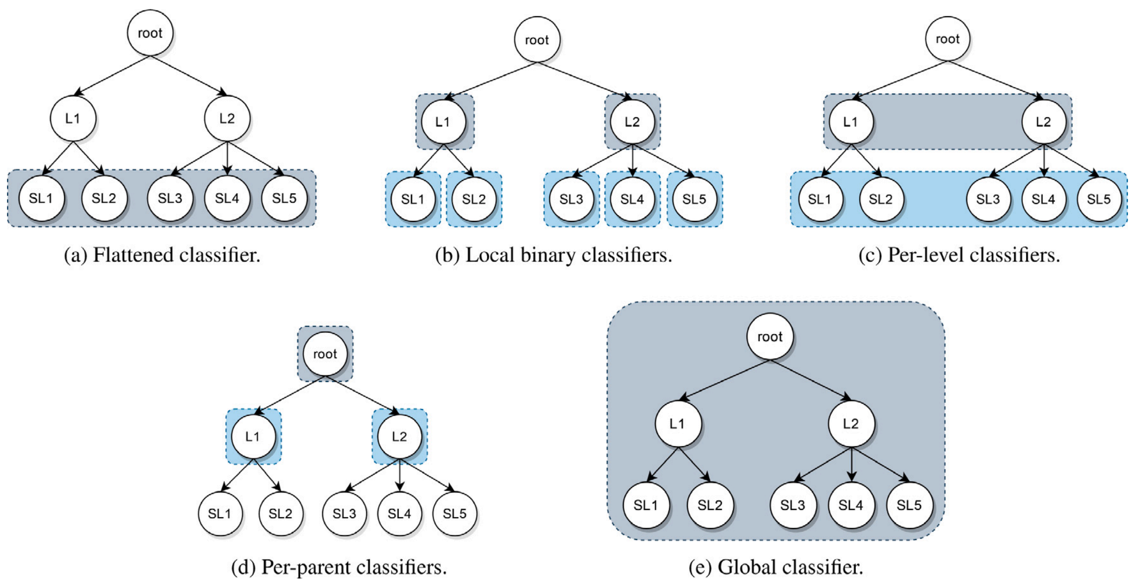
Many Transformer-based LMs have been devised since the original architecture was proposed by Vaswani et al. (2017). Among them, the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al., 2019) and Generative Pre-trained Transformer (GPT) (Radford et al., 2018) stand as notable examples because of their widespread use. Both have been revisited with numerous advances, such as RoBERTa (Liu et al., 2019), GPT-2 and GPT-3 (Brown et al., 2020; Radford et al., 2019), T5 (Raffel et al., 2020; Xue et al., 2022) and XLNet (Yang et al., 2019).

Currently, much of research's attention has been focused on *large LMs* (Carlini et al., 2021), which aim to exploit the high-performance scalability by training very deep networks on massive datasets, with examples such as GPT-3 (175 billion parameters) (Brown et al., 2020), GShard (600 billion parameters) (Lepikhin et al., 2021), and Switch-C (1.6 trillion parameters) (Fedus, Zoph, & Shazeer, 2022). To make a comparison, the widely studied BERT LM contains 345 million parameters in its largest iteration. Indeed, scale has been arguably a bigger factor than architectural changes in the latest proposals. For an in-depth overview of the Transformer architecture and contextualized LMs, as well as a more detailed description of recent advancements, we refer the reader to Gasparetto et al. (2022).

3. Analysis of ticket automation literature

Automatic TC can be seen as a specific field of application of text classification (Cunha et al., 2021; Gasparetto et al., 2022; Pistellato, Cosmo, Bergamasco, Gasparetto, & Albarelli, 2018; Revina, Buza, & Meister, 2020). The processing of support tickets is made challenging by the nature of the bodies of text involved: many of these help requests are very brief, and almost always contain technical jargon that should be taken into careful consideration (Cristian, Christian, & Dumitru-Tudor, 2019).

In this section, we provide an overview of Ticket Automation, describing its most prominent subtasks as well as listing notable and recent work in this field. We will put particular emphasis on TC, as



(a) Flattened classifier. (b) Local binary classifiers. (c) Per-level classifiers. (d) Per-parent classifiers. (e) Global classifier.

Fig. 2. Overview of common approaches to HTC, exemplified on a two-level hierarchy.

our research reveals it to be the most common automation procedure in practice. At the end of this section, we also provide a list of datasets often used in this domain's literature. First, however, we describe the similarities between TA and HTC, another sub-field of text classification.

3.1. Relatedness with hierarchical text classification

It is common for ticket categories to have a *hierarchical structure*; these levels identify the incident or help request at different degrees of specificity. In practice, most real-world ticketing scenarios will have at most a shallow hierarchy of two or three levels (Bhowmik, Paul, Usha Nandini, & Prince Mary, 2019).

In this context, *Hierarchical Text Classification* (HTC) methods are a group of approaches especially devised to be applied to text classification environments characterized by a hierarchical label structure. As such, these methods are indeed applicable to these scenarios, though most of these approaches are devised for more complex hierarchies, as well as often being framed as multilabel tasks, which we found to be less common in TC. Nevertheless, many concepts within HTC literature are still useful for the purpose of TC, and we introduce here some basic concepts.

There are multiple generic approaches to HTC, of which Fig. 2 provides a graphical overview. One of the most traditional ways to tackle the hierarchy is to simply convert the problem to a multiclass (or multilabel) classification by *flattening* the hierarchy itself (Koller & Sahami, 1997). Obviously, the main downside of this approach is the loss of hierarchical information. An alternative can be to adopt *local* approaches which, on the other hand, construct classifiers at different levels of the label hierarchy. Classifiers might be per parent, per-node, or per-level (Javed, Shahzad, & Arshad, 2021). While this approach can integrate hierarchical information successfully, it is possible for misclassifications to be propagated incorrectly. Furthermore, having multiple classifiers might not always be convenient. *Global* approaches have been proposed as a solution, devising a single model that is usually built on a flattened classification basis, but modified to integrate hierarchical information (Kiritchenko, Matwin, Nock, & Famili, 2006; Labrou & Finin, 1999).

Briefly, these approaches can be summarized as follows:

- **Flattened**: Unravel the hierarchy and classify on a flat multiclass/multilabel problem (Fig. 2(a));
- **Local (binary)**: Apply a binary classifier on each label node (Fig. 2(b));
- **Local (per-level)**: Apply a (possibly multiclass) classifier to each level of the hierarchy (Fig. 2(c));
- **Local (per-parent)**: Apply a (possibly multiclass) classifier to each parent node. Similar to the per-level approach, but classifiers are specialized to each subset of children labels rather than the entire level (Fig. 2(d));
- **Global**: Apply a single framework that integrates hierarchical information (Fig. 2(e)).

Combination strategies will be required whenever adopting multiple classifiers, the simplest approach being that of considering the result correct only if the result is consistent with the hierarchy (*i.e.*, the labels and sub-labels form a path within the tree). In Section 4, we will discuss our approach to TC. Within this categorization, and as will be shown, our proposal can be seen as a global approach (which, within itself, is comprised of per-level classifiers).

3.2. Ticket automation tasks

As previously mentioned, multiple automation procedures have been devised in the context of TA. In many cases, the straightforward "topical" classification of tickets is sufficient to assign tickets to a specialized group that can deal effectively with the issue. However, depending on the circumstances and resources available, more nuanced and refined techniques can also be applied, often solving more complex tasks. We briefly outline these tasks, though we will focus mostly on the former approach. Whenever appropriate, we will discuss methods within these classes if they provide useful insight into our objectives.

First off, TC itself need not be limited to a topical categorization; some approaches attempt to capture facets other than the topic of a ticket, most commonly a *priority* to determine the urgency of the incident and a *type* which, in a related fashion, is utilized to determine its importance (*e.g.*, information request *vs* incident report) (Beckers, Frommholz, & Bönning, 2009; Lyubinets, Boiko, & Nicholas, 2018). Other methods attempt instead to match tickets directly with an individual expert, rather than groups of experts based on topics (a task related to *expert finding*) (Husain, Salim, Alias, Abdelsalam, & Hassan, 2019). While similar, this task is rendered more complex by the necessity of clearly defining the skills of the expert and the ones required to solve the issue, often also having to consider the availability of the expert in question. Some approaches seek to find the optimal (minimal)

Table 1
Common TA frameworks applied to support tickets.

Task	Application
Ticket Classification (TC)	Categorize tickets in terms of topic, sometimes also type and priority
Expert Finding (EF)	Automatically assign the resolving expert to a ticket
Ticket Routing (TR)	Direct a ticket through the shortest path in a network of experts
Ticket Resolution (RE)	Find an automatic resolution to a ticket, usually based on past solutions

"routing" of tickets through the network of experts (Han, Li, & Sun, 2020; Shao, Chen, Tao, Yan, & Anerousis, 2008). While this might reduce to expert finding whenever only one expert is necessary, this is not always the case. In a medical scenario, for example, it is often important to gather a sequence of opinions from different specialists, therefore requiring a "path" traversing the network of experts. Lastly, in some cases, it might be possible to match tickets directly with their resolution without the need for human intervention (Zhou et al., 2017). There are various approaches to this latter task, the most common being either retrieval-based (*i.e.*, match most suitable historical solution) or generative (*i.e.*, generating an entirely new response, learning from previous ones). Table 1 provides a summary of the different automation procedures that may be applied in the context of support ticket resolution.

3.3. Related work

Most research studies in the context of TC propose new methods or analyze their functioning within a particular domain. There are a few reviews and surveys that discuss the subject. Revina et al. (2020), for instance, deal with TC in the IT domain, exploring text representation techniques, and the performance of various text classifiers. However, they limit their review to more traditional classification methods, such as Support Vector Machines (SVMs) and Random Forests. They discuss the need for explainable TC, as well as which factors are relevant for prediction quality. Kubiak and Rass (2018) discuss TC as a part of a larger work on data-driven techniques for IT-Service-Management. They also discuss its relatedness to hierarchical classification, and thoroughly discuss performance evaluation methods suitable for hierarchical approaches. Again, they mostly discuss traditional methods, such as SVMs, Bayesian models, k-NN methods and Decision Trees. In Fuchs et al. (2022), a literature review of technologies in the field of automated support ticket systems is provided. This review in particular is largely aimed at automated ticket resolution, rather than classification. Young et al. (2019) review text classification procedures in the context of healthcare incident reporting and adverse event analysis. The authors list a wide selection of methods that have been utilized in the healthcare environment and discuss how they can be effectively utilized. The reviewed works mostly consist of traditional classification methods.

Among TA tasks, expert finding may be considered the most closely related to TC, depending on the specific situation and the required ticket routing solution. A generic overview is provided by Husain et al. (2019), which review work in the period 2010–2019. They describe the finding of experts for technical support as an early formulation of expert finding. In Xu and He (2018), trouble ticket routing is framed as an expert recommendation task that also integrates TR components, by learning social profiles for the experts in order to suggest other experts in case the current one is unable to solve the issue. They devise several two-stage expert recommendation algorithms to determine appropriate resolvers for a ticket. They also make the interesting argument that the more standard approach of classifying tickets within single problem types might be limiting, due to the negative impact of misclassifications. Lin, Hong, Wang, and Li (2017) review expert finding methods, focusing on the parts of this task involved in expertise resource selection (extracting expertise-related data for experts) and expertise modeling (building models on the data to identify experts). They examine state-of-the-art algorithms for expert identification up to 2015, by which they rank previously modeled experts based on the probability of them being an expert on a query topic.

3.4. Recent TA methods

As mentioned, Ticket Automation refers to a set of algorithmic solutions that automatically process support tickets and customer requests. In this section, we first describe our study selection procedure; then, we list recent advancements in Tables 2–4. The tables contain a high-level overview of these methods and the notable contribution it brings to the literature. Then, in order to later compare it with our proposal, we analyze in more detail the most relevant works we have found in the TC field. Because of the large number of works, we apply strict constraints in determining their relevance; in particular, we require that an existing code implementation has been made available for a proposed method, such as to reproduce those models. In the last part of this section, we showcase public TA datasets, as found referenced in the reviewed works.

3.4.1. Study selection

We reviewed literature on TA using Google Scholar,[3] DBLP,[4] Scopus,[5] Web of Science,[6] and PapersWithCode.[7] We focused on research published from 2018 onwards, but we still included influential earlier works when we found them to be frequently referenced. The keywords queried were "*ticket automation*", "*ticket classification*", "*support ticket*", "*trouble ticket*", "*expert finding*" and "*ticket routing*". We found that a considerable number of matching results are articles regarding the application of ML algorithms to real-world ticketing systems, but do not provide new solutions or interesting research insights. Moreover, many of these works report the results of relatively few classification methods, most of which are traditional or otherwise not very recent or use paid API-based services. As they do not contribute to our goal, we have excluded them from our search. Much of the work found matching the keyword "expert finding" relates to applications of recommendation systems or ranking methods (Marcuzzo, Zangari, Albarelli, & Gasparetto, 2022) as applied to several domains; we limit our selection to the methods tested on customer-care data. We summarize the results of our search in Tables 2–4, which contain articles related to TC, EF, and TR/RE, respectively.

3.4.2. Description of recent methods

In this section, we analyze recent additions to the TA literature. Table 2 provides a complete list of the works we reviewed, highlighting a wide range of methods and contributions. Moreover, we describe in more detail the methods we considered as baseline comparisons for our proposed methods. These methods were chosen as they provide a public code implementation, as well as being directly applicable to our datasets. As we will discuss, we were not able to reproduce all models selected this way.

[3] https://scholar.google.com
[4] https://dblp.org
[5] https://www.scopus.com/
[6] https://www.webofscience.com
[7] https://paperswithcode.com

Table 2
Ticket Automation literature on the TC sub-task.

Article	Methods included	Datasets	Contribution	Code
Beneker and Gips (2017)	NMF, k-means	–	Clustering and topic modeling methods for TC with TF-IDF features	✗
Montgomery, Damian, Bulmer, and Quader (2018)	RF, XGBoost	–	Ticket escalation prediction with oversampling, feature engineering based on domain knowledge	✗
Xu, Zhang, Zhou, He, and Li (2018)	Clustering	–	Ticket partition and signature-based construction algorithm, problem type classification	✗
Paramesh et al. (2018)	LR, SVM, RF, NB, k-NN	–	Ensemble (voting) classifier, bagging and boosting	✗
Han, Goh, Sun, and Akbari (2018)	CRF, SVM	–	Entity (software product name) extraction and linking for support ticket routing	✗
Lyubinets et al. (2018)	GRU, ATT	Chromium, Linux	Hierarchical Attention model, comparison with traditional classifiers	✓
Wang, Li, Iyengar, Shwartz, and Grabarnik (2018)	MAB	–	TC modeled as a contextual multi-armed bandit problem	✗
Wang, Zeng et al. (2018)	MAB	–	Two novel multi-armed bandit algorithms	✗
Parmar, Biju, Shankar, and Kadiresan (2018)	SVM, k-NN, NB, RF	–	Comparative analysis of five classifiers to label customer issues with predefined topics	✗
Mandal, Agarwal et al. (2019)	SVM, MLP, CNN	–	Enrichment of ticket text with data from attached images	✗
Cristian et al. (2019)	LSTM, Word2Vec	–	Hierarchical Classification, ticket similarity method, multilinguality	✗
Young et al. (2019)	k-NN, DT, NB, LR, SVM, NN	–	Systematic review of classification methods in the context of healthcare incident reports	✗
Kallis, Di Sorbo, Canfora, and Panichella (2019)	FastText	GitHub	Plugin app to automatically label GitHub issues through FastText, issues dataset	✓
Pikies and Ali (2019)	String Matching	–	Comparison of string matching and similarity algorithms	✗
Werner, Li, and Damian (2019)	k-NN, MLP, LR, RF, SVM, Ensemble	–	Comparison of several algorithms for prediction of ticket escalation based on the extracted sentiment	✗
Mukunthan and Selvakumar (2019)	Multi-level Petri net	–	Assignment scheme based on Petri Net model	✗
Xu, Mu, and Chen (2020)	k-NN	–	Integration of several similarity measures through data-driven policy	✗
Revina et al. (2020)	k-NN, DT, NB, LR, SVM	–	Comparative analysis of text representation techniques (TF-IDF, Word2Vec, FastText) and classifiers, linguistic feature extraction	n/a
Asres et al. (2021)	GB, Word2Vec	–	Ticket opening prediction using time windows for feature engineering	✗
Perez, Jean, Urtado, and Vauttier (2021)	TF-IDF, MLP, SVM, SGD, RF, RR, kNN	–	Binary bug classification, genetic algorithm optimization of MLP hyperparameters	✓
Yang (2021)	SVM, Word2Vec	–	Fuzzy SVM to handle outliers and noise	✗
Zicari et al. (2021)	CNN, MLP	Endava	Oversampling of imbalanced classes, ensembles	✗
Tolciu, Săcărea, and Matei (2021)	LSTM, TSNE	–	Supervised classification of German tickets, clustering of embeddings to detect similar classes	✗
Meng et al. (2021)	LSTM	–	Learning character- and word-level representation with bi-LSTM	✗
Ricciardi Celsi et al. (2021)	Bayesian Network	–	Prediction of ticket reopening, comparison of different methods, development of efficient BN	✗
Gamboa et al. (2022)	k-NN	AMS, LTI	Improve k-NN performance with distance function and determination of a suitable k value	✗

Implemented baselines. DeepTriage's authors (Mani et al., 2019) propose a deep bidirectional RNN enhanced with the attention mechanism for TC. An initial preprocessing step removes stopwords and tokenizes text through Stanford's NLTK package (Bird, 2006). Embeddings for each word are then initialized using the Word2Vec algorithm (Mikolov, Chen et al., 2013) and fine-tuned for a few epochs. These word embeddings are then passed to a bidirectional LSTM which acts as an encoder, effectively performing a language modeling task. The outputs of the LSTM layer are aggregated and weighted using the attention mechanism (Bahdanau et al., 2015) to produce the final ticket representations, which are then classified using a linear layer with softmax activation. The Cross-Entropy loss is used during training. The method is validated on three public bug report datasets, which are extracted from the list of reported issues on the Chromium and Firefox browsers, as well as within Mozilla Core software components. To construct the dataset, the authors only consider fixed bugs, and the target label for each ticket is the developer ID that has resolved it.

Kallis et al. (2019) propose TicketTagger, a GitHub plugin for the automatic assignment of labels to GitHub issues. The tool uses the FastText (Bojanowski et al., 2017) library to assign one of three categories based on the title and description of each issue. The labels, used by repository maintainers to organize open issues, can be either "*bug report*", "*enhancement*", or "*question*". The FastText classifier extracts n-grams from documents and learns representations fine-tuned on the target dataset. It then averages these embeddings and feeds them to a linear classifier with a softmax output to obtain the final probabilities for each label (Joulin, Grave, Bojanowski, & Mikolov, 2017).

Lyubinets et al. (2018) describe a hierarchical attention model, combining hierarchical RNNs with attention blocks. As a preprocessing step, the dataset is lowercased, cleaned of noisy words and stopwords, and tokenized using the NLTK package (Bird, 2006). Then, a bidirectional layer with Gated Recurrent Units (GRU) (Cho, van Merriënboer, Bahdanau, & Bengio, 2014) is used to learn word representations based on the context of each word in the sentence (this is the word encoder module). An attention mechanism is used to weigh the contribution of each word to obtain a latent sentence representation. Subsequently, another encoder identical to the previous one is used at the sentence level, and attention is applied to learn overall document embeddings. Finally, similar to the previous works, they use a linear layer with softmax activation for classification (in a multiclass setting). The authors validate this approach on the Linux Bugs dataset using the "Priority" and "Product" fields, and on the "Type" attribute on the Chromium

Table 3
Ticket Automation literature on the EF sub-task.

Article	Methods included	Datasets	Contribution	Code
Mani et al. (2019)	ATT, LSTM	Chromium, Mozilla, Firefox	Bidirectional LSTM with attention, comparison with baselines, new datasets	✓
Husain et al. (2019)	n/a	n/a	Review of main research on applications of EF	n/a
Mandal, Malhotra, Agarwal, Ray, and Sridhara (2019)	MLP, LSTM, CNN, SVM	–	Ensembles of classifiers to assign tickets to a sparse and frequent resolver groups	✗
Li, Jiang, Sun, and Wang (2019)	LSTM, CNN	StackExchange	Combination of question's raiser, content and answerer embeddings for EF	✗
Han and Sun (2020)	GCN	–	Combination of graph- and text-matching to score tickets and expert groups	✗
Agarwal, Bandlamudi, Mandal, Ray, and Sridhara (2020)	Siamese-LSTM, SVM, RF	–	Siamese LSTM to support classification of tickets with infrequent labels, RF with Gini impurity for explainable rule mining	✗
Kundu, Pal, and Mandal (2021)	Link analysis, LDA	–	Combination of topic, reputation and authority based on QA network	✗
Rostami and Neshati (2021)	Probabilistic model	StackOverflow	Intern candidates ranking using community QA data, ensemble method	✗
Ghasemi, Fatourechi, and Momtazi (2021a)	Node2Vec, Word2Vec, MF	StackExchange	Usage of social information and document information for EF in community QA	✗
Ghasemi, Fatourechi, and Momtazi (2021b)	Word2Vec, Node2Vec, MF	StackExchange	User embeddings from community QA social graph to improve text-based EF	✗
Fallahnejad and Beigy (2022)	LSTM, ATT	StackExchange	Learning of semantic relationships between words in answers and experts' skills	✓
Liu, Tang, Liu, Ding, and Tang (2022)	BERT, LDA	StackExchange	Combination of experts' topic interests and authority-based model to improve ranking	✗
Askari, Verberne, and Pasi (2022)	BERT	LegalCQA	Creation of query-dependent user profiles using their answers and the comments to their answers	✓
Peng et al. (2022)	TRAN	StackExchange	Transformer-based multi-view encoders to learn relation between experts and answered questions	✓

Table 4
Ticket Automation literature on the RE and TR sub-tasks.

Article	Methods included	Datasets	Contribution	Code	Task
Zhou et al. (2017)	MLP, CNN	–	Mapping tickets to resolution actions using a quality-model estimation	✗	RE
Han and Sun (2017)	n/a	n/a	Framework for the evaluation of TR and human-assisted algorithms	✗	TR
Madaan, Singh, Kumar, and Dasgupta (2017)	Word2Vec	–	Retrieval of tickets and solutions from noisy hardware tickets with recall- and precision-oriented classifiers	✗	RE
Xu and He (2018)	Network analysis	–	Combination of expertise profiles and social profiles from tickets resolution sequences	✗	RE
Xu, He, Zhou, and Li (2018)	Network model, similarity	–	Routing models using textual descriptions and historical routing sequences	✗	TR
Watanabe et al. (2018)	HMM	–	Resolution action suggestion with sentence alignment strategy	✗	RE
Wang, Shwartz, Grabarnik, Nidd, and Hwang (2019)	CNN, XGBoost, k-NN, MLP, RF, SVM, DT	–	Combination of CNN for feature learning and traditional methods for classification step	✗	RE
Han et al. (2020)	Network analysis	–	Ranking method for TR, group feature engineering	✗	TR, EF
Ferland, Sun, Fan, Yu, and Yang (2020)	LSTM, LDA, k-NN, Siamese NN	–	Ensemble model for ticket resolution using supervised and unsupervised methods	✗	RE
Ali Zaidi et al. (2022)	Similarity, LSTM	–	Flexible framework for action suggestion, action extraction module	✓	RE
Bannihatti Kumar, Yarramsetty, Sun, and Goel (2021)	BERT, GPT	–	Multi-task model for experts support in problem resolution with multi-task training	✗	RE
Subbarao, Venkatarao, and Suresh (2022)	k-means, LDA, NMF	–	Prediction of incident resolution action using topic modeling	✗	RE
Fuchs et al. (2022)	n/a	n/a	Review of ML applications in Automated Ticketing Systems	n/a	TR, TC

dataset that was used by DeepTriage's authors (Mani et al., 2019). Unfortunately, we were unable to reproduce this framework using the published code because of dependency issues.

3.5. Datasets used in research

Table 5 showcases datasets utilized in the works we have reviewed in this manuscript. The table only reports openly available datasets, describing their size, whether they are multilabel and hierarchical in nature, the generic automation task as previously defined, and the topical domain that describes its content. It is worth noting that many works in TA literature apply their methods to proprietary datasets, which are therefore not available.

4. Proposed approach

In previous sections, we provided an overview of the automatic support ticket resolution landscape. In this section, we propose a novel method for the specific task of automated topical classification of tickets within shallow hierarchies. Our approach is based on pre-trained

Table 5
Public TA datasets referenced in the reviewed works.

Dataset	Reference	N. samples	Multilabel	Hierarchical	Task	Domain
Financial	Sundaramahadevan (2022)	78,313	✓	✗	TC	Finance
Endava	Żak (2018)	48,549	✓	✓	TC	Helpdesk
IT company	Polato (2017)	21,348	✓	✓	TC	Helpdesk
GitHub	Kallis et al. (2019)	30,000	✗	✗	TC	Issues/bugs
LTI	GitHub.com (2020)	48,543	✗	✗	TC	Helpdesk
AMS	Sunil (2020)	12,810	✗	✗	TC	Helpdesk
Chromium	Mani et al. (2019)	383,104	✗	✗	EF	Issues/bugs
Mozilla	Mani et al. (2019)	314,388	✗	✗	EF	Issues/bugs
Firefox	Mani et al. (2019)	162,307	✗	✗	EF	Issues/bugs
Enron	Klimt and Yang (2004)	500,000	✗	✗	MC	E-mail
StackExchange	Stack Exchange Inc. (2022)	(77 GB)	✓	✗	TC, EF	(QA) Coding
LegalCQA	Askari et al. (2022)	9,897	✗	✗	EF	(QA) Legal
Linux	Lyubinets et al. (2018)	16,456	✓	✓	TC	Issues/bugs
Bitext	Bitext Innovations (2019)	20,000	✗	✗	TC	Helpdesk
StackOverflow	Gharebagh, Rostami, and Neshati (2018)	2,2M	✗	✗	EF	(QA) coding

Transformer-based LMs, which are currently state-of-the-art in terms of text representation. In particular, while the LMs are tasked to extract a semantically meaningful representation, our main addition is a multi-level framework that exploits hierarchical information contained within the label structure to perform a more accurate classification. In addition, we wish to explore various alternatives proposed in the literature for the creation of unified document embeddings to be used for the purpose of classification. These experiments were essential to our investigation, as they allowed for much better results in practice. Moreover, they provide useful insights into the development of individual document embeddings in this specific domain.

The following section will introduce our experimental approach. We first detail the datasets utilized and the baselines implemented. Then, we discuss multiple strategies to combine word embeddings as derived from a BERT model. Note that, though our specific approaches utilize BERT as a basis, they are agnostic to any LM capable of producing embeddings for words in a document. We conclude the section by describing our proposed approaches in more detail.

4.1. Datasets used

We evaluated our methods on the Financial (Sundaramahadevan, 2022) and Linux Bugs (Lyubinets et al., 2018) datasets. We utilized the scraping script provided by Lyubinets et al. (2018) to produce a larger dataset of bugs, while we derive the Financial dataset from the one made available on Kaggle.[8] General statistics for the datasets can be found in Table 6, where we also report statistics after the preprocessing operations described in the next section. A sample of labels organized in their hierarchical structure is shown in Figs. 3 and 4.

Financial dataset. The Financial dataset contains 78,313 anonymized customer complaints from a financial company, which are essentially support tickets, although only 21,071 of these have a valid message written in natural language. All 9tickets in the dataset have been annotated by customers and/or helpdesk personnel with a "Product" and a "Service" category. While the original dataset has been used for topic modeling tasks based on the products/services of the tickets, we use these labels as the prediction target of our models. We utilize the product field as the main label (*e.g.*, "*Debt collection*", "*Mortgage*", etc.), and the sub-product field as a sub-label (*e.g.*, "*Credit card debt*", "*Checking account*", etc.). It is worth noting that the hierarchy of this dataset is rather weak, and we found the classification task to be hard in terms of predicting categories. There are multiple labels with similar or ambiguous meanings; for instance, there are pairs of labels such as "*Credit card*" and "*Credit card or prepaid card*", or even multiple similar categories such as "*Mortgage*", "*Mortgage debt*", "*FHA mortgage*", "*Other mortgage*" and "*Other type of mortgage*". Clearly, choosing the correct label would be non-trivial even for a human labeler. Below is an example extracted from this dataset:

Product (label): Vehicle loan or lease
Sub-product (sub-label): Loan
Description Vehicle was financed on `[XXX]` and last payment was `[XXX]` balance was `[XXX]`. Chase auto never release the Title of the vehicle. I called chase Auto finance division in year their answer was last payment was not clear. I asked them if they sent any written notice by mail if they did it send that but they never replied back.

The linux bugs dataset. The Linux Bugs dataset contains bugs from the Linux kernel bug-tracker,[9] and was first proposed by Lyubinets et al. (2018). The one used in this work is an expanded version that we obtain by further crawling the online bug-tracking portal, and contains more than double the number of bug reports with respect to the original version. The support tickets are classified by users in terms of importance, related product, and specific component. We utilize the "Product" field as the main label (*e.g.*, "*Network*", "*Drivers*", etc.), and the "Component" category as a sub-label (*e.g.*, "*BIOS*", "*Scheduler*", etc.). Again, below is an example:

Product (label): SCSI Drivers
Component (sub-label): Other
Title: MPT2SAS drops all HDDs when under high I/O
Description: I have this issue that refused to be solved no matter what I do. My ASRock comes with onboard SAS controller (LSI 2308), since I recieved it always does this one thing: Drops all HDDs connected to it. It happens only under heavy IO operations after a few minutes. I can recreate it easily by running either dd, md5deep or even btrfs scrub. Kernel locks, can't even shut it down from console and a quick ls /dev/disk/by-id shows that all the HDDs connected to the SAS controller have disappeared. It happens with the stable kernel (3.9 and 3.10.3) and the mainline (3.11-rc2) as of this day. It's not a hardware issue, because I installed a Windows Server 2012 on the same machine with a few HDDs I have laying around and beat the controller to the ground and it never hanged. So I know it's a Linux-specific issue. Dmesg logs before and after the issue are attached. Thank you.

For both datasets, the aim of our classifiers is to predict the *flattened label*, which is obtained by concatenating the label with the sub-label. As previously mentioned, we refer to this as the T2 task. For example, the bug report above would be labeled as "SCSI-Drivers_Other".

[8] https://www.kaggle.com/datasets/venkatasubramanian/automatic-ticket-classification

[9] https://bugzilla.kernel.org

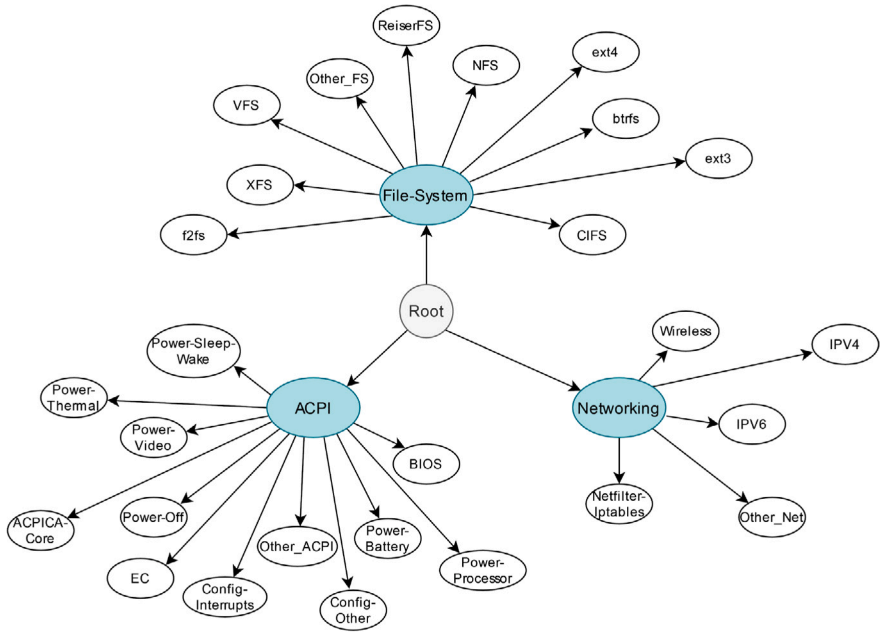


Fig. 3. Example of the hierarchical structure of 3 labels in the Linux Bugs dataset.

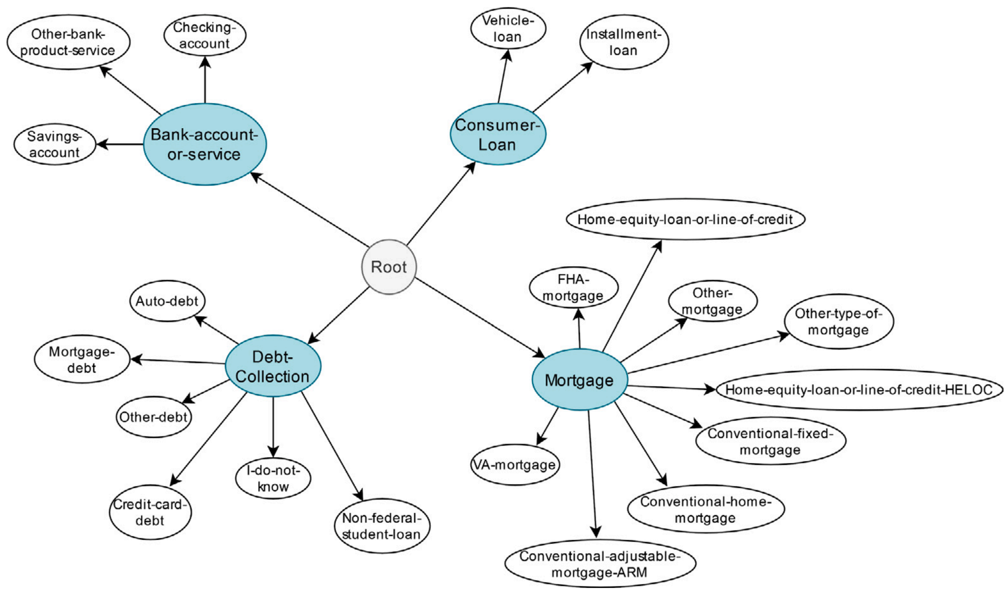


Fig. 4. Example of the hierarchical structure of 4 labels in the Financial dataset.

Table 6
Statistics for the datasets utilized in the experiments.

Dataset	Version	Samples	Avg chars	Labels	Sub-labels	Flattened labels
Linux Bugs (Lyubinets et al., 2018)	Original[a]	38,194	2,541	20	177	190
	Final[b]	35,050	2,536	17	73	85
Financial (Sundaramahadevan, 2022)	Original	20,925	1,342	17	67	81
	Final	20,576	1,344	13	35	42

[a]After removal of tickets that are unlabeled, missing a message body or duplicated.
[b]After the removal of rarest classes and the preprocessing operations shared across methods.

4.1.1. Preprocessing

We followed a process similar to the one adopted by Mani et al. (2019) for the initial cleanup of both datasets. This preprocessing is aimed at removing noise and non-informative bits of text while maintaining sentence structure intact as much as possible. Indeed, this is necessary for recent, contextualized LMs to be effective (Gasparetto, Cosmo, Rodola, Bronstein, & Torsello, 2017; Gasparetto et al., 2022).

The raw datasets are filtered by removing duplicates and any entry where the main body of the ticket is void. Titles and descriptions

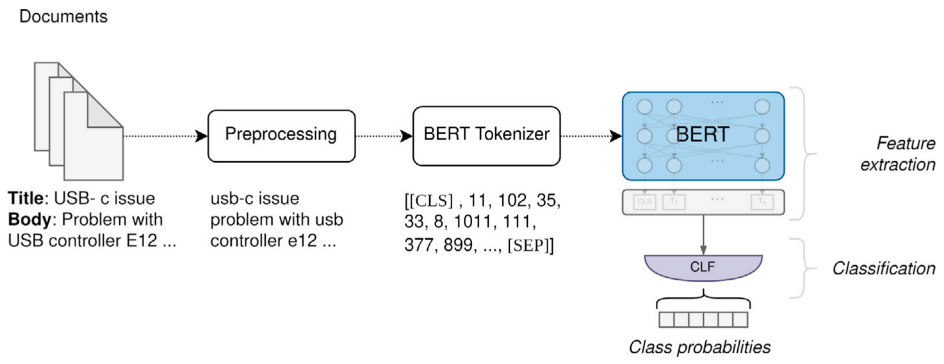

Documents
Title: USB- c issue
Body: Problem with USB controller E12 ...
Preprocessing
usb-c issue problem with usb controller e12 ...
BERT Tokenizer
[[CLS] , 11, 102, 35, 33, 8, 1011, 111, 377, 899, ..., [SEP]]
BERT
Feature extraction
CLF
Classification
Class probabilities

Fig. 5. Tickets processing pipeline used with our BERT-based models.

are concatenated to generate a single content descriptor for tickets in the case of the Linux dataset (as Financial tickets have no title). Furthermore, in order to reduce the already severe imbalance within these datasets, we apply a threshold constraint: labels and sub-labels with fewer than a certain number of representative tickets are excluded. The threshold chosen for the comparatively smaller Financial dataset was 30, while it was 100 for the larger Linux dataset. Bar charts describing the distribution of all labels are attached in the supplemental material.

The DeepTriage-inspired (Mani et al., 2019) text sanitization procedure consists of the removal of URLs, some HEX codes, as well as a lowercasing of all words. Following the procedure from Lyubinets et al. (2018), which initially proposed the Linux Bugs dataset, we also experimented with a more aggressive preprocessing approach to remove "garbage" text from Linux bug reports, which mainly consists of the removal of memory addresses. Though the filter works well, we did not register improvements whenever including this procedure and therefore decided to exclude it from the finalized pipeline. We also note that traditional methods showcased an improvement when excluding stopwords from the list of processed tokens. The effect of our basic preprocessing can be visualized in the sample ticket below, which is the same as the one shown in the previous section, annotated with the label used as target for classification. The size and statistics of the dataset we used in our experiments are summarized in Table 6 (final version).

Category (flattened label): SCSI-Drivers_Other
Ticket body: mpt2sas drops all hdds when under high i/o i have this issue that refused to be solved no matter what i do . my asrock comes with onboard sas controller (lsi 2308) , since i recieved it always does this one thing : drops all hdds connected to it . it happens only under heavy io operations after a few minutes . i can recreate it easily by running either dd , md5deep or even btrfs scrub . kernel locks , can't even shut it down from console and a quick ls /dev/disk/by-id shows that all the hdds connected to the sas controller have disappeared . it happens with the stable kernel (3.9 and 3.10.3) and the mainline (3.11-rc2) as of this day . it's not a hardware issue , because i installed a windows server 2012 on the same machine with a few hdds i have laying around and beat the controller to the ground and it never hanged . so i know it's a linux-specific issue . dmesg logs before and after the issue are attached . thank you .

After preprocessing, all models require a tokenization step. In the case of LMs like BERT, the tokenizer utilized was the one associated with the original work (*i.e.*, WordPiece for BERT) (Schuster & Nakajima, 2012). In the case of DeepTriage and traditional methods, we utilized NLTK's *word_tokenize* function (Bird, 2006). In short, this approach is an improved word-level tokenization based on regular expressions to split text as in Treebank-3 (Marcus, Santorini, Marcinkiewicz, & Taylor, 1999). The overall classification pipeline for our experiments is visualized in Fig. 5.

4.2. Baselines

In order to validate our results, we tested a set of baselines drawn from similar works which have obtained state-of-the-art results on the specific TC task, as well as some broader state-of-the-art text classification approaches. We report results for the following:

- **SVM** (Boser et al., 1992): Similarly to Lyubinets et al. (2018), we test an approach based on a TF-IDF (Jones, 1972) document representation, which is then fed to SVM classifiers with linear kernel. The method utilizes a one-vs-rest approach, creating a classifier for each node of the hierarchy (similar to Fig. 2, but only on the flattened representation). The best parameters are sought through a grid search prioritizing macro F_1, and are then utilized to retrain the model (test data is never seen during this procedure). As additional preprocessing, this method removes stopwords and seeks bi-grams within tokens;
- **TicketTagger/FastText** (Kallis et al., 2019): We follow the approach of the authors of TicketTagger, which is a straightforward application of FastText (Bojanowski et al., 2017). The classifier itself is quite simple and consists of an MLP optimized to utilize FastText's embeddings (Joulin et al., 2017). In our experiments, we use the "autotune" option provided, extracting a selection of 20% of the training samples for validation (as with other methods);
- **DeepTriage** (Mani et al., 2019): The main architecture is based on a Deep Bidirectional Recurrent Neural Network (DBRNN-A), enriched with the attention mechanism and LSTM units. Text representation is achieved by creating fresh Word2Vec embeddings, first trained and then fine-tuned on the recurrent architecture. In their work, the authors build their model in the context of predicting a developer available and able to resolve the bug, therefore resembling EF more closely. Nonetheless, as an (extreme) multiclass method developed in the same context, we applied it to our setting.
- **BERT** (*flattened*) (Devlin et al., 2019): This is a straightforward application of the BERT LM for sequence classification, which is based on the attachment of a classifier head on top of the LM. The classifier head is a single feed-forward layer, and the entire model is trained to predict the flattened, second-level labels. This is the standard approach to classification with BERT. We experimented with various document embedding summarization strategies, as discussed before;
- **XLNet** (*flattened*) (Yang et al., 2019): The approach is the same as with BERT. XLNet is an autoregressive model, more akin to traditional LMs, but devises a clever pre-training approach based on word token permutations in order to introduce bidirectional information (*i.e.*, both left and right context). We experimented with a few document embedding summarization strategies (see Appendix A), but only displayed the results of the best-performing one in the main body of this article (which utilizes the last token as a representative).

Table 7
Summarization strategies for document embeddings.

Basis	Strategy	Emb. size	Description
cls	*last*	d	[CLS] token embedding from last layer (default strategy)
	avg$_h$		Average of the [CLS] token embeddings from the last h layers
	concat$_h$	$d*h$	Concatenation of the [CLS] token embeddings from the last h layers
avg	*last*	d	Average of all token embeddings[a] from the last layer
	avg$_h$		Average of the average of token embeddings from the last h layers
	concat$_h$	$d*h$	Concatenation of the average of token embeddings from the last h layers
max	*last*	d	Column-wise maximum of all token embeddings[a] from the last layer
	avg$_h$		Average of the max of token embeddings from the last h layers
	concat$_h$	$d*h$	Concatenation of the max of token embeddings from the last h layers
max_min	*last*	$d*2$	Concatenation of the max and min of embeddings from the last layer
	avg$_h$		As above, but averaging vectors from the last h layers
max_avg	*last*	$d*2$	Concatenation of the max and average of embeddings from the last layer
	avg$_h$		As above, but averaging vectors from the last h layers
sum_sum_norm	*last*	d	Sum of token embeddings divided by the sum of the norm of embeddings
	concat$_h$	$d*h$	Like *last* but concatenating the last h layers
sum_norm	*last*	d	Sum of token embeddings divided by its norm (*i.e.*, normalized sum)
	concat$_h$	$d*h$	Like *last* but concatenating the last h layers

[a]Excluding special symbols (*e.g.* [CLS] and padding).

4.2.1. Other methods considered

In addition to the ones reported, we also tested a number of other approaches. The results are not listed because the methods either obtained unsatisfying performances or we were not able to reproduce the same results as reported in the original work. Other traditional approaches such as Naive Bayes (Xu et al., 2017) and Random Forests (Ho, 1995) resulted in below-average performances. We were unable to run the code published by Lyubinets et al. (2018), and thus unable to verify their proposed method.

4.3. Document embedding summarization strategies

Before a body of text can be embedded in any way, it must first be split into atomic units (words or parts thereof); this task is performed by specialized modules called *tokenizers*. In the case of BERT models, tokenization is based on the WordPiece algorithm (Devlin et al., 2019; Schuster & Nakajima, 2012). Without going into detail, this is a sub-word tokenization strategy, which has been trained on a vast corpus of documents to extract an efficient (in terms of vocabulary size) subset of tokens. These algorithms operate on the assumption that common words should be kept in the vocabulary as-is, while rare words should be split, in order for a more significant representation of its composing segments to be learned.

A trained tokenizer truncates each input document to a certain threshold of maximum tokens as determined by the specific model, also padding any shorter sequences to the same length (in this case, 512). The BERT tokenizer additionally pre-pends to each input sequence a special symbol, the [CLS] token, which is expected to predict the target binary label of the Next Sentence Prediction task (NSP) during pre-training (Devlin et al., 2019).

Tokenized text is presented as a sequence of ids, each one mapping to a word (or special symbol) in the BERT vocabulary — which is composed of about 30,500 English words and symbols in the HuggingFace model we adopted.[10] In its smallest version (*BERT-base*), BERT consists of twelve stacked encoder blocks, each one producing contextualized embeddings for input tokens, including the [CLS] token. Being pre-trained to capture sequence-wise information for a binary classification task, the [CLS] representation of the last layer is considered a good candidate to be used in a general classification task.

This was the strategy adopted by the authors of the BERT architecture, which also tested the model in a multiclass setting. In their work, the [CLS] embedding is passed through the NSP prediction head and to a final linear layer with softmax activation (Devlin et al., 2019). This strategy has been widely accepted as the "default" way to adapt BERT-like models to downstream classification tasks. However, some researchers suggest that other strategies may be preferred. Tanaka, Shinnou, Cao, Bai, and Ma (2019) test several strategies to enrich BERT embeddings with a BoW feature vector, as well as averaging together word embeddings in the last and second-to-last layers; they report considerable improvements in classification over the default strategy. Similarly, Reimers and Gurevych (2019) also argue that the [CLS] embedding makes for a sub-optimal sentence representation, and propose SentenceBERT for the generation of sentence embeddings. Moreover, some researchers have suggested that the different encoder layers within the architecture can become "specialized" in the extraction of particular linguistic features, like syntactic and semantic ones, hence potentially providing additional information for classification (de Vries, van Cranenburgh, & Nissim, 2020; Jawahar, Sagot, & Seddah, 2019).

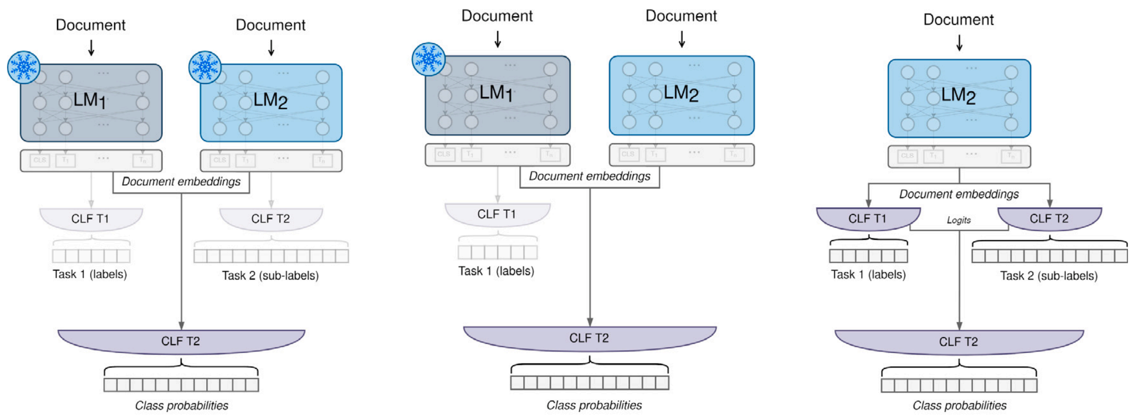
In this work, we test several strategies to obtain document embeddings from word embeddings and compare them with the standard approach using the [CLS] token. The tested strategies are described in detail in Table 7. Most of them are inspired by experiments in previous work (Miraj & Aono, 2021; Tanaka et al., 2019). On the other hand, the *sum_sum_norm* and *sum_norm* strategies were devised following the intuition that embeddings could be considered oriented vectors in a high dimensional space, and that the meaning of a document could be approximated by summing these vectors and normalizing the sum in different ways. Our findings will be outlined in Section 5.

4.4. Multi-level models

TC datasets most commonly contain several labels organized in a hierarchical structure. For instance, the two ticket datasets used in this work are labeled with two levels of categories: a macro category, and a secondary, more fine-grained topic indicator. These datasets are derived from a real-world ticketing system and bug report repository respectively; thus, they provide a good indication of common structures within these environments.

In the following paragraphs, we will describe how the categorization of documents in this two-level hierarchical setting can be improved by combining models that work on different levels. All models use pre-trained LMs (in this case, BERT) to extract features from each document's text. Classification is achieved by adding a single linear layer with softmax activation and fine-tuning the model, a widely utilized

[10] https://huggingface.co/bert-base-uncased



(a) ML-LM. (b) SupportedLM. (c) DoubleHeadLM.

Fig. 6. Two-level classification models.

approach (Devlin et al., 2019; Gasparetto et al., 2022). Unless specified otherwise, we assume that BERT embeddings are also trainable during the fine-tuning procedure — again, as it is standard in these cases.

As a first step, we define two multiclass classification objectives with their respective set of target classes:

1. **T1**: prediction of first-level target class (*i.e.*, the macro-label);
2. **T2**: prediction of second-level target class (*i.e.*, the sub-label).

Note that, to avoid redundancy in T2, we flatten the tree of categories to obtain the final sub-labels. Through this process, any duplicate pair of sub-labels is transformed into two separate classes (*e.g.*, if two labels both have the "*other*" sub-label). Notably, if all sets of sub-labels are already disjoint, this procedure does not increase the number of sub-labels.

We experiment with three multi-level classifier frameworks:

- The first one combines two classifiers that were previously trained on T1 and T2, respectively (`ML-LM`);
- The second one is trained on T2 and is supported by a classifier pre-trained on T1 (`SupportedLM`);
- The third classifier is similar in spirit to the first one but is trained end-to-end on both tasks with a single LM (`DoubleHeadLM`).

4.4.1. *Multi-level language model*

The first Multi-Level LM, which we call `ML-LM`, is shown in Fig. 6(a). It utilizes two LMs. The first one is trained to predict the first level of labels (T1), while the second one is trained to predict second-level target classes (T2). Again, task T2 operates on a flattened representation to avoid sub-label duplicates.

After the models have been trained disjointly on the separate tasks, their respective weights are frozen. In `ML-LM`, the document embeddings obtained by these models (using one of the summarization strategies of Table 7) are concatenated together and fed to a linear layer, which is then trained on T2. To reiterate, at this point only the parameters of the classification head are learnable, meaning computational costs are much more affordable. Note that the classifier outputs from both base models are discarded since we are interested in the pre-classification embeddings only.

4.4.2. *Supported language model*

The `SupportedLM` approach utilizes a LM previously fine-tuned on T1 as "support" to a secondary LM, which has not yet been fine-tuned. The second model is trained on the T2 task as before, but with additional information derived from the support model. The document embeddings from the two models are concatenated before the final classification layer. In this case, this step effectively adds extracted features from the first-level label, giving the second model a chance to rectify its prediction on T2 based on the feedback from the first model. The architecture of the model is showcased in Fig. 6(b).

4.4.3. *Double-head language model*

In the `DoubleHeadLM` approach, a single LM is used to produce document embeddings that are fed to two intermediate linear classifiers, one for each task. The two outputs of this prediction step are then concatenated and fed to a final linear classifier trained on the second task. The whole model is trained end-to-end, with three loss objectives, one for each classifier. The goal of this approach is to enforce a regularization effect on the produced embeddings so that they better reflect all the information needed to predict both levels of labels. The final classifier combines the predictions of the intermediate classifiers to predict the correct labels. Fig. 6(c) showcases the architecture of this model.

5. Experimental results

In this section, we showcase the results of our models. After discussing our performance metrics, we illustrate our results. A thorough discussion of our findings and implications are provided in Section 6.

Experiments are run on a machine with an Intel i9-9900K CPU, an Nvidia GeForce RTX 2080 Ti GPU and 64gb of RAM, with CUDA 10.1 and Python 3.10 used at runtime. The SVM algorithm is based on Scikit-learn's (Pedregosa et al., 2011) linearSVC implementation, while DeepTriage is developed in Keras (Chollet et al., 2015). All contextualized language models (including BERT, XLNet and their custom variations) are developed in PyTorch 1.11 (Paszke et al., 2019).

5.1. *Metrics*

In order to benchmark the presented methods, we utilize standard classification metrics, *i.e.*, accuracy, precision, recall, and F_1-score. As we are performing a typical supervised task, we can consider the truthfulness of the predictions against the *ground truth* from the datasets. In binary classification, the two classes are denoted as positive (P) and negative (N), in which the former expresses a correct prediction. The *accuracy* metric is expressed as the ratio of correct predictions, both true positives (TP) and negatives (TN), with respect to the total number of predictions, as follows:

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

Other metrics (precision and recall) have a larger focus on the impact of false predictions. *Precision* measures the proportion of positive predictions which were truly positive (*i.e.*, correctness), while *recall* measures the proportion of overall positives captured by the model (*i.e.*,

Table 8
Test set results[a] with BERT classifier on T2 comparing summarization strategies on the Linux Bugs dataset. Best results are outlined in bold.

Basis	Strategy	Acc	F_1	Prec	Rec
cls	*last* (p)[b]	0.518 [± 0.006]	0.354 [± 0.009]	0.386 [± 0.009]	0.365 [± 0.006]
	last	0.566 [± 0.012]	0.446 [± 0.018]	0.479 [± 0.027]	0.452 [± 0.017]
	avg_2	0.531 [± 0.010]	0.393 [± 0.012]	0.420 [± 0.018]	0.398 [± 0.012]
	$concat_2$	0.535 [± 0.010]	0.400 [± 0.014]	0.426 [± 0.015]	0.401 [± 0.012]
	$concat_3$	**0.571** [± 0.008]	0.456 [± 0.013]	**0.498** [± 0.013]	**0.458** [± 0.015]
	$concat_4$	0.568 [± 0.009]	**0.457** [± 0.014]	0.490 [± 0.013]	0.458 [± 0.017]
	$concat_5$	0.565 [± 0.012]	0.456 [± 0.013]	0.486 [± 0.011]	0.458 [± 0.018]
	$concat_6$	0.560 [± 0.011]	0.453 [± 0.015]	0.486 [± 0.013]	0.453 [± 0.017]
	$concat_{10}$	0.562 [± 0.008]	0.455 [± 0.014]	0.485 [± 0.015]	0.457 [± 0.016]
avg	*last*	0.525 [± 0.008]	0.387 [± 0.010]	0.420 [± 0.015]	0.388 [± 0.010]
	avg_2	0.522 [± 0.005]	0.383 [± 0.013]	0.409 [± 0.021]	0.387 [± 0.010]
	$concat_2$	0.523 [± 0.007]	0.390 [± 0.009]	0.411 [± 0.015]	0.394 [± 0.011]
max	*last*	0.522 [± 0.011]	0.385 [± 0.014]	0.415 [± 0.011]	0.391 [± 0.014]
	avg_2	0.519 [± 0.007]	0.375 [± 0.013]	0.395 [± 0.021]	0.387 [± 0.014]
	$concat_2$	0.518 [± 0.006]	0.373 [± 0.015]	0.401 [± 0.021]	0.383 [± 0.012]
max_min	*last*	0.522 [± 0.010]	0.377 [± 0.011]	0.395 [± 0.019]	0.395 [± 0.010]
	avg_2	0.522 [± 0.009]	0.374 [± 0.010]	0.395 [± 0.019]	0.385 [± 0.011]
max_avg	*last*	0.516 [± 0.007]	0.381 [± 0.012]	0.406 [± 0.017]	0.390 [± 0.012]
	avg_2	0.519 [± 0.006]	0.379 [± 0.007]	0.402 [± 0.011]	0.392 [± 0.007]
sum_sum_norm	*last*	0.278 [± 0.018]	0.070 [± 0.006]	0.076 [± 0.006]	0.087 [± 0.007]
	$concat_2$	0.252 [± 0.032]	0.050 [± 0.007]	0.046 [± 0.007]	0.071 [± 0.008]
sum_norm	*last*	0.406 [± 0.018]	0.171 [± 0.017]	0.192 [± 0.023]	0.206 [± 0.016]
	$concat_2$	0.379 [± 0.013]	0.135 [± 0.019]	0.149 [± 0.022]	0.179 [± 0.021]
	$concat_5$	0.388 [± 0.015]	0.135 [± 0.015]	0.149 [± 0.015]	0.180 [± 0.014]

[a]The standard deviation over the 6 runs is reported in brackets.
[b]Pooled, as in the standard approach for BERT classification (*i.e. cls pooled*).

completeness). Finally, the *F-score* is a combination of precision and recall. In particular, the most commonly utilized version of this metric is the F_1*-score*, which combines these values by taking their harmonic mean:

$$\text{Precision} = \frac{TP}{TP+FP} \qquad \text{Recall} = \frac{TP}{TP+FN}$$

$$F_1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

In the case of multiclass problems (such as the ones examined in this work), the metrics above can be applied separately to each class and averaged. In this work, we utilize *macro* averaging, meaning that all classes contribute to the average in the same manner (*i.e.*, without weighing for class imbalance).

5.2. Results

This section contains the results of our experiments. We compute metric values using two repetitions of 3-fold cross-validation: in every run 33% of data is used as test set, and the remaining part is used for training. We select this specific number of folds to reduce the time needed to train and test all models, since the procedure is repeated twice for each one. This allows us to account for the variability of results and still contain the overall training time. Splits generated for testing or validation are sampled using stratification, to ensure that labeled documents are selected in the same proportion as they appear in the whole dataset. All results reported in this section are obtained by averaging the measured metrics over all six runs. Before testing, we used 20% of the training split as validation set to select the models' hyper-parameters. More details on the validation procedures are provided in the supplemental material for this work.

5.2.1. Document embedding summarization strategies results

Table 8 lists results obtained by training BERT-based classifiers on the T2 task with different document embedding summarization strategies. Again, before measuring performance on the test split, we used 20% of the training set to determine the optimal values for the following hyperparameters:

- Number of fine-tuning epochs;
- Learning rate (choosing between $5e^{-6}$, $1e^{-5}$, $2e^{-5}$, $5e^{-5}$);
- Whether to apply stronger preprocessing procedures;
- Whether to train the model with weighted cross-entropy.

The learning rate values experimented with were inspired by the ones used in the original BERT paper, scaling them to suit our reduced training batch size. We apply the same preprocessing used in DeepTriage (Mani et al., 2019), and we verify whether the additional cleaning operations proposed by Lyubinets et al. (2018) are beneficial. We additionally try an approach that weighs each class's contribution to the cross-entropy value according to their support, so that less occurring classes contribute the most. Finally, to select the best number of epochs, we train using early stopping based on the loss value, with patience set to 2 epochs (as fine-tuning procedures usually run for very few epochs). We test all the combinations of the above-mentioned parameters on the validation set using a 3-fold CV. We validate using both the *cls last* and *avg last* strategies (among the ones listed in Table 7).

In both cases, the best results were achieved by training for no more than 3 epochs, with the learning rate set to $2e^{-5}$, unweighted loss, and no additional preprocessing. Therefore, we used these settings for all tests reported in Table 8. As the *cls* $concat_2$ strategy was the best among strategies that utilized multiple hidden layers, we further tested this strategy for varying values of h. As it turns out, the *cls* $concat_3$ strategy outperforms the previous one.

5.2.2. Multi-level models results

We present in Table 9 results obtained with the multi-level classifiers. We test these configurations using the "bert-base-uncased" model (Devlin et al., 2019) available on HuggingFace (Wolf et al., 2020). Models are tested using only the best-performing averaging strategy, based on the results presented in the previous section.

As with the previous tests, we use the unweighted cross-entropy loss and make no additional preprocessing. The base LMs and respective classifiers utilized in the first and second architecture are first trained with the same hyperparameters chosen for the T2 task. When specified,

Table 9
Test set results[a] with multi-level classifiers on T2 with *cls concat*$_3$ averaging strategy. Best results are outlined in bold.

Model	Acc	F_1	Prec	Rec
Linux Bugs				
ML-BERT	0.602 [± 0.010]	**0.500** [± 0.014]	0.518 [± 0.012]	**0.501** [± 0.013]
SupportedBERT	**0.611** [± 0.007]	0.485 [± 0.013]	**0.525** [± 0.010]	0.487 [± 0.014]
DoubleHeadBERT	0.549 [± 0.008]	0.400 [± 0.013]	0.442 [± 0.010]	0.410 [± 0.014]
Financial				
ML-BERT	0.633 [± 0.029]	**0.267** [± 0.020]	**0.311** [± 0.027]	**0.268** [± 0.019]
SupportedBERT	**0.649** [± 0.021]	0.249 [± 0.010]	0.303 [± 0.019]	0.251 [± 0.010]
DoubleHeadBERT	0.576 [± 0.011]	0.184 [± 0.009]	0.221 [± 0.023]	0.193 [± 0.010]

[a]The standard deviation over the 6 runs is reported in brackets.

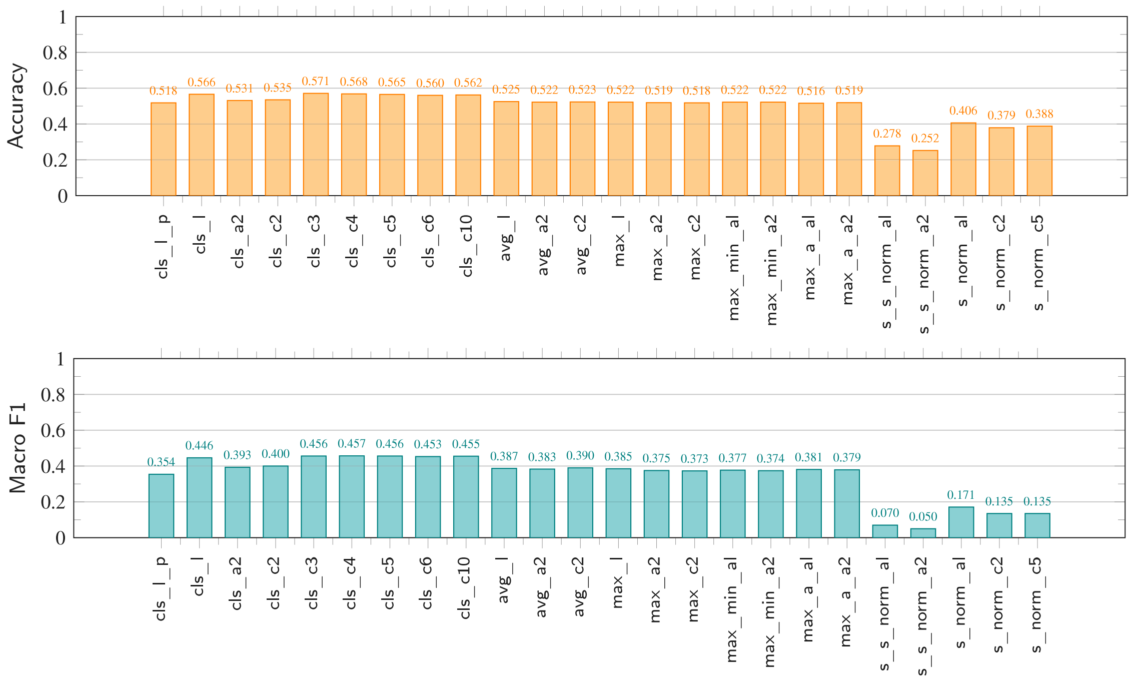


Fig. 7. Visual comparison of the accuracy and macro F_1 score of various approaches to document embedding summarization on the Linux Bugs dataset. Abbreviations: l = last, a = average, c = concat, s = sum.

the LMs then have their weights frozen for the remainder of the training process. To reiterate, these are LM_1 and LM_2 in the first architecture, while this applies only to LM_1 in the second architecture (Fig. 6). We perform a separate hyperparameter tuning for learning rate and number of epochs for the final classifier of these architectures (*i.e.*, the bottom CLF T2 in the figures), testing them as before on the validation set and utilizing a 3-fold CV. During tests the models are trained for 2 epochs with learning rate set to $2e^{-5}$ for SupportedLM and DoubleHeadLM and $2e^{-4}$ for ML-LM.

5.2.3. *Baselines results*

We report in Table 10 the results obtained using the baselines outlined before. We perform a hyperparameter search for all neural models to select the best learning rate. For FastText, we report test results after performing the auto-tune procedure for 25 min on 20% of the training set. Before testing, the model is retrained on the entire training set with the best hyperparameters. XLNet is validated on the same learning rates used for BERT, and we use the same early-stopping strategy to select the number of epochs. We used the loss value as the target score in validation with neural networks, and we use the F_1-score for the other baselines. Other details on our parameter validation experiments are detailed in the supplemental material we provide. We also tested a set of averaging strategies for XLNet, which are reported in Appendix A. The default strategy suggested by the authors, which utilizes the last token as document representation, provides the best results and is the one reported in this section.

6. Discussion

We summarize in Fig. 9 the results of both baseline methods as well as our proposed approaches. The results demonstrate that our proposed methods can be quite a bit more effective than baselines trained on the "flattened" version of the datasets. We observed an improvement on the Linux Bugs dataset especially, and on the Financial dataset as well (though to a lesser extent), despite the latter not being strictly hierarchically labeled. Our findings confirm that models trained on the T2 task can benefit from the integration of information from a model specialized in the T1 task.

6.1. *Our models*

On the Linux Bugs dataset, our experiments show that ML-BERT and SupportedBERT respectively achieve 9.7% and 6.4% of F_1-score improvement over the flattened classifier with the best averaging strategy. The improvements in terms of accuracy instead amount to 5.4% (ML-BERT) and 7.0% (SupportedBERT). The results on the Financial dataset follow a similar trend; the two models achieve 11.3% and 3.8% improvement in F_1-score over the flattened classifier, while accuracy

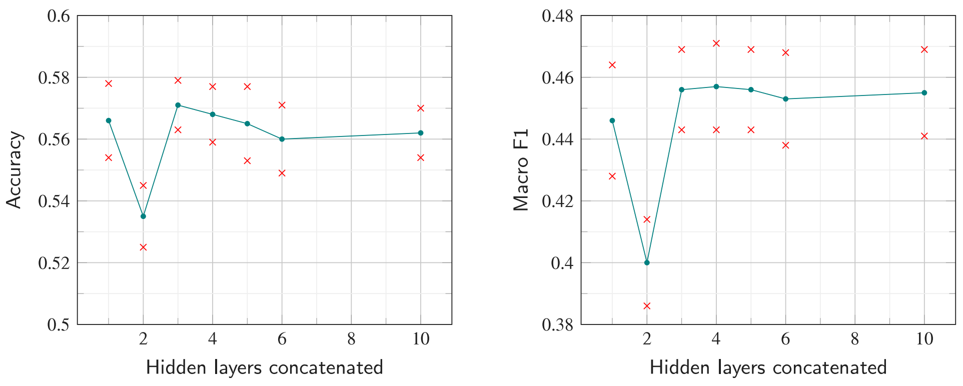


Fig. 8. Test set results on the Linux Bugs dataset, utilizing BERT fine-tuned with a linear classifier. The graph represents the change in accuracy (left) and macro F_1 (right) scores with the number of hidden layers concatenated for the *cls_concat* strategy. In this case, 1 stands for the raw `[CLS]` token (not pooled). The "x" marks represent the upper and lower bound given by standard deviation.

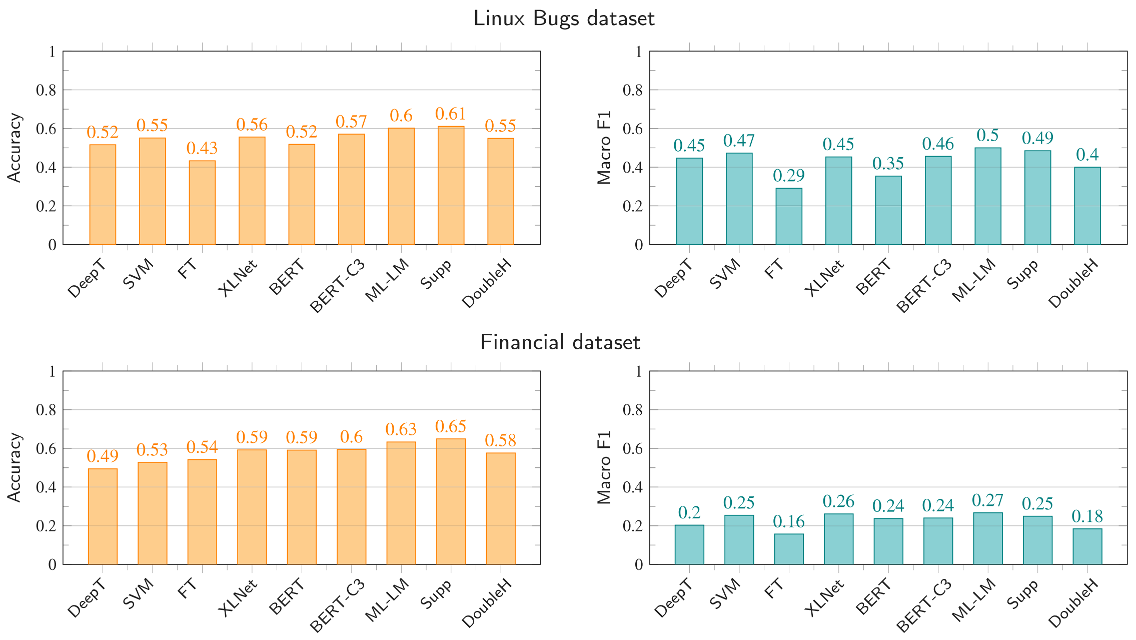


Fig. 9. Visual comparison between the tested methods in test set accuracy (left) and macro F_1 (right) for the Linux Bugs and Financial dataset. Abbreviations: DeepT = DeepTriage, FT = FastText, ML-LM/Supp/DoubleH = our proposed strategies. As before, BERT and XLNet refer to the standard usage of those models with a single-layer classifier head.

Table 10
Test set results[a] on baseline algorithms. Best results are outlined in bold.

Model	Acc	F_1	Prec	Rec
		Linux Bugs		
DeepTriage	0.516 [± 0.004]	0.447 [± 0.006]	0.493 [± 0.007]	0.432 [± 0.006]
SVM	0.551 [± 0.004]	**0.473** [± 0.006]	**0.552** [± 0.013]	**0.459** [± 0.006]
FastText	0.433 [± 0.003]	0.291 [± 0.003]	0.381 [± 0.014]	0.281 [± 0.002]
XLNet	0.556 [± 0.006]	0.453 [± 0.006]	0.480 [± 0.009]	0.453 [± 0.001]
BERT (*cls pooled*)	0.518 [± 0.006]	0.354 [± 0.009]	0.386 [± 0.009]	0.365 [± 0.006]
BERT (*cls concat$_3$*)	**0.571** [± 0.008]	0.456 [± 0.013]	0.498 [± 0.013]	0.458 [± 0.015]
		Financial		
DeepTriage	0.494 [± 0.027]	0.203 [± 0.011]	0.262 [± 0.023]	0.194[± 0.013]
SVM	0.528 [± 0.010]	0.254 [± 0.010]	**0.364** [± 0.019]	0.227 [± 0.007]
FastText	0.542 [± 0.006]	0.157 [± 0.022]	0.202 [± 0.028]	0.153 [± 0.019]
XLNet	0.592 [± 0.021]	**0.261** [± 0.030]	0.317 [± 0.030]	**0.258** [± 0.038]
BERT (*cls pooled*)	0.591 [± 0.004]	0.237 [± 0.011]	0.264 [± 0.021]	0.240 [± 0.012]
BERT (*cls concat$_3$*)	**0.595** [± 0.009]	0.240 [± 0.007]	0.290 [± 0.032]	0.242 [± 0.009]

[a]The standard deviation over the 6 runs is reported in brackets.

score improves by 6.4% and 9.1%, respectively. Overall, `ML-BERT` and `SupportedBERT` achieve the highest F_1 and accuracy scores among all baselines, with a slight tendency by `ML-BERT` towards higher F_1 and, conversely, by `SupportedBERT` to favor accuracy.

The best baseline method in terms of accuracy is the *cls_concat*$_3$ flattened classifier. In terms of F_1-score, the SVM classifiers are the best on the Linux Bugs datasets, while XLNet is the best on the Financial dataset. Still, `ML-BERT` outperforms them by 5.7% and 2.3%, in each dataset respectively. A notable exception can be seen in the performance of XLNet on the Financial dataset. When gauged against F_1-score, XLNet performs better than both the *cls_concat*$_3$ classifier and `SupportedBERT`, and only 2.2% worse than `ML-BERT`. The non-strict hierarchical structure of the Financial dataset most likely affects the performance of our framework, which targets the dependency among labels directly.

The performance of `DoubleHeadBERT` is overall lackluster. In this regard, we can assume that the embeddings contain the most useful information classification-wise. Our experiments suggest that combining the classification output (*i.e.*, the logits) of two separate classifiers does not provide enough semantic information for a third classifier to make the required adjustments to the prediction.

6.2. Baselines

Performance metrics across baselines are quite close; in some cases, we found that more recent approaches would perform worse than the SVM-based classifier, which instead performed remarkably well on both datasets. The most crucial advantage that we would expect BERT to have over models based on traditional text representations is the ability to extract more expressive features that can embed both contextual and sequential information from the tokens. However, the Linux Bugs dataset is very noisy, with many grammatical inconsistencies and technical readings, like stack traces or memory addresses. These likely make little sense for a LM pre-trained on more structured natural language. On the other hand, the SVM-based approach utilizes BoW features weighted with TF-IDF; therefore, the classifier only looks at global word frequency without considering any structural information. Indeed, it is conceivable that the strength of the SVM classifier can be explained by the lack of particular expressiveness in the structural information of these datasets. While the Financial dataset is less cluttered in terms of technical jargon, it is still rather noisy, while also being characterized by many structurally unsound sentences. Still, sentence structure is more expressive here, as demonstrated by the stronger performance of both our baseline LMs and our proposed approaches. Again, as mentioned in Section 4.1, the labeling of this dataset is not ideal for the task, which explains the poor performances in terms of precision and recall.

The FastText classifier obtained worse performances than other methods on both datasets. Given the amount of noise in the datasets we experimented with, it is possible that fine-tuning the embeddings before applying them to the classification task may improve the results of this approach — similar to what we do in our Transformer-based approaches.

DeepTriage obtains decent results, though not on par with BERT-based models. This was to be expected, as Transformers are capable of higher semantic and syntactic understanding as compared to recurrent models, and therefore create more meaningful representations for the documents. A noteworthy disadvantage of DeepTriage is indeed its recurrent nature; the computational expensiveness for training becomes quickly unmanageable when attempting to process longer sequences. While the original authors limit sentence length to 30 tokens, our preliminary tests showcased major improvements when allowing for longer sentences in the model, leading us to increase this threshold to 200 in our tests. The results are still noteworthy, as they are rather close to the best-performing flattened classifier (more so in terms of F_1-score).

In terms of framework, the approach based on XLNet is quite similar to the standard BERT approach. In our experiments, it was only tested on the flattened dataset, but it can be adapted for use with the proposed Multi-Layer architectures. However, XLNet is not pre-trained on a NSP task like BERT, and has no `[CLS]` token ready for classification; therefore, one of the summarization strategies outlined in Appendix A should be used. In general, we observe that XLNet performs better than BERT with the *cls pooled* strategy, and its metrics are very similar to the one obtained with BERT's *cls last* method. XLNet likely suffers from the same issues that BERT has on these datasets, *i.e.*, is hindered by technical jargon and very concise sentence formulation. Still, as mentioned before, the model manages to beat our `SupportedBERT` approach on the Financial dataset in terms of F_1-score, and is rather close to `ML-BERT`. As the hierarchical structure of this dataset is rather weak, this likely showcases that a better understanding of the documents (*i.e.*, better document embeddings) is more important than integrating the already inconsistent structure of the labels.

6.3. Document embedding summarization strategies

The choice of document embedding summarization strategy has a considerable impact on classification performance, a fact that is confirmed by our tests on the Linux Bugs dataset (results in Table 8 and Fig. 7). Using the *cls concat*$_3$ strategy improves F_1-score and accuracy by 28.8% and 10.2% with respect to the standard classification method, which utilizes the `[CLS]` token after pooling from the last hidden layer only. We point out that the "raw" `[CLS]` token (not pooled) is superior to its pooled counterpart on this particular dataset; we have discussed the reasons behind the usage of such "pooled embeddings" in Section 4.3. We also observe that both the *avg last* and *avg concat*$_2$ strategies improve in precision and recall over the widely used *cls pooled* approach, with the latter being the second best strategy for F_1-score. However, in our experience, the *avg* strategy does not seem to work well with an approach based on concatenating the average of multiple hidden layers, and therefore only report the result obtained by concatenating two hidden layers.

The aim of our experiments was to measure the importance of the summarization strategies for word embeddings. By analyzing combination strategies based on normalized sums, averages, and other operations, we sought to verify whether we could effectively "follow a path" in the high-dimensional vectorial space to obtain a meaningful representation for a document. In more detail, word embeddings can be seen as points in a n-dimensional space; sequences of words can then be interpreted as a path of oriented vectors in the same space. Under this assumption, averaging embeddings is a reasonable way to obtain a single document vector representing the overall direction of a sequence of words. However, an average also accounts for the magnitude of word embeddings. To reduce its importance and focus on the direction of vectors, we also normalize the sum of vectors in two different ways. We obtain worse results, suggesting that the length of vectors should be considered for a good document representation. We also report results using other approaches that have been used in the literature, like maximum and minimum, even if we find the geometrical interpretation of these operations to be less theoretically justifiable.

An interesting takeaway of these results is the effectiveness of the `[CLS]` token as a summarization of the document in BERT models. Moreover, this is true for all (or at least, a majority of) hidden layers of the model, as demonstrated by the effectiveness of combining multiple `[CLS]` tokens. Overall, we found the difference in results utilizing different summarization strategies to be quite striking. As we mentioned, some authors have suggested the ability of different hidden layers to capture more "specialized" linguistic features (de Vries et al., 2020; Jawahar et al., 2019). On these grounds, it is possible to hypothesize that providing the information from multiple hidden layers allows the model to understand these specialized features better, therefore leading to better classification. Upon examining Fig. 8, however, we observe a

Table A.11
Test set results[a] with XLNet classifier on T2 comparing averaging strategies on the Linux Bugs dataset.

Strategy	Acc	F_1	Prec	Rec
last	0.556 [± 0.006]	0.453 [± 0.006]	0.480 [± 0.009]	0.453 [± 0.001]
first	0.383 [± 0.299]	0.299 [± 0.257]	0.313 [± 0.264]	0.304 [± 0.262]
mean	**0.558** [± 0.152]	**0.456** [± 0.118]	**0.488** [± 0.132]	0.455 [± 0.115]
cls_index	0.556 [± 0.236]	0.451 [± 0.199]	0.478 [± 0.205]	**0.459** [± 0.206]

[a]The standard deviation over the 3 runs is reported in brackets.

peculiar trend in performance, with the addition of more layers failing to provide a steady improvement. This could be attributed to *(a)* the fact that later hidden layers represent more useful features, or *(b)* that concatenation (as well as our other tested strategies) is not the ideal approach in combining the information provided by these layers.

6.4. Future work

Many of the points discussed in this section lead to fascinating questions, many of which we would like to explore in future works. First of all, other LMs can be used with the `SupportedLM` and `ML-LM` architectures; it would be interesting to verify further the effectiveness of these frameworks. As an example, ByT5 is a model which feeds raw bytes directly to the LM, effectively bypassing many issues of character- and word-based models (Xue et al., 2022). As the vocabulary is based on UTF-8 bytes, this approach is much less affected by OOV issues, which are particularly relevant for applications to noisy texts, like support tickets. Other pre-trained models could also be tested, especially those specialized in shorter sequences of text. However, recent research seems to suggest that larger, more general LMs trained on huge corpora perform better in downstream tasks (Gasparetto et al., 2022), regardless of sentence length. In this regard, we would also like to investigate the performance of larger LMs over both the flattened and hierarchical classifiers, such as to determine whether the injection of hierarchical information can scale up as well.

Another interesting point to expand on is related to the significance of hidden layers within contextualized LMs. While many works already explore these aspects (Schiavinato, Gasparetto, & Torsello, 2015; Tenney et al., 2020), it would be interesting to attempt to pinpoint the significance (or an approximation) of these features in terms of semantic representation, such as to understand how they can improve downstream task performance. Moreover, advanced tools that allow in-depth analysis of LMs such as the Language Interpretability Tool (Tenney et al., 2020), Errudite (Wu, Ribeiro, Heer, & Weld, 2019), and iSEA (Yuan, Vig, & Rajani, 2022) have recently been developed and made available, and would allow to perform a meaningful (in terms of word and sentence semantics) error analysis of the models.

Finally, we point out that the supervised approach may not be the most suitable for real-world applications, since companies may not know the set of target labels or may want to change it dynamically. Hence, it would be interesting to compare the effectiveness of clustering methods in a hierarchical setting such as ours. Document embeddings could be generated with several strategies and then clustered in a desired number of label groups. The best possible assignment between target labels and clusters could be sought, and the procedure could be applied again within each cluster to match sub-labels.

7. Conclusions

In this article we provide an up-to-date view of recent research in the field of Ticket Automation, categorizing the current literature depending on the sub-task it aims to solve. Specifically, we identify four sub-tasks that are commonly applied to support tickets. Then, we delve into one of these sub-tasks, that of ticket classification, which aims to assign a topical categorization to tickets in order to speed up their resolution. We explore the application of contextualized LMs – in particular, BERT and XLNet – on two public hierarchical TC datasets. The first contains bug reports crawled from a notable bug-reporting website, and has a more meaningful hierarchy to its labels. The second is a collection of anonymized customer requests sent to financial companies, where the labels are less well-structured. We explore the usage of the BERT model for classification with several strategies to produce document-level summaries from word embeddings. Our results on both datasets show that the chosen embedding strategy can have a considerable impact on the reported metrics. As such, the best one should be determined using a validation procedure, akin to other hyperparameters. Moreover, we test three multi-level classifiers based on BERT that we use to predict hierarchically-dependent labels, and show how two of our proposed model-agnostic frameworks solidly improve results over the flattened classifiers.

CRediT authorship contribution statement

Alessandro Zangari: Conceptualization, Methodology, Software, Data curation, Writing – original draft. **Matteo Marcuzzo:** Conceptualization, Methodology, Software, Visualization, Formal analysis, Writing – original draft. **Michele Schiavinato:** Conceptualization, Validation, Software, Investigation, Writing – original draft. **Andrea Gasparetto:** Project administration, Methodology, Resources, Supervision, Writing – review & editing. **Andrea Albarelli:** Conceptualization, Methodology, Validation, Supervision, Writing – review & editing.

Declaration of competing interest

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

Data availability

Code and datasets are linked within the manuscript.

Appendix A. Experiments with XLNet

To provide a fair representation of XLNet's classification capabilities, we tested a number of document representation strategies as provided by the HuggingFace (Wolf et al., 2020) library. In particular, the library provides a *SequenceSummary* module which allows for the following summarization strategies (quoted directly from the documentation):

- `last` – Take the last token hidden state (like XLNet);
- `first` – Take the first token hidden state (like BERT);
- `mean` – Take the mean of all tokens hidden states;
- `cls_index` – Supply a Tensor of classification token position (GPT/GPT-2).

As referenced, XLNet's default strategy is to utilize the last token as a summary of the sentence/document (by virtue of its auto-regressive nature) (Li, Choi, Lee, & Ahn, 2020; Pistellato et al., 2019; Yang et al., 2019). Similarly, in our experiments, we found this strategy to be the most stable and consistent. Table A.11 contains the results of our tests on these summarization strategies. While the embedding averaging strategy has slightly better results on average, it has a considerably higher standard deviation, hence proving to be less reliable. Overall, we find that only the *last* strategy performed consistently across different runs, while results with other strategies fluctuated substantially with different splits. This seems to agree with the considerations made by XL-Net's authors, *i.e.*, that the representation of the last token in a sentence is the best one to capture the global meaning of a document. The setup of the experiments is almost identical to the one presented in the main text, with the only difference being that the cross-validation procedure is not repeated twice, but rather performed only once (because of time constraints). The model is left to train for up to 5 epochs with early stopping (always stopping at the second or third epoch).

Appendix B. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.eswa.2023.119984.

References

Agarwal, S., Bandlamudi, J., Mandal, A., Ray, A., & Sridhara, G. (2020). Automated assignment of helpdesk email tickets: An AI lifecycle case study. *AI Magazine*, *41*(3), 45–62. http://dx.doi.org/10.1609/aimag.v41i3.5321.

Al-Hawari, F., & Barham, H. (2021). A machine learning based help desk system for IT service management. *Journal of King Saud University - Computer and Information Sciences*, *33*(6), 702–718. http://dx.doi.org/10.1016/j.jksuci.2019.04.001, URL https://www.sciencedirect.com/science/article/pii/S1319157819300515.

Ali Zaidi, S. S., Fraz, M. M., Shahzad, M., & Khan, S. (2022). A multiapproach generalized framework for automated solution suggestion of support tickets. *International Journal of Intelligent Systems*, *37*(6), 3654–3681. http://dx.doi.org/10.1002/int.22701.

Askari, A., Verberne, S., & Pasi, G. (2022). Expert finding in legal community question answering. In M. Hagen, S. Verberne, C. Macdonald, C. Seifert, K. Balog, K. Nørvåg, & V. Setty (Eds.), *Advances in information retrieval* (pp. 22–30). Cham: Springer International Publishing.

Asres, M. W., Mengistu, M. A., Castrogiovanni, P., Bottaccioli, L., Macii, E., Patti, E., et al. (2021). Supporting telecommunication alarm management system with trouble ticket prediction. *IEEE Transactions on Industrial Informatics*, *17*(2), 1459–1469. http://dx.doi.org/10.1109/TII.2020.2996942.

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. http://dx.doi.org/10.48550/arXiv.1409.0473, ArXiv.Org abs/1409.0473, arXiv:1409.0473.

Bannihatti Kumar, V., Yarramsetty, M., Sun, S., & Goel, A. (2021). SupportNet: Neural networks for summary generation and key segment extraction from technical support tickets. In *Proceedings of the 4th workshop on E-commerce and NLP* (pp. 164–173). Online: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/2021.ecnlp-1.20, URL https://aclanthology.org/2021.ecnlp-1.20.

Beckers, T., Frommholz, I., & Bönning, R. (2009). *Multi-facet classification of E-mails in a helpdesk scenario*. University of Bedfordshire Repository, URL https://core.ac.uk/reader/29820850.

Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency* (pp. 610–623). Association for Computing Machinery, http://dx.doi.org/10.1145/3442188.3445922.

Beneker, D., & Gips, C. (2017). Using clustering for categorization of support tickets. In M. Leyer (Ed.), *CEUR Workshop Proceedings*: *vol.1917*, *Lernen, wissen, daten, analysen (lwda) conference proceedings* (pp. 51–62). CEUR-WS.org, URL http://ceur-ws.org/Vol-1917/paper10.pdf.

Bhowmik, A., Paul, S., Usha Nandini, D., & Prince Mary, S. (2019). Study of the management of tickets in IT administration. In *Conference of 1st international conference on frontiers in materials and smart system technologies, Vol. 590, no. 1*. Institute of Physics Publishing, Article 012006. http://dx.doi.org/10.1088/1757-899X/590/1/012006.

Bird, S. (2006). NLTK: The natural language toolkit. In *Proceedings of the COLING/ACL 2006 interactive presentation sessions* (pp. 69–72). Sydney, Australia: Association for Computational Linguistics, http://dx.doi.org/10.3115/1225403.1225421.

Bitext Innovations (2019). Bitext's customer support dataset. URL https://blog.bitext.com/free-customer-support-dataset (Accessed 15 July 2022).

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching Word Vectors with Subword Information. *Transactions of the Association for Computational Linguistics*, *5*, 135–146. http://dx.doi.org/10.1162/tacl_a_00051.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on computational learning theory* (pp. 144–152). Association for Computing Machinery, http://dx.doi.org/10.1145/130385.130401.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., et al. (2020). Language models are few-shot learners. In *Advances in neural information processing systems, Vol. 33* (pp. 1877–1901). Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., et al. (2021). Extracting training data from large language models. In *30th USENIX security symposium* (pp. 2633–2650). USENIX Association, URL https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting.

Cho, K., van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, eighth workshop on syntax, semantics and structure in statistical translation* (pp. 103–111). Doha, Qatar: Association for Computational Linguistics, http://dx.doi.org/10.3115/v1/W14-4012, URL https://aclanthology.org/W14-4012.

Chollet, F., et al. (2015). Keras. https://keras.io.

Cortes, C., & Vapnik, V. N. (1995). Support-vector networks. *Machine Learning*, *20*(3), http://dx.doi.org/10.1023/A:1022627411411.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*(1), 21–27. http://dx.doi.org/10.1109/TIT.1967.1053964.

Cristian, M., Christian, S., & Dumitru-Tudor, T. (2019). A study in the automation of service ticket recognition using natural language processing. In *2019 international conference on software, telecommunications and computer networks* (pp. 1–6). http://dx.doi.org/10.23919/SOFTCOM.2019.8903676.

Cunha, W., Mangaravite, V., Gomes, C., Canuto, S., Resende, E., Nascimento, C., et al. (2021). On the cost-effectiveness of neural and non-neural approaches and representations for text classification: A comprehensive comparative study. *Information Processing & Management*, *58*(3), Article 102481. http://dx.doi.org/10.1016/j.ipm.2020.102481.

de Vries, W., van Cranenburgh, A., & Nissim, M. (2020). What's so special about BERT's layers? A closer look at the NLP pipeline in monolingual and multilingual models. In *Findings of the association for computational linguistics: EMNLP 2020* (pp. 4339–4350). Online: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/2020.findings-emnlp.389, URL https://aclanthology.org/2020.findings-emnlp.389.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Minneapolis, Minnesota: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/N19-1423, URL https://aclanthology.org/N19-1423.

Fallahnejad, Z., & Beigy, H. (2022). Attention-based skill translation models for expert finding. *Expert Systems with Applications*, *193*, Article 116433. http://dx.doi.org/10.1016/j.eswa.2021.116433.

Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, *23*(120), 1–39, URL http://jmlr.org/papers/v23/21-0998.html.

Ferland, N., Sun, W., Fan, X., Yu, L., & Yang, J. (2020). Automatically resolve trouble tickets with hybrid NLP. In *2020 IEEE symposium series on computational intelligence* (pp. 1334–1340). IEEE, http://dx.doi.org/10.1109/SSCI47803.2020.9308327.

Fuchs, S., Drieschner, C., & Wittges, H. (2022). Improving support ticket systems using machine learning: A literature review. In *Proceedings of the 55th hawaii international conference on system sciences* (pp. 1893–1902). Honolulu, HI 96822: ScholarSpace, URL https://hdl.handle.net/10125/79570.

Gamboa, C. F. G., Concepcion, M. B., Alipio, A. J., Cortez, D. M. A., Bitancor, A. G., Santos, M. S., et al. (2022). Further enhancement of KNN algorithm based on clustering applied to IT support ticket routing. In *2022 3rd international conference on computing, networks and internet of things* (pp. 186–190). http://dx.doi.org/10.1109/CNIOT55862.2022.00040.

Gasparetto, A., Cosmo, L., Rodola, E., Bronstein, M., & Torsello, A. (2017). Spatial maps: From low rank spectral to sparse spatial functional representations. In *2017 international conference on 3D vision* (pp. 477–485). http://dx.doi.org/10.1109/3DV.2017.00061.

Gasparetto, A., Marcuzzo, M., Zangari, A., & Albarelli, A. (2022). A survey on text classification algorithms: From text to predictions. *Information*, *13*(2), http://dx.doi.org/10.3390/info13020083.

Gasparetto, A., Ressi, D., Bergamasco, F., Pistellato, M., Cosmo, L., Boschetti, M., et al. (2018). Cross-dataset data augmentation for convolutional neural networks training. In *2018 24th international conference on pattern recognition* (pp. 910–915). http://dx.doi.org/10.1109/ICPR.2018.8545812.

Gharebagh, S. S., Rostami, P., & Neshati, M. (2018). T-shaped mining: A novel approach to talent finding for agile software teams. In G. Pasi, B. Piwowarski, L. Azzopardi, & A. Hanbury (Eds.), *Advances in information retrieval* (pp. 411–423). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-76941-7_31.

Ghasemi, N., Fatourechi, R., & Momtazi, S. (2021a). User embedding for expert finding in community question answering. *ACM Transactions on Knowledge Discovery from Data*, *15*(4), http://dx.doi.org/10.1145/3441302.

Ghasemi, N., Fatourechi, R., & Momtazi, S. (2021b). User embedding for expert finding in community question answering. *ACM Transactions on Knowledge Discovery from Data*, *15*(4), http://dx.doi.org/10.1145/3441302.

GitHub. com (2020). LTI_Citi_Hackathon_My_ML. URL https://github.com/umeshdeorukhkar/LTI_Citi_Hackathon_My_ML (Accessed 15 July 2022).

Gupta, H. S., & Sengupta, B. (2012). Scheduling service tickets in shared delivery. In C. Liu, H. Ludwig, F. Toumani, & Q. Yu (Eds.), *Service-oriented computing* (pp. 79–95). Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-642-34321-6_6.

Han, J., Goh, K. H., Sun, A., & Akbari, M. (2018). Towards effective extraction and linking of software mentions from user-generated support tickets. In *Proceedings of the 27th ACM international conference on information and knowledge management* (pp. 2263–2271). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3269206.3272026.

Han, J., Li, J., & Sun, A. (2020). UFTR: A unified framework for ticket routing. http://dx.doi.org/10.48550/ARXIV.2003.00703, ArXiv.

Han, J., & Sun, A. (2017). Mean average distance to resolver: An evaluation metric for ticket routing in expert network. In *2017 IEEE international conference on software maintenance and evolution* (pp. 594–602). http://dx.doi.org/10.1109/ICSME.2017.18.

Han, J., & Sun, A. (2020). DeepRouting: A deep neural network approach for ticket routing in expert network. In *2020 IEEE international conference on services computing* (pp. 386–393). http://dx.doi.org/10.1109/SCC49832.2020.00057.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition, Vol. 1* (pp. 278–282). http://dx.doi.org/10.1109/ICDAR.1995.598994.

Husain, O., Salim, N., Alias, R. A., Abdelsalam, S., & Hassan, A. (2019). Expert finding systems: A systematic review. *Applied Sciences*, *9*(20), http://dx.doi.org/10.3390/app9204250, URL https://www.mdpi.com/2076-3417/9/20/4250.

Javed, T. A., Shahzad, W., & Arshad, M. U. (2021). Hierarchical text classification of Urdu news using deep neural network. http://dx.doi.org/10.48550/arXiv.2107.03141, CoRR abs/2107.03141, arXiv:2107.03141.

Jawahar, G., Sagot, B., & Seddah, D. (2019). What does BERT learn about the structure of language? In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 3651–3657). Florence, Italy: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/P19-1356, URL https://aclanthology.org/P19-1356.

Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal Document*, *28*(1), 11–21. http://dx.doi.org/10.1108/eb026526.

Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th conference of the european chapter of the association for computational linguistics: Volume 2, Short Papers* (pp. 427–431). Valencia, Spain: Association for Computational Linguistics, URL https://aclanthology.org/E17-2068.

Jurafsky, D., & Martin, J. (2020). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition* (3rd (draft) ed.). (pp. 30–35). URL https://web.stanford.edu/~jurafsky/slp3.

Kallis, R., Di Sorbo, A., Canfora, G., & Panichella, S. (2019). Ticket tagger: Machine learning driven issue classification. In *2019 IEEE international conference on software maintenance and evolution* (pp. 406–409). http://dx.doi.org/10.1109/ICSME.2019.00070.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., et al. (2020). Scaling laws for neural language models. http://dx.doi.org/10.48550/arXiv.2001.08361, ArXiv abs/2001.08361, arXiv:2001.08361.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1746–1751). Association for Computational Linguistics, http://dx.doi.org/10.3115/v1/D14-1181.

Kiritchenko, S., Matwin, S., Nock, R., & Famili, A. F. (2006). Learning and evaluation in the presence of class hierarchies: Application to text categorization. In L. Lamontagne, & M. Marchand (Eds.), *Advances in artificial intelligence* (pp. 395–406). Berlin, Heidelberg: Springer Berlin Heidelberg.

Klimt, B., & Yang, Y. (2004). The enron corpus: A new dataset for email classification research. In J.-F. Boulicaut, F. Esposito, F. Giannotti, & D. Pedreschi (Eds.), *Machine learning: ECML 2004* (pp. 217–226). Berlin, Heidelberg: Springer Berlin Heidelberg, http://dx.doi.org/10.1007/978-3-540-30115-8_22.

Koller, D., & Sahami, M. (1997). Hierarchically classifying documents using very few words. In *Proceedings of the fourteenth international conference on machine learning* (pp. 170–178). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc..

Kowsari, K., Jafari Meimandi, K., Heidarysafa, M., Mendu, S., Barnes, L., & Brown, D. (2019). Text classification algorithms: A survey. *Information*, *10*(4), http://dx.doi.org/10.3390/info10040150.

Kubiak, P., & Rass, S. (2018). An overview of data-driven techniques for IT-service-management. *IEEE Access*, *6*, 63664–63688. http://dx.doi.org/10.1109/ACCESS.2018.2875975.

Kundu, D., Pal, R. K., & Mandal, D. P. (2021). Topic sensitive hybrid expertise retrieval system in community question answering services. *Knowledge-Based Systems*, *211*(106535), http://dx.doi.org/10.1016/j.knosys.2020.106535.

Labrou, Y., & Finin, T. (1999). Yahoo! as an ontology: Using yahoo! categories to describe documents. In *Proceedings of the eighth international conference on information and knowledge management* (pp. 180–187). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/319950.319976.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., et al. (2021). GShard: Scaling giant models with conditional computation and automatic sharding. In *International conference on learning representations* (pp. 1–23). http://dx.doi.org/10.48550/arXiv.2006.16668, arXiv:2006.16668.

Li, H., Choi, J., Lee, S., & Ahn, J. H. (2020). Comparing BERT and XLNet from the perspective of computational characteristics. In *2020 international conference on electronics, information, and communication* (pp. 1–4). http://dx.doi.org/10.1109/ICEIC49074.2020.9051081.

Li, Z., Jiang, J.-Y., Sun, Y., & Wang, W. (2019). Personalized question routing via heterogeneous network embedding. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 33, no. 1* (pp. 192–199). http://dx.doi.org/10.1609/aaai.v33i01.3301192, URL https://ojs.aaai.org/index.php/AAAI/article/view/3785.

Li, Q., Peng, H., Li, J., Xia, C., Yang, R., Sun, L., et al. (2020). A survey on text classification: From shallow to deep learning. CoRR abs/2008.00364, arXiv:2008.00364.

Li, B., Yu, S., & Lu, Q. (2003). An improved k-nearest neighbor algorithm for text categorization. http://dx.doi.org/10.48550/arXiv.cs/0306099, CoRR cs.CL/0306099.

Lin, S., Hong, W., Wang, D., & Li, T. (2017). A survey on expert finding techniques. *Journal of Intelligent Information Systems*, *49*(2), 255–279. http://dx.doi.org/10.1007/s10844-016-0440-5.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., et al. (2019). RoBERTa: A robustly optimized BERT pretraining approach. http://dx.doi.org/10.48550/arXiv.1907.11692, CoRR abs/1907.11692, arXiv:1907.11692.

Liu, Y., Tang, W., Liu, Z., Ding, L., & Tang, A. (2022). High-quality domain expert finding method in CQA based on multi-granularity semantic analysis and interest drift. *Information Sciences*, *596*, 395–413. http://dx.doi.org/10.1016/j.ins.2022.02.039.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1412–1421). Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/D15-1166.

Lyubinets, V., Boiko, T., & Nicholas, D. (2018). Automated labeling of bugs and tickets using attention-based mechanisms in recurrent neural networks. In *2018 IEEE second international conference on data stream mining & processing* (pp. 271–275). http://dx.doi.org/10.1109/DSMP.2018.8478511.

Madaan, N., Singh, G., Kumar, A., & Dasgupta, G. B. (2017). Neev: A cognitive support agent for content improvement in hardware tickets. In *2017 IFIP/IEEE symposium on integrated network and service management* (pp. 239–246). http://dx.doi.org/10.23919/INM.2017.7987285.

Mandal, A., Agarwal, S., Malhotra, N., Sridhara, G., Ray, A., & Swarup, D. (2019). Improving IT support by enhancing incident management process with multi-modal analysis. In S. Yangui, I. Bouassida Rodriguez, K. Drira, & Z. Tari (Eds.), *Service-oriented computing* (pp. 431–446). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-33702-5_33.

Mandal, A., Malhotra, N., Agarwal, S., Ray, A., & Sridhara, G. (2019). Automated dispatch of helpdesk email tickets: Pushing the limits with AI. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 33, no. 01* (pp. 9381–9388). http://dx.doi.org/10.1609/aaai.v33i01.33019381.

Mani, S., Sankaran, A., & Aralikatte, R. (2019). DeepTriage: Exploring the effectiveness of deep learning for bug triaging. In *CoDS-COMAD '19, Proceedings of the ACM india joint international conference on data science and management of data* (pp. 171–179). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3297001.3297023.

Marcus, M. P., Santorini, B., Marcinkiewicz, M. A., & Taylor, A. (1999). Treebank-3. In *Linguistic data consortium, Vol. 14*. Philadelphia.

Marcuzzo, M., Zangari, A., Albarelli, A., & Gasparetto, A. (2022). Recommendation systems: An insight into current development and future research challenges. *IEEE Access*, http://dx.doi.org/10.1109/ACCESS.2022.3194536.

Meng, J., Li, Y., Liu, C., Dong, Y., Wang, Z., & Zhang, Y. (2021). Classification of customer service tickets in power system based on character and word level semantic understanding. In *2021 china international conference on electricity distribution* (pp. 1062–1066). http://dx.doi.org/10.1109/CICED50259.2021.9556759.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *1st international conference on learning representations* (pp. 1–12). http://dx.doi.org/10.48550/arXiv.1301.3781, arXiv:1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems, Vol. 26* (pp. 3111–3119). Curran Associates, Inc., URL https://proceedings.neurips.cc/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf.

Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., & Gao, J. (2021). Deep learning–based text classification: A comprehensive review. *ACM Computing Surveys*, *54*(3), http://dx.doi.org/10.1145/3439726.

Miraj, R., & Aono, M. (2021). Combining BERT and multiple embedding methods with the deep neural network for humor detection. In H. Qiu, C. Zhang, Z. Fei, M. Qiu, & S.-Y. Kung (Eds.), *Knowledge science, engineering and management* (pp. 53–61). Cham: Springer International Publishing.

Montgomery, L., Damian, D., Bulmer, T., & Quader, S. (2018). Customer support ticket escalation prediction using feature engineering. *Requirements Engineering*, *23*(3), 333–355. http://dx.doi.org/10.1007/s00766-018-0292-3.

Mukunthan, M., & Selvakumar, S. (2019). Multilevel Petri net-based ticket assignment and IT management for improved IT organization support. *Concurrency Computations: Practice and Experience*, *31*(14), Article e5297. http://dx.doi.org/10.1002/cpe.5297, e5297 CPE-18-0330.R1.

Paramesh, S., Ramya, C., & Shreedhara, K. (2018). 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* (pp. 221–227). http://dx.doi.org/10.1109/CSITSS.2018.8768734.

Paramesh, S. P., & Shreedhara, K. S. (2019). Automated IT service desk systems using machine learning techniques. In P. Nagabhushan, D. S. Guru, B. H. Shekar, & Y. H. S. Kumar (Eds.), *Data analytics and learning* (pp. 331–346). Singapore: Springer Singapore, http://dx.doi.org/10.1007/978-981-13-2514-4_28.

Parmar, P., Biju, P., Shankar, M., & Kadiresan, N. (2018). Multiclass text classification and analytics for improving customer support response through different classifiers. In *7th international conference on advances in computing, communications and informatics* (ICACCI 2018), (pp. 538–542). Institute of Electrical and Electronics Engineers Inc., http://dx.doi.org/10.1109/ICACCI.2018.8554881.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc., URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Peng, Q., Liu, H., Wang, Y., Xu, H., Jiao, P., Shao, M., et al. (2022). Towards a multi-view attentive matching for personalized expert finding. In *Proceedings of the ACM web conference 2022* (pp. 2131–2140). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3485447.3512086.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1532–1543). Association for Computational Linguistics, http://dx.doi.org/10.3115/v1/D14-1162.

Perez, Q., Jean, P.-A., Urtado, C., & Vauttier, S. (2021). Bug or not bug? That is the question. In *2021 IEEE/ACM 29th international conference on program comprehension* (pp. 47–58). http://dx.doi.org/10.1109/ICPC52881.2021.00014, cited By.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., et al. (2018). Deep contextualized word representations. In *Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics: human language technologies, Vol. 1* (pp. 2227–2237). Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/N18-1202.

Pikies, M., & Ali, J. (2019). String similarity algorithms for a ticket classification system. In *2019 6th international conference on control, decision and information technologies* (pp. 36–41). http://dx.doi.org/10.1109/CoDIT.2019.8820497.

Pistellato, M., Bergamasco, F., Albarelli, A., Cosmo, L., Gasparetto, A., & Torsello, A. (2019). Robust phase unwrapping by probabilistic consensus. *Optics and Lasers in Engineering*, *121*, 428–440. http://dx.doi.org/10.1016/j.optlaseng.2019.05.006.

Pistellato, M., Cosmo, L., Bergamasco, F., Gasparetto, A., & Albarelli, A. (2018). Adaptive Albedo compensation for accurate phase-shift coding. In *2018 24th international conference on pattern recognition* (pp. 2450–2455).

Polato, M. (2017). Dataset belonging to the help desk log of an Italian Company. http://dx.doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb, URL https://data.4tu.nl/articles/Dataset_belonging_to_the_help_desk_log_of_an_Italian_Company/12675977.

Powell, M., Rotz, J. A., & O'Malley, K. D. (2020). How machine learning is improving U.S. navy customer support. In *Proceedings of the AAAI conference on artificial intelligence, Vol. 34, no. 08* (pp. 13188–13195). http://dx.doi.org/10.1609/aaai.v34i08.7023, URL https://ojs.aaai.org/index.php/AAAI/article/view/7023.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., et al. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, *21*(140), 1–67, URL http://jmlr.org/papers/v21/20-074.html.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing* (pp. 3982–3992). Hong Kong, China: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/D19-1410.

Revina, A., Buza, K., & Meister, V. G. (2020). IT ticket classification: The simpler, the better. *IEEE Access*, *8*, 193380–193395. http://dx.doi.org/10.1109/ACCESS.2020.3032840.

Ricciardi Celsi, L., Caliciotti, A., D'Onorio, M., Scocchi, E., Sulieman, N. A., & Villari, M. (2021). On predicting ticket reopening for improving customer service in 5G fiber optic networks. *Future Internet*, *13*(10), URL https://www.mdpi.com/1999-5903/13/10/259.

Rostami, P., & Neshati, M. (2021). Intern retrieval from community question answering websites: A new variation of expert finding problem. *Expert Systems with Applications*, *181*(115044), http://dx.doi.org/10.1016/j.eswa.2021.115044, URL https://www.sciencedirect.com/science/article/pii/S0957417421004851.

Safavian, S., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, *21*(3), 660–674. http://dx.doi.org/10.1109/21.97458.

Schiavinato, M., Gasparetto, A., & Torsello, A. (2015). Transitive assignment kernels for structural classification. In A. Feragen, M. Pelillo, & M. Loog (Eds.), *Similarity-based pattern recognition, Vol. 9370* (pp. 146–159). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-319-24261-3_12.

Schuster, M., & Nakajima, K. (2012). Japanese and Korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing* (pp. 5149–5152). http://dx.doi.org/10.1109/ICASSP.2012.6289079.

Shao, Q., Chen, Y., Tao, S., Yan, X., & Anerousis, N. (2008). Efficient ticket routing by resolution sequence mining. In *Proceedings of the 14th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 605–613). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/1401890.1401964.

Stack Exchange Inc. (2022). *Stack exchange data dump*. Internet Archive, URL https://archive.org/details/stackexchange (Accessed 15 July 2022).

Subbarao, M., Venkatarao, K., & Suresh, C. (2022). Automation of incident response and it ticket management by ML and NLP mechanisms. *Journal of Theoretical and Applied Information Technology*, *100*(12), 3945–3955, URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85134195529&partnerID=40&md5=5a554db8c40cde16751e960131386267.

Sundaramahadevan, V. (2022). Automatic ticket classification. URL https://www.kaggle.com/datasets/venkatasubramanian/automatic-ticket-classification (Accessed 15 July 2022).

Sunil, K. (2020). AMS Ticket Data. URL https://www.kaggle.com/datasets/karthikcs1/amsticketdata (Accessed 15 July 2022).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th international conference on neural information processing systems - volume 2* (pp. 3104–3112). Cambridge, MA, USA: MIT Press.

Sutton, C., & McCallum, A. (2012). An introduction to conditional random fields. *Foundation Trends in Machine Learning*, *4*(4), 267–373. http://dx.doi.org/10.1561/2200000013.

Tanaka, H., Shinnou, H., Cao, R., Bai, J., & Ma, W. (2019). Document classification by word embeddings of BERT. In L.-M. Nguyen, X.-H. Phan, K. Hasida, & S. Tojo (Eds.), *16th international conference of the pacific association for computational linguistics* PACLING 2019, (pp. 145–154). Hanoi, Vietnam: Springer Singapore, http://dx.doi.org/10.1007/978-981-15-6168-9_13.

Tenney, I., Wexler, J., Bastings, J., Bolukbasi, T., Coenen, A., Gehrmann, S., et al. (2020). The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 107–118). Online: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/2020.emnlp-demos.15.

Tolciu, D.-T., Săcărea, C., & Matei, C. (2021). Analysis of patterns and similarities in service tickets using natural language processing. *Journal of Communications Software and Systems*, *17*(1), 29–35. http://dx.doi.org/10.24138/JCOMSS.V17I1.1024.

Torsello, A., Gasparetto, A., Rossi, L., Bai, L., & Hancock, E. (2014). Transitive state alignment for the quantum Jensen-Shannon kernel. *Lecture Notes in Computer Science*, *8621*, 22–31. http://dx.doi.org/10.1007/978-3-662-44415-3_3.

van den Bosch, A. (2017). Hidden Markov models. In *Encyclopedia of machine learning and data mining* (pp. 609–611). Boston, MA: Springer US, http://dx.doi.org/10.1007/978-1-4899-7687-1_124.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 6000–6010). Red Hook, NY, USA: Curran Associates Inc..

Wang, Q., Li, T., Iyengar, S. S., Shwartz, L., & Grabarnik, G. Y. (2018). Online IT ticket automation recommendation using hierarchical multi-armed bandit algorithms. In *Proceedings of the 2018 SIAM international conference on data mining* (pp. 657–665). http://dx.doi.org/10.1137/1.9781611975321.74.

Wang, Q., Shwartz, L., Grabarnik, G. Y., Nidd, M., & Hwang, J. (2019). Leveraging AI in service automation modeling: From classical AI through deep learning to combination models. In S. Yangui, I. Bouassida Rodriguez, K. Drira, & Z. Tari (Eds.), *Service-oriented computing* (pp. 186–201). Cham: Springer International Publishing, http://dx.doi.org/10.1007/978-3-030-33702-5_14.

Wang, Q., Zeng, C., Iyengar, S. S., Li, T., Shwartz, L., & Grabarnik, G. Y. (2018). AISTAR: An intelligent system for online IT ticket automation recommendation. In *IEEE international conference on big data* (pp. 1875–1884). IEEE, http://dx.doi.org/10.1109/BigData.2018.8622446.

Watanabe, A., Ishibashi, K., Toyono, T., Watanabe, K., Kimura, T., Matsuo, Y., et al. (2018). Workflow extraction for service operation using multiple unstructured trouble tickets. *IEICE Transactions on Information and Systems*, *E101.D*(4), 1030–1041. http://dx.doi.org/10.1587/transinf.2017DAP0014.

Werner, C., Li, Z. S., & Damian, D. (2019). Can a machine learn through customer sentiment?: A cost-aware approach to predict support ticket escalations. *IEEE Software*, *36*(5), 38–45. http://dx.doi.org/10.1109/MS.2019.2923408.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations* (pp. 38–45). Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/2020.emnlp-demos.6.

Wu, T., Ribeiro, M. T., Heer, J., & Weld, D. (2019). Errudite: Scalable, reproducible, and testable error analysis. In *Proceedings of the 57th annual meeting of the association for computational linguistics* (pp. 747–763). Florence, Italy: Association for Computational Linguistics, http://dx.doi.org/10.18653/v1/P19-1073, URL https://aclanthology.org/P19-1073.

Xu, J., & He, R. (2018). Expert recommendation for trouble ticket routing. *Data & Knowledge Engineering*, *116*, 205–218. http://dx.doi.org/10.1016/j.datak.2018.06.004.

Xu, J., He, R., Zhou, W., & Li, T. (2018). Trouble ticket routing models and their applications. *IEEE Transactions on Network and Service Management*, *15*(2), 530–543. http://dx.doi.org/10.1109/TNSM.2018.2790956.

Xu, S., Li, Y., & Wang, Z. (2017). Bayesian multinomial naïve Bayes classifier to text classification. In *Advanced multimedia and ubiquitous engineering* (pp. 347–352). Springer Singapore, http://dx.doi.org/10.1007/978-981-10-5041-1_57.

Xu, J., Mu, J., & Chen, G. (2020). A multi-view similarity measure framework for trouble ticket mining. *Data & Knowledge Engineering*, *127*(101800), http://dx.doi.org/10.1016/j.datak.2020.101800.

Xu, J., Zhang, H., Zhou, W., He, R., & Li, T. (2018). Signature based trouble ticket classification. *Future Generation Computer Systems*, *78*, 41–58. http://dx.doi.org/10.1016/j.future.2017.07.054.

Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., et al. (2022). ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, *10*, 291–306. http://dx.doi.org/10.1162/tacl_a_00461.

Yang, L. (2021). Fuzzy output support vector machine based incident ticket classification. *IEICE Transactions on Information and Systems*, *E104.D*(1), 146–151. http://dx.doi.org/10.1587/transinf.2020EDP7044.

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized autoregressive pretraining for language understanding. In *Proceedings of the 33rd international conference on neural information processing systems* (pp. 5753–5763). Red Hook, NY, USA: Curran Associates Inc..

Young, I. J. B., Luz, S., & Lone, N. (2019). A systematic review of natural language processing for classification tasks in the field of incident reporting and adverse event analysis. *International Journal of Medical Informatics*, *132*(103971), http://dx.doi.org/10.1016/j.ijmedinf.2019.103971.

Yuan, J., Vig, J., & Rajani, N. (2022). ISEA: An interactive pipeline for semantic error analysis of NLP models. In *27th international conference on intelligent user interfaces* (pp. 878–888). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3490099.3511146.

Żak, K. (2018). Support-tickets-classification. URL https://github.com/karolzak/support-tickets-classification (Accessed 15 July 2022).

Zhou, W., Xue, W., Baral, R., Wang, Q., Zeng, C., Li, T., et al. (2017). STAR: A system for ticket analysis and resolution. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 2181–2190). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3097983.3098190.

Zicari, P., Folino, G., Guarascio, M., & Pontieri, L. (2021). Discovering accurate deep learning based predictive models for automatic customer support ticket classification. In *Proceedings of the 36th annual acm symposium on applied computing* (pp. 1098–1101). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3412841.3442109.

Alessandro Zangari is studying for his Ph.D. in Computer Science at the Ca’ Foscari University of Venice, doing research in the Machine Learning field. He received his Master’s Degree in Computer Science from the University of Padua in 2020. He is currently researching Deep Learning applications for Natural Language Processing algorithms and Recommendation Systems. He is also interested in Computer Vision, and the Interpretability of AI.

Matteo Marcuzzo is a Ph.D. student in Computer Science at the Ca’ Foscari University of Venice. He received a Bachelor’s Degree in Computer Games Technology from the University of the West of Scotland in Paisley, United Kingdom, and a Master’s Degree in Computer Science from the University of Padua, Italy. He is interested in Natural Language Processing, Representation learning, Interpretable, AI and High-Performance Computing.

Michele Schiavinato is an associate researcher for the Ca’ Foscari Department of Management working in the fields of machine learning, artificial intelligence, and computer vision. He started his academic career at the Ca’ Foscari University of Venice, receiving a Ph.D. in Computer Science. He was consequently granted a postdoc fellowship to work on several applied research projects. He has contributed to many projects which connect the academic and industrial world, providing software solutions for local companies. He has also joined the Odycceus European project developing and maintaining the Aqua Granda database.

Andrea Gasparetto received an MSc degree in Computer Science from the University of Venice, Italy, in 2012 and received his Ph.D. degree in computer science from the Ca’ Foscari University. Since 2016 he is a Researcher and Teaching Assistant at the Management Department of Ca’ Foscari University. His research interests are mainly in the artificial intelligence field, and more precisely in computer vision, shape analysis, retrieval and classification, and non-vectorial data models.

Andrea Albarelli is a researcher in the field of Artificial Intelligence, with a special focus on the design of disruptive data-driven methodologies to be applied to real-world scenarios. To this end, he works in close collaboration with companies willing to undertake a radical digital transformation process. His approach is end-to-end, spanning from the co-design of digital-first business models to the scientific advising needed to fulfill their methodological and technological infrastructure. He has led several technological transfer projects, resulting in research papers published in top international journals and presented at key Engineering conferences. He received several scientific and industrial recognitions, including the NVIDIA best paper award, for his research on 3D data processing, and innovation grants from companies like Electrolux and TIM, for their technical contributions. He is currently a professor for the multidisciplinary Master Program in Data Analytics for Business and Society at the Ca’ Foscari University of Venice, where he is responsible for Artificial Intelligence teaching.