An AI-Based Methodology for the Automatic Classification of a Multiclass Ebook Collection Using Information From the Tables of Contents

ELENI GIANNOPOLOU AND NIKOLAS MITROU, (Member, IEEE)
School of Electrical and Computer Engineering, National Technical University of Athens, 157 80 Athens, Greece
Corresponding author: Eleni Giannopoulou (egiann@cn.ntua.gr)

ABSTRACT Book recommendation to support professors and students in the identification of relevant sources is of significant importance for both universities and digital libraries and, hence, motivates the development of a recommendation system. This paper aims at automatically classifying a multiclass corpus that was created from ebooks from the Springer collection, which is available through the Hellenic Academic Libraries’ subscription, by utilizing an unsupervised neural network (NN) (self-organizing maps, SOM) and two deep neural network (DNN) architectures, namely, a long short-term memory (LSTM) and a convolutional neural network (CNN) combined with a LSTM(CNN+LSTM) under various configuration scenarios. The vector construction leverages information that was extracted from the table of contents (ToC) of each book using the TF-IDF weighting scheme (for the first case) and the Keras tokenizer (for the second). Extensive experiments were conducted using various configurations of preprocessing steps, NN set up and vector and vocabulary sizes to assess their impact on the classifier’s performance. Furthermore, we show that majority voting is more suitable for selecting the dominant label for a specified node. The experimental analysis showed the feasibility of developing a recommendation system for supporting professors and students in the identification of related sources based on a detailed thematic description (e.g., abstract or table of contents of a book) rather than a few keywords. In the conducted experiments, the subsystem that utilized the DNN (LSTM) performed the best, with F1-scores of 67% for the 26 categories and 80% for the 5 general categories, whereas SOM realizes F1-scores of less than 5% in both cases.

INDEX TERMS Artificial Intelligence, natural language processing, classification algorithms, self-organizing maps, unsupervised learning, deep neural networks, digital libraries, book classification, ToC vectorization.

I. INTRODUCTION
When searching for sources, professors and students often use their institution’s digital library and provide keywords that they believe to be relevant as query terms. However, the results that are returned by the library systems are often irrelevant as they may correspond to an entirely different topic or relevant books may be excluded from the results list if they do not contain the exact search term. This problem becomes more evident as the content of a digital library increases. The results are ranked based on the similarity of the search term with terms in each book. However, the use of a more detailed description of the theme under search (e.g., course description or the table of contents of a known book) may lead to more accurate results should a suitable classification and retrieval system be available based on similarity measures of such text components. In this paper, we model the problem of finding similar books as a multiclass classification problem and utilize information from the table of contents (ToC) for effective book recommendation.

The task of automatic document classification, especially multiclass classification, in which each document can be assigned only to one class, is important in various domains, including digital libraries. Various research studies have been conducted that focus on multiclass classification [1]–[7], [9], [10]. Most of the approaches aim at transforming the considered multiclass classification problem into a set of binary classification problems. In this transformation, for each category in the set, a binary classifier is created and trained on the dataset such that each classifier can distinguish...
whether a testing sample belongs to a specified category or not. Unfortunately, this approach suffices only under the assumption that the dataset is practically unchangeable, as for each new category added, a new classifier would need to be developed. In the Springer dataset that was selected in our case, ebooks can be added or removed and new categories may be introduced; therefore, the transformation to a set of binary classification problems poses severe difficulties.

Traditionally, the automatic categorization of a set of electronic documents into a set of categories, which are denoted by labels, is a supervised classification task [10], [26]. In this case, the classifier learns on the basis of labeled instances that are used during the training phase, and afterwards, the testing subset is used to assess its performance. However, this approach requires large, well-annotated corpora and is ineffective on datasets that have an unbalanced distribution of samples among the classes. Due to this limitation, unsupervised and deep learning approaches are typically selected over supervised approaches. Recently, pretrained language models such as Transformers, BERT, XLNet and GPT-2 have attracted attention. The models have been deemed effective compared to embeddings that are learned from scratch. Furthermore, transfer learning from supervised data can be applied as an antidote to the class imbalance problem, as will be analyzed in the Related Work section.

In this study, the methodology and the results of a strategy that combines two approaches are reported: a self-organizing maps (SOM) neural network (NN) and two deep neural network (DNN) architectures, namely, an LSTM and a CNN+LSTM. SOM is an unsupervised machine learning method that can cluster similar documents together using vector similarity measures without prior knowledge of the categories. In this case, the labels of the training instances will be used only to guide the assignment of labels of the testing dataset after the training phase. Another differentiating characteristic from the supervised techniques is that a trained SOM contains identifiable but nondelimited clusters [42]. In the created clusters, similar documents tend to share common labels; therefore, SOM inherently preserves the correlations between labels as related labels can be found either on the same or on a neighboring node on the map. Similarly, the DNNs use vector embeddings to cluster similar documents together in a low-dimensional vector space. In contrast to SOM, however, the selected DNN approach is a supervised approach since labels are used to specify the category of each sample during the training phase. The DNN approach was selected as it has been reported to realize high accuracy without extensive feature engineering.

Here, we compare these two NNs for the multiclass classification of a real-world dataset under various configuration options. Several grid sizes are compared, and various sets of parameters are utilized, namely, for the map size, vector size, weighting scheme, ToC max length, and vocabulary size, to show that such choices can significantly affect the classification performance. Based on this experimentation and the observations, we propose a methodology for automatic multiclass classification based solely on information from the ToC. To the best of our knowledge, this is the first attempt to conduct such a comparison on this task. Furthermore, to avoid extensive feature engineering, we used a simple set of features that depend on word statistics and can easily be applied on any dataset without modifications. Moreover, two algorithms were utilized to select the label to be assigned to each node on the map after training, namely, majority voting and the INterSECT algorithm, which is presented in [8] and selects labels to be assigned to a node based on the labels of neighboring nodes. As INterSECT is tailored to multilabel scenarios, we expected that majority voting would be more suitable for the multiclass problem that is addressed in this study. The contributions of our paper are as follows:

- We have created a dataset that consists of approximately 56K ebooks from Springer, which we make publicly available.
- We propose a methodology for automatically classifying a multiclass ebook collection using information from the ToC.
- We show that the selected feature selection process reduces the need for extensive feature engineering and is independent of changes to the dataset.
- We evaluate two methods for selecting the label for each node in the SOM scenario. We show that majority voting outperforms INterSECT.
- We extensively evaluate the methodology by utilizing an unsupervised NN (SOM) and two DNN architectures (LSTM and CNN+LSTM). It is proved that LSTM outperforms SOM by 5%.

The remainder of this paper is organized as follows: In Section II, recent approaches for multiclass document classification and feature selection and of classification approaches that use SOM and DNNs are reviewed. Section III describes the dataset that is used to conduct the experiments, and Section IV presents the proposed methodology in detail. Section V addresses the experimental evaluation, the conducted experiments, the evaluation measures, and the obtained results and compares and discusses the results. Finally, Section VI presents the conclusions of this study and provides thoughts on improvements, and Section VII describes potential extensions of this study.

II. RELATED WORK

A. MULTI-CLASS DOCUMENT CLASSIFICATION

The main objective of a document classification task is to allocate each item of a text corpus to one or more categories depending on whether it is a multiclass or a multilabel classification task [2]. Based on the training data, the system can classify previously unseen items to their corresponding categories. The main objective of this task is to identify an optimal classification scheme, namely, a scheme that reduces the dimensionality of the feature vectors while increasing the accuracy of the classifier [10].

In most approaches for allocating previously unseen documents to one or more categories, supervised machine learning...
techniques are utilized, as they are deemed more effective. A shortcoming of these approaches is that they require accurately annotated large-scale corpora with satisfactory balance among the classes [10]. The Springer ebook corpus, which is utilized in our case, is a multiclass corpus with an unbalanced distribution of samples among the classes; thus, challenges are encountered in the incorporation of supervised techniques.

Reference [1] proposed a pairwise multiclass document classification approach for identifying relationships between Wikipedia articles, and SCVD-MS was presented in [6], which utilized multisense embeddings to improve multiclass classification on the 20NewsGroup dataset\(^1\) while also targeting a lower dimensional representation compared to that of its predecessor SCDV. The 20NewsGroup dataset was also utilized in [2], in which an extension to the Word2Vec and FastText [12] word embedding algorithms is proposed. The word embeddings were augmented with semantic information by assigning a part-of-speech (POS) tag to each word with the objective of evaluating the enhanced model’s performance on a multiclass classification task.

Another approach to multiclass document classification was presented in [3], in which an improved Naïve Bayes vectorization technique, to realize dimensionality reduction, was applied in combination with an SVM classifier. Similarly, [4] presented a multiclass classification model that was based on quantum detection theory. Reference [5] investigated the task of multilingual multiclass sentiment classification using convolutional neural networks on 3 datasets in 3 languages. According to the authors, the discriminative factor in their study is that the produced model does not rely on language-specific features such as ontologies or dictionaries and does not require morphological or syntactical preprocessing. They also use oversampling to address the class imbalance problem.

Reference [7] approached the multiclass classification from a different angle. HDLTex was proposed in this study as a hierarchical classification approach by applying different stacks of deep learning architectures at each level of the document hierarchy. Cotraining, which is a semisupervised approach, was presented in [9] for efficient document classification. Three document representation methods were utilized, namely, the term frequency-inverse document frequency (TF-IDF), latent Dirichlet allocation (LDA) and document to vector (Doc2Vec), on 5 multiclass datasets. In this study, the authors attempt to solve the problem of insufficient label information by using a semisupervised approach with the incorporation of various document representation methods for resolving the unstructured sparse format of the documents.

In our case, an unsupervised NN (SOM) and two DNN architectures, namely, an LSTM and a CNN+LSTM, are utilized. TF-IDF is used for vector construction by extracting information from the ToC of each book using SOM, while LDA and Doc2Vec can also be applied as alternatives for the ToC representation. Finally, the Keras tokenizer is used as a ToC representation method in both DNN architectures.

B. PRE-TRAINED LANGUAGE MODELS

Recently, many research efforts have relied on pretrained models. Pretrained word embeddings have been deemed more effective for several natural language processing (NLP) tasks than embeddings that are learned from scratch [17]. The use of pretrained models also has the advantage of transfer learning from supervised data, which can be a solution to the class imbalance problem, as discussed above.

The approach that is proposed in [1] aims at modeling the problem of identifying the relationship between two documents as a pairwise multiclass document classification task by utilizing Wikipedia and Wikidata to extract semantic relations. The identification of semantic relations between documents leverages 4 methods: GloVe [16], Paragraph-Vectors [15], BERT [17], XLNet [18] and GPT-2 [19]. Furthermore, in this study, a Vanilla transformer and a Siamese transformer have been used. In the latter case, BERT and XLNet have been combined in a Siamese architecture.

In contrast to GloVe and Paragraph-Vectors, which are context-free word embeddings, BERT and XLNet are contextual embeddings. Thus, the combination of contextual embeddings with the Transformer architecture has led to the unsupervised pretraining of deep language models, which has been proved to produce remarkable results on various NLP tasks. However, until now, Transformers have not been widely used in other domains such as content recommendation systems. [21] has used BERT for scientific paper recommendation.

According to [20], the Transformer is the first transduction model that relies entirely on self-attention to compute representations of its input and output by replacing the recurrent layers that are most commonly used in encoder-decoder architectures with multiheaded self-attention. The authors argue that by utilizing self-attention, the computational complexity for each layer is smaller compared to those of recurrent or convolutional networks. Furthermore, a larger degree of parallelization can be realized in terms of computation as in this case, significantly fewer sequential operations are needed.

BERT\(^2\) aims at pretraining deep bidirectional representations from unlabeled text by jointly conditioning on both the left and right contexts in all layers [17]. BERT’s architecture can be perceived as a multilayer bidirectional Transformer encoder. The major advantage of this technique is that it can be easily adapted to various NLP tasks just by adding one additional output layer to the basic pretrained model. It has been proved that pretrained representations reduce the need for heavily engineered task-specific architectures.

XLNet is a generalized autoregressive pretraining method that enables the learning of bidirectional contexts, as the

\(^1\) http://qwone.com/~jason/20Newsgroups/

\(^2\) BERT: Bidirectional encoder representations from transformers
context for each position can consist of tokens from both the left and right [18]. The authors suggest that empirically, under comparable experiment settings, XLNet outperforms BERT on 20 tasks due to its autoregressive formulation. They also argue that XLNet inherently (based on the autoregressive objective) eliminates the assumption that is made by BERT that the predicted tokens are independent of each other.

GPT-2 is a large language model with 1.5 billion parameters that is trained on a dataset of 8 million web pages and leverages the Transformer architecture [19]. Its main objective is to predict the next word in the scenario that all previous words are known. Similar to BERT and XLNet, GPT-2 is a contextual embedding, and it is scaled up 10x (in terms of parameters and the amount of training data) with respect to its predecessor GPT. It has been evaluated on various tasks, such as question answering, reading comprehension, summarization, and translation, and remarkable GPT-2 performance is realized without using task-specific training data. The main conclusion of this study is that these tasks can benefit from unsupervised techniques if the data and computing power are sufficient.

However, Transformers, BERT, XLNet and GPT-2 are deemed effective mainly on tasks such as question answering, natural language inference, sentiment analysis, document ranking and full-text machine translation. In our case, the use of ToCs and not full text (as this was not available) poses challenges in the incorporation of such techniques but would provide interesting insights into their applicability in other domains, such as content recommendation systems and multiclass classification using information on the ebook ToCs; this is left for future investigation.

C. FEATURE SELECTION

An important aspect when building an automatic text classification system is efficient text representation [26]. Most approaches encode documents as vectors, which leads to effective methods for vector quantization and feature selection with the objective of reducing the feature space representation [13], [30]. The feature space size also plays an important role in text classification, as an increase may cause a deterioration in the classifier’s performance [27]. Distributed embeddings such as word2vec [14], doc2vec [15] and Glove [16] have been proven effective in capturing the semantic meaning of texts while representing them in a lower-dimensional feature space. Methods for efficient document representation are deemed of significant importance as many irrelevant or redundant features may significantly affect the classifier’s performance [22], [23].

Various approaches have been proposed for document representation, such as the simple bag-of-words model [24], term frequency (TF) and inverse document frequency (IDF), LDA [9], information gain [11], [29], mutual information and pointwise mutual information [30], x2-test [28] and combinations thereof, while TF in combination with stemming has been proposed in [25]. Modified frequency-based term weighting schemes have been proposed in [28] that consider the number of missing terms in calculating the weights of available terms.

Simple averaging of word-vectors with an unweighted scheme was presented in [31]. Furthermore, a simple averaging of the TF-IDF weighting scheme of word vectors to produce document vectors has been investigated in [32]. The sparse composite document vector (SCDV), which was proposed in [33], extended the weighted averaging of word vectors from sentences to documents by using soft clustering over word vectors, while in [34], the approach was extended to capture also the multisense nature of words and to solve the problem of high dimensionality. This was realized by utilizing multisense word embeddings and by learning in a lower-dimensional manifold.

DNNs such as convolutional neural networks (CNNs) [34], [35], recursive neural networks (RNNs) [6] and long short-term memory networks (LSTM) [36] have also been utilized to capture syntax when representing documents.

The use of novel unsupervised features based on latent dirichlet allocation (LDA), stemming and semantic spaces has been proposed in [37]. LDA has been utilized to extract high-level representations from documents but requires finetuning on hyperparameters such as the number of classes or the word and topic distributions [38]. Averaging of word vectors based on hard clustering has been proposed in [39], while fuzzy clustering has been combined with TF-IDF weighting in [33] to form document vectors.

In this paper, we focus on selecting the most informative features of each book by experimenting with the Keras tokenizer for both DNN architectures, namely, the LSTM and the CNN+LSTM architectures, and with the TF-IDF scheme in combination with the snowball stemmer [40] for SOM to form the document vectors. The methodology that is applied in our case is generic and easily applicable to various datasets, thereby avoiding extensive feature engineering, which may yield better results on a given dataset but will not generalize when a different dataset is selected.

D. SELF-ORGANIZING MAPS AND DNNs FOR MULTI-CLASS CLASSIFICATION

SOM is an unsupervised NN that can cluster similar instances of the data together irrespective of the labels that are assigned to them [41]. SOM can map and display higher-dimensional vectors to two-dimensional space inherently [41]–[43]. SOM has been extensively applied in various application domains, including document classification. The problem of managing large document collections using SOM, followed by a set of suggestions on realizing dimensionality reduction for large SOMs, has been investigated in [42]. In this study, the authors demonstrate that the use of very large SOMs increases the complexity and processing time and, thus, does not constitute a satisfactory modeling approach. To address the multiclass classification efficiently, [44] investigated a cascaded SOM-based architecture in which classes were handled hierarchically. The authors suggest that the proposed approach
can be applied to any multiclass classification problem with a small number of labeled samples.

In contrast to SOM, which is an unsupervised classifier, deep learning is a family of efficient NNs [45] that can perform both supervised, unsupervised and semisupervised learning [46]. Hierarchical deep learning for text classification has been proposed in [7] as an antidote to the degradation of the performance of supervised classifiers when increasing the number of documents in multiclass settings. This study utilizes deep learning architectures not only to classify a document to a specialized domain but also to organize it into the proper subdomain, thereby creating a hierarchical classification approach. Reference [47] aimed at solving the problem of sequential short text classification by utilizing both RNNs and CNNs, and the evaluation was conducted on three multiclass datasets that were related to dialog act prediction. Similarly, [48] used CNNs for multiclass text classification using character-level features, while RNNs where used in [7] to capture time-dependent structures in the text.

Overall, our proposed solution aims at exploiting the advantages of the state-of-the-art approaches for classifying a multiclass corpus regarding feature selection and training processes, along with the label attribution to the testing samples, with the objective of enhancing the classifier’s performance in terms of accuracy while minimizing the processing time and computational resources that are needed. Our approach incorporates a specified procedure for vector construction to handle the multiclass requirements, thereby avoiding the reduction of the problem to a set of binary classification problems. Furthermore, the feature selection process is designed to be relatively generic, as it is based on word statistics; thus, it is easily applicable to various settings. We use the batch SOM algorithm for training, which has been deemed more stable and significantly faster than the online version, and two DNN architectures, namely, an LSTM and a CNN+LSTM, which seem to be efficient in NLP tasks.

III. DATASET DESCRIPTION

The dataset that is used in this paper has been created from a collection of 56403 multidisciplinary book titles from Springer, to which the Hellenic Academic Libraries Link is subscribed.3 This collection serves as a reference for tutors when suggesting sources for further reading, in addition to the learning material they provide to their students. To obtain this dataset, a parser was created for extracting relevant information, such as the title, subtitle and ToC, from each book. The extracted information was stored in a database for further processing. Each book title in the database includes information regarding the title, subtitle, authors, publication year, publisher, ToC, abstract and subject. As a next step, a team of librarians who were working in the NTUA Digital Library manually added the subject field information. The output of this step was a list of subjects for each book title. In this study, we use the primary subject field as each book’s label.

To obtain the primary subject label, we have implemented a mapping procedure, which will be analyzed below (Fig. 1). An example of the input data after the primary subject has been selected through the mapping procedure is presented in Fig. 2.

Not all fields were available for every book, which rendered the selection of information as input for our classification system less straightforward. We chose to model the problem of identifying similar books as a multitask classification problem by utilizing information from the ToC for effective book recommendation. The selection of the ToC is far from an incidental choice as a significant number of abstracts were missing from the collection. Furthermore, we believe that by utilizing information from the ToC, we can better capture the topics in each book, thereby facilitating the identification of similar books.

As this collection is a real-world dataset and there are no training and testing samples, we had to randomly split the dataset into testing and training sets during preprocessing, as will be analyzed below, using 80% of the samples for training and 20% for testing. Initially, there were 252 categories in the dataset, but as there were categories that could be grouped together, we mapped them to more general categories and finally obtained 26 general categories. The mapping has been conducted with the facilitation of the librarians of the NTUA4 digital library. An excerpt of this mapping procedure is presented in Fig. 1. We conducted experiments using the 26 resulting categories and the five general categories that contained most of the samples, namely, Medicine, Mathematics, Engineering, Computer Science and Physics. In Table 1, the distribution among the 26 different categories is reported. The total number of documents in the 26 categories is 56405, whereas the total number of documents for the 5 categories is 35271, which accounts for 63% of the total number of documents in the dataset. The distribution of samples for the 5 categories is reported in Table 2.

Furthermore, according to Table 1, most documents belong to Computer Science and Physical Sciences, and the distribution among the categories is not uniform. This poses challenges regarding the quality measures that will be used to measure the performance of the classifier, as the accuracy on the smaller categories may be falsely perceived being the same as that on the larger categories. Moreover, there is a significant subject overlap between some of the categories, e.g., between Engineering and Computer Science. Thus, micro- and macro-average measures will be used to measure the accuracy of the classifier, as will be analyzed in a subsequent section.

IV. METHODOLOGY

In this section, we propose, design and implement a novel approach for classifying a real-world multiclass dataset by leveraging information that is extracted from the ToCs. As discussed in the previous section, we had to randomly

---

3 https://www.heal-link.gr/en/home-2/

4 National Technical University of Athens
divide the samples into training and testing samples, and the labels that are used for the evaluation of the models come from the ‘subject’ field. Fig. 3 illustrates the proposed methodology at a high level. The solution is explained in detail in the following subsections. The proposed approach aims at extracting the most informative features from the ebook ToCs by utilizing two methods for their representation, namely, the TF-IDF weighting scheme in the SOM case and the Keras tokenizer in the LSTM and CNN + LSTM cases.

Both training and testing ToC vectors are constructed using the same procedure, which is explained below in detail. As a next step, the produced ToC vectors are fed to two models, namely, a SOM and a DNN (LSTM and CNN + LSTM), which can predict the most suitable label for each testing sample once training has been completed. Finally, the predicted output of the two models is evaluated against the actual labels from the testing samples using micro- and macro-average accuracy measures, as will be discussed in the Experimental Evaluation section.

### A. PREPROCESSING

During the preprocessing phase, the training samples are transformed into numeric vectors to be provided as input to the classifiers. This is a common step among most classification approaches that are reported in the literature, which usually encompasses some of the following processes:

- **Cleaning and Normalization**: This involves tasks such as removing special characters, converting text to lowercase, and handling missing values.
- **Tokenization**: This involves breaking the text into smaller units, typically words or subwords.
- **Stop Word Removal**: This involves removing common words that do not carry much information, such as “a”, “an”, “the”, etc.
- **Stemming and Lemmatization**: This involves reducing words to their base form or root.
- **Vectorization**: This involves converting the tokenized text into a numerical format.

The process of preprocessing is crucial as it can significantly impact the performance of the classification models. It is also important to note that the preprocessing steps can vary depending on the specific requirements of the project and the nature of the data.
E. Giannopoulou, N. Mitrou: AI-Based Methodology for the Automatic Classification of a Multiclass Ebook Collection

**FIGURE 3.** High-level schematic representation of the proposed methodology.

**FIGURE 4.** Preprocessing steps for every feature extraction method.

1. **word tokenization**, 2. **stop-word removal**, 3. **lemmatization** and 4. **stemming**. In our approach, this procedure depends on the ToC representation method, as summarized in Fig. 4.

We followed a different set of preprocessing steps when using the TF-IDF weighting scheme than when using the Keras tokenizer. In the first case, we removed any html-characters and punctuation, which was followed by numerical character removal. We also had to carefully process Latin numbers, which were usually found in the chapter titles. Then, the text was tokenized to retrieve a list of words for each ToC. For each list of words that was created in the previous step, we detached the stop words (such as “the”, “a”, “an”, “in”, “be”, “can”, etc.) using the corresponding function from the NLTK\(^6\) Python package. To convert each word to its base form by stripping words from their derivational affixes, we experimented with three stemmers: namely, Porter,\(^7\) Snowball\(^8\) and Lancaster.\(^9\) The next step concerns the removal of “too short” words, namely, words of less than five characters in this case; the threshold number of words is a parameter that could be altered in the initial configuration. This step should be performed after the tokens have been stemmed. Finally, the preprocessing phase concludes with the stripping of common words (words that are usually present in tables of contents), such as preface, introduction, chapter, references, appendix etc. In contrast, a minimal preprocessing procedure has been followed when using the Keras tokenizer for the ToC representation. In this case, only the first two steps were followed, as presented in Fig. 4. This procedure was selected because the objective was to use sequences of words, in contrast to the bag of words (BoW) model, and for sequences, even small words (e.g., conjunctions and pronouns) may impact the classification of an ebook. The output of this procedure is a vocabulary of keywords for each book ToC, an example of which is presented in Fig. 5.

In the next subsection, the methods that were utilized for the ToC vector representation and the steps that were followed for the construction of the vector representation of each ebook in the corpus are presented.

**B. TOC VECTOR REPRESENTATION**

This step aims at selecting the most informative features for representing a document as a vector to be fed to SOM and the DNNs that are selected in this paper. The vector

---

\(^6\)http://www.nltk.org/

\(^7\)https://www.nltk.org/_modules/nltk/stem/porter.html

\(^8\)https://www.nltk.org/_modules/nltk/stem/snowball.html

\(^9\)https://www.nltk.org/_modules/nltk/stem/lancaster.html
dimensionality depends on the number of selected features that comprise the feature space. The higher the dimension of the vectors, the more computationally intensive the classification task will be.

Let us suppose \( S = \{ ToC_1, ToC_2, \ldots, ToC_K \} \) is the collection of the tables of contents of the Springer ebook corpus. The ToC of each book in the collection \( S \) is represented by an \( m \)-dimensional vector of the form \( ToC_i = \{ w_{1i}, w_{2i}, \ldots, w_{mi} \} \), where \( w_{mi} \) is the weight of table of contents \( ToC_i \) for feature \( m \). The weights are computed using the TF-IDF weighting scheme, which is a complex weighting scheme in which the product of the TF and IDF measures is utilized. TF of term \( t \) in document \( d \) is the raw number of times that \( t \) appears in \( d \), whereas the IDF measure is the rarity of a word across all documents in the collection. The full mathematical formulations of the IDF and TF-IDF measures are presented in [8] and are omitted here for brevity. The vectors were constructed using the TfidfVectorizer, which is available through the scikit-learn Python package.  

First, the TF-IDF values for all words in the corpus are calculated, and the \( x \) words that have the highest TF-IDF scores are selected to construct the vectors. As will be analyzed in detail in the Experimental Setup section, \( x \) ranges from 2000-3000 words. When using the TfidfVectorizer, there is the option to use only single words (1-grams) or \( n \)-grams of words as features. In this paper, we use only 1-grams due to the option to use only single words (1-grams) or \( n \)-grams of words. In this paper, we use only 1-grams due to the option to use only single words (1-grams) or \( n \)-grams of words. In this paper, we use only 1-grams due to the option to use only single words (1-grams) or \( n \)-grams of words.

TABLE 3. Constructed vector with assigned labels.

| Sample | \( f_1, f_2, \ldots, f_n \) | \( l_1 \) |
|--------|--------------------------|---------|
| Sample2| \( f_1, f_2, \ldots, f_n \) | \( l_2 \) |
| SampleN| \( f_1, f_2, \ldots, f_n \) | \( l_m \) |

Previously, multiclass problems are usually transformed into a set of binary classification problems. Instead of constructing multiple binary classifiers, we propose the use of two NNs, namely, an unsupervised NN and two DNN architectures, as was explained above and will be discussed in detail in the next subsections, that take as input the constructed vectors.

The second vectorization method that is used in this study is the Keras tokenizer, namely, the Tokenizer API. We constructed the tokenizer and fit it to the preprocessed ToCs. To construct the tokenizer, we used the following configuration: a. set the maximum number of words to keep (20000 words in this case) based on the word frequency, b. convert all words to lowercase, and c. use whitespace as a word separator. When the tokenizer is fit to the list of ToCs, a vocabulary index that is based on the word frequency is created. In this case, a smaller integer corresponds to a more frequent word, while the first words are usually stop-words as they tend to appear frequently. Then, each ToC is transformed into a sequence of integers using the word index dictionary that was constructed in the previous step. Finally, we construct fixed-length vectors by padding the sequence of integers with zeros using the max_length parameter (which ranges from 800-3000 in our case). Exactly the same word index dictionary is utilized for the testing samples to convert the text into sequences to be fed to the NNs. The final vector for this case has the same structure as that used in the TF-IDF case (presented in Table 3).

C. SOM ALGORITHM

SOM, which was proposed by Kohonen in [41], is a two-dimensional array of neurons in which each neuron is a codebook vector. As the vectors pass from the input layer, they are projected to a spatial location of an output neuron on the map. One of the main advantages of SOM, which is implied by its name, is that the network can learn from the vectors that pass through its input layer in a self-organized manner. The weights that are assigned to each output neuron change in every training epoch depending on the input vector that is provided. The objective is to minimize the distance of

FIGURE 5. Example of the preprocessing procedure for the TF-IDF representation method. The output of this procedure is a vocabulary of keywords for each book ToC.
each input vector from its allocated best matching unit (BMU hereinafter). The input data constitute a set of \( n \)-dimensional vectors \( x \). The codebook vectors that are assigned to each BMU are represented by \( m_i \), where \( i \) is the index of the codebook vectors. This index depends on the selected grid size. For example, if the selected grid is \( 10 \times 10 \), then the index of the codebook vectors ranges from 0-99. The winning codebook vector index, which is denoted by \( c \), can be identified via the following equation:

\[
c(x) = \arg \min_i \{\|x - m_i\|\}
\]  

(1)

To identify the optimal partitioning settings, two measures are available: the mean quantization error (QE) and the topographic error (TE). The QE is computed by determining the average distance of the sample vectors to the codebook vectors by which they are represented, whereas the TE measures the topology preservation for a given input by using information from the BMUs. The TE is computed for all input vectors and is normalized to the 0-1 range, with 1 corresponding to perfect topological preservation. It has been reported that there is a trade-off between QE and TE as increasing projection quality usually degrades the projection properties [41]. Furthermore, larger grid sizes tend to decrease QE monotonically. In this study, we use the formulas that are presented in [8] to compute these measures. Essentially, we use the QE and TE measures as guides for selecting the best model. This is presented in Table 4 in the Results subsection.

For the implementation of SOM in this paper, we used the SOMPY python library, \(^{12}\) which is based on the Matlab SOMToolbox library, along with the following configuration: We selected the batch version of SOM, together with linear initialization and variance normalization for the input vectors, as it converges faster and there is no need for the learning rate parameter. Linear initialization is preferable to random initialization as the codebook vectors lie in the same input space that is spanned by the two eigenvectors that correspond to the largest eigenvalues of the input data and, thus, stretching SOM to the same orientation as the data playing the most significant role.

The SOM training and testing procedure is illustrated in Fig. 6. First, each node on the map is associated with a codebook vector. Then, for every training vector, the BMU for which the weight is closest to the vector in terms of the Euclidean distance is identified. Finally, the values of the codebook vectors in the neighborhood of the winning unit are updated to come closer to the BMU (marked with red – Fig. 6). This procedure is repeated for all training epochs. Via this process, SOM shapes an elastic network that folds onto the schema that is formed by the input data during the training procedure. After the training phase is completed, each BMU is associated implicitly with a set of labels that correspond to the training samples that are assigned to each node on the map. Each BMU on SOM can have one or more labels assigned to it.

The next step is to project the testing vectors on the map and calculate the BMU for each of them. After the projection, each vector from the testing subset is associated similarly with a set of labels based on the BMU in which

\(^{12}\)https://github.com/sevamoo/SOMPY

![FIGURE 6. Flowchart of the proposed solution using SOM. The left part depicts the training procedure and the right the testing procedure. The output of the training is an SOM, which is provided as input, together with the testing documents, to identify the suitable label for each testing sample.](image)
it was allocated. Various approaches have been proposed for selecting the labels to associate with each testing sample. In multilabel settings, one can either assign all labels in the BMU or conduct majority voting to select the most frequent labels. However, the first option cannot be applied in our case as it is a multiclass setting but not a multilabel setting.

In [8], we have proposed an intelligent algorithm that is especially tailored to multilabel settings and leverages information from the neighboring nodes to select the labels for each node. In this study, we attempted to apply INterSECT in a multiclass setting and utilized majority voting as an alternative. An example of the process of selecting labels based on majority voting and INterSECT algorithms is presented in Fig. 7. The performances of these methods for label selection were evaluated experimentally, as described in detail in the following section.

D. DEEP NEURAL NETWORKS

Deep neural networks (DNNs) can be used to build efficient computational models in supervised, unsupervised and semisupervised settings. In its basic form, a DNN is a fully connected set of processing units (nodes) that are organized into layers. The first layer is the input layer, the last layer is the output layer and all intermediate layers are hidden. Feedforward neural networks have been proven to be universal function approximators and, thus, can find a nonlinear mapping from high-dimensional input to lower-dimensional output. In this paper, we experimented with various deep learning architectures and hyperparameter combinations to successfully classify a multiclass corpus by leveraging information from the ToCs. We have utilized an embedding layer to decrease the sparsity of the input while obtaining word embeddings that provide a dense representation of the words and their meaning. These word embeddings are learnable and can be used on other similar datasets effectively. In this study, we utilize a single-layer bidirectional LSTM and a CNN+LSTM multilayer architecture with different network configurations, as will be analyzed in the subsections below.

The LSTM architecture is illustrated in Fig. 8, and the CNN+LSTM architecture is illustrated in Fig. 9. The LSTM input is a 64-dimensional embedding of each word. This setting aims at capturing the temporal dependencies of the words in each ToC. As the output, we obtain a dense representation of the ToC, with 64 features that are used to perform the classification. The utilized bidirectional LSTM consists of 128 features, which are passed as input to the next layer. Two hidden layers with ReLU activation, which output 64 and 32 features, have been used, which are followed by an output layer with a softmax activation function. The output layer has a node for each classification label, namely, 26 or 5 categories in this case. According to [1], an LSTM can be biased when later words are more influential than the earlier words. To address this bias, we have included a max pooling layer in the convolutional neural network (CNN) that will be
discussed below to identify discriminative phrases in the ToC of each book.

The main characteristic that distinguishes a CNN is that it incorporates convolutional layers. Each convolutional layer connects to a small subset of the inputs; typically, a kernel of size 3 is utilized. When convolutional layers are stacked, the subset of inputs from the previous layers is connected only with the subset of inputs of the next layer. These convolutional layers are called feature maps, and when stacked, they can provide multiple filters on the input. As discussed above, one of the main benefits in using CNNs is that they make feature selection more robust, thereby avoiding the potential bias when the later words have more influence than the preceding words. Furthermore, by using downsampling, it is possible to reduce the number of parameters in the model and, thus, reduce the computational complexity.

Different pooling techniques have been proposed so as to reduce the size of the output from one stack of layers to the next in the network [52], the main aim of which is to capture the input’s most informative features. The most common pooling method and the method that is used in this paper is max-pooling, in which the max() function is applied over the contents of the pooling (sliding) window. Another approach is to select the statistical mean of the contents. Finally, to feed the pooled output to the next layer, feature maps are flattened into one column and are fed to a fully connected set of layers.

In this paper, we use a slightly different architecture that consists of a CNN with 3 layers, in combination with a bidirectional LSTM. The CNN+LSTM architecture is illustrated in Fig. 9. Each layer consists of a 1D convolutional layer with a scaled exponential linear unit (SELU) [53] activation function, followed by a max-pooling layer. A dropout layer is used to slow the learning process, as CNNs tend to learn quickly, with the objective of producing a better model at the end. For this model, we use a standard configuration of 32, 64 and 128 parallel feature maps and a kernel of size 3. The size of the feature maps corresponds to the number of times the input is interpreted, whereas the kernel size corresponds to the number of input time steps that are considered as the input sequence is read or processed into the feature maps.

Finally, the output from the 3 layers is fed into a bidirectional LSTM with one hidden layer using the ReLU activation function, which is followed by the output layer, in which softmax is used. The final output is, as in the previous case, either the 26 or the 5 categories.

V. EXPERIMENTAL EVALUATION

In this section, an experimental evaluation of the proposed approach will be presented. The main objective of this procedure is to evaluate the classification accuracy for the selected dataset. First, the metrics that are used to evaluate the classifiers’ performance will be presented, followed by the optimization functions that are selected for the deep learning models, the experimental set up, and the results that are obtained from the conducted experiments.

A. PERFORMANCE METRICS

The most commonly used performance metrics to assess a classifier’s performance for text classification are precision, recall and F1-score [9]–[11], [28]. Micro- and macro-average precision measures are commonly used to evaluate the accuracy of classifiers in multiclass problems [11], [28]. However, when comparing micro- with macroaverage results the outcome may be quite different. This stems from the difference when calculating the two measures. In macroaverage all classes are given identical weights, whereas in microaverage the equal weights are given to each per-document classification decisions [28]. In this case, we calculate both the micro- and macroaverage precisions to obtain an overall view of how the classifiers perform on the less frequent categories and on the largest categories. Then, we use the micro- and macroaverage precisions to calculate the micro- and macroaverage F1-scores to assess the accuracy of the classifiers.

The micro- and macroaverage precision measures that are available through the scikit-learn python module are used in this paper, which are formally defined in the equations below.

**Precision** is calculated by dividing the number of true positives (TP) by the sum of TP and the number of false negatives (FN):

\[
\text{Precision} = \frac{TP}{TP + FP} \quad (2)
\]

**Recall** is calculated by dividing the number of correctly classified data items TP with the sum TP + FN, where FN is the number of items that are incorrectly classified as not belonging in the class.

\[
\text{Recall} = \frac{TP}{TP + FN} \quad (3)
\]

The **F1-score** is based on the precision and recall measures that are defined above. In terms of precision, either the micro-, macro- or weighted average can be used to measure F1. 

1http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
2http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html
The higher the score, the better the classification performance.

$$F_1\text{-score} = \frac{Precision \times Recall}{Precision + Recall}$$  \hspace{1cm} (4)

**Microaverage Precision:** Every ToC has the same importance. The most common categories more strongly affect the aggregate quality than the smaller categories [28]. In this case, the precision is equal to the recall, and they are also equal to their harmonic mean. The microaverage precision also represents the classifier’s overall accuracy and is calculated from the number of correctly classified data items (TP) and the total number of data items \(n\), as expressed in Equation (5).

$$Micro\ Average\ Precision = \frac{TP}{n}$$  \hspace{1cm} (5)

**Macroaverage Precision:** In this case, all categories are equally important. The quality for each category is calculated independently as their average. The macro-average is a satisfactory quality measure for smaller categories [11], [28]. In equation (6), \(TP_j\) denotes the correctly classified ToCs that belong to class \(j\), and \(n_j\) denotes the total number of documents that belong to that class.

$$Macro\ Average\ Precision = \frac{\sum_{j} TP_j}{|\text{Classes}|}$$  \hspace{1cm} (6)

### B. OPTIMIZATION

Two types of stochastic gradient optimization are used in this paper for the DNNs, namely, root mean square propagation (RMSProp)\(^{15}\) and adaptive moment optimization (Adam) [54], as described below.

**RMSProp Optimizer:** The main objective of RMSProp is to maintain a moving (discounted) average of the square of the gradient and to divide the gradient by the root of this average. The implementation of RMSProp that is used in this paper considers the plain momentum, not the Nesterov momentum.

**Adam Optimizer:** Adam is a stochastic gradient descent method that is based on adaptive estimation of the first-order and second-order moments. According to [54], Adam is computationally efficient and has low requirements in terms of memory, and it is invariant to diagonal rescaling of gradients. Furthermore, it is considered to be well suited for problems that have a huge number of parameters. In this paper, we use the RMSProp and Adam optimizers from the Keras API\(^{16}\), which is available in TensorFlow.

### C. EXPERIMENTAL SETUP

The experimental evaluation is conducted on the Springer corpus that is described in Section III. The feature vectors were constructed by following the approach that was described previously, whereas the corpus was trained using both the batch SOM algorithm and two DNN architectures, namely, an LSTM and a CNN+LSTM. Rounds of experiments were conducted using various map sizes, configuration options (SOM), lengths, and vocabulary sizes (DNNs) on both the 26 and 5 categories of the corpus.

The first round of experiments was aimed at assessing classification performance when using TF-IDF as the weighting scheme with SOM. As such, the same configuration options were used for both the entire Springer dataset (26 categories) and the 5-category subset. The parameters regarding the rough and finetuning training epochs were set to 15 and 7, respectively. Three vector sizes were utilized, namely, 2000, 2500, and 3000, to assess the impact of the vocabulary size on the classifier’s performance. In this case, the considered map sizes are \(10 \times 10, 16 \times 16, 20 \times 20, 30 \times 30, 40 \times 40, 50 \times 50, 60 \times 60\) and \(70 \times 70\). Experiments where conducted with both the INterSECT algorithm and majority voting for label selection.

The second round of experiments was aimed at evaluating the use of DNNs, namely, an LSTM and a CNN+LSTM, instead of an unsupervised NN (SOM) in the classification of a multiclass corpus. In this round, the Keras tokenizer has been used, and various max length sizes for the ToCs were considered, namely, 800, 1000, 2000, 2500 and 3000 words, along with a vocabulary of 20000 words in total. Experiments were also conducted to assess the effects of the max length parameter and the lack of preprocessing on the classifier’s accuracy. Furthermore, the hyperparameters and the optimizers that are presented in subsection B were used as analyzed below.

**LSTM:** Embedding dimension: 64, max features: 20000, input length: 800, dropout: 0.2-0.25, loss: categorical cross-entropy, training batch size: 128, and epochs: 10.

**CNN:** Embedding dimension: 64, max features: 20000, input length: 800, dropout: 0.25-0.5, loss: categorical cross-entropy, training batch size: 128, epochs: 15, and learning_rate=3e-4.

### D. RESULTS

1) 1st ROUND: SOM + TF-IDF

As described in the previous subsection, we conducted experiments using both the 5 and all 26 categories of the Springer ebook corpus. To evaluate the SOM performance, we experimented with various map and vector sizes while utilizing square maps and 1-grams.

With respect to the 26 categories, we observed that the results tend to follow the same pattern, as for each map size small differences were observed between vector sizes. For the same map size, the differences between vector sizes range from 0.05-0.2%. The classifier realized a microaverage F1-score of approximately 0.62, or 62%, with a \(16 \times 16\) map. We also observed that as the map size increases, the F1-scores for the micro-average tend to decrease to a minimum of 58% for a \(70 \times 70\) grid. This was expected because with larger map sizes, the samples and the corresponding labels tend to spread over the grid. For example, in a \(20 \times 20\) grid, 5 labels may be concentrated at a single map node, whereas in a \(40 \times 40\) grid, only 1 label may be present at the same map node. We should also consider the case in which no label is present at the node.

---

\(^{15}\) https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/RMSprop  
\(^{16}\) https://keras.io/
which occurs for the larger grids, as presented in Table 5. These two factors explain the decrease in the F1-scores for the larger grids, which are 2-4% lower than those in the smaller grids.

For macro-average F1 scores, the classifier can better predict the document class when using 3000 dimensions per vector as it yields better results for most of the map sizes that we considered, as presented in Fig. 9. The classifier scores approximately 27% for a 40 × 40 map. The best performing map sizes are 16 × 16, 30 × 30 and 40 × 40. In this case, the deviation between the vector sizes ranges approximately 2%. The results for the macro-average are significantly lower than the micro-average results, but this is reasonable as it is more difficult for the classifier to predict well on categories that have few samples, such as “Music”, “Literature” and “Food”, as detailed in Table 1.

Fig. 10 compares the two different label selection methods that are utilized in this paper, namely, the majority voting and INterSECT algorithms, for the 26 categories. As discussed previously, INterSECT is an algorithm that is tailored to multilabel settings; hence, we expected that its performance would be lower in a multiclass scenario. This is also reflected in the obtained results, which are presented in Fig. 10 for the 26 categories and Fig. 11 for the 5 categories. According to Fig. 10, majority voting outperforms INterSECT in terms of the micro-average F1-score for all vector sizes.

The proportion difference between majority voting and INterSECT is almost the same (50% percent lower for INterSECT) for both the micro- and macroaverages. The corresponding figures for the macro-average F1-scores are omitted here for brevity.

The best results that were obtained for the 5 general categories for each map size are presented in Table 4. According to the obtained results for the 5 categories, a trend in the classifier’s performance is identified for both the micro- and macroaverages. Similar to the 26 categories, the F1-scores also tend to decrease as the map size increases. Another observation is that, especially for the micro-average scores, the difference between vector sizes is negligible and, hence, the results for various vector sizes are omitted. According to the table, the 16 × 16 map is the best performing with F1-scores of 75% for the micro-average and 73% for the macro-average.

Table 5 presents the precision, recall and F1-scores on a per-class basis. According to the results, the classifier can better predict the ‘Medicine’ category compared to the remaining categories. In this case, the F1-scores range from 87-89%. This is because ‘Medicine’ is the 2nd largest category in terms of samples and is conceptually disjoint compared, for
TABLE 6. U-matrices, hitmaps and cluster SOM views using the TF-IDF tokenizer for various map sizes (5 categories).

Example, to ‘Computer Science’ and ‘Mathematics’. Another observation is that the classifier is more capable of predicting the ‘Computer science’ category than the ‘Engineering’ category, with the results ranging from 77-79% and 58-61%, respectively. This is because the ‘Engineering’ category is closely related to the ‘Computer science’ category and, hence, testing samples can be falsely perceived as belonging to ‘Computer science’ instead of ‘Engineering’. Furthermore, the ‘Engineering’ category is half the size of ‘Computer science’ in terms of samples. In addition, the classifier scored approximately 67-71% for ‘Mathematics’ and 63-68% for ‘Physics’, which is relatively satisfactory if we consider the number of samples that are available for training and validation, as presented in Table 1.

Table 6 presents unified distance matrices (U-Matrices), hit maps and cluster SOM views for various grid sizes for the 5 categories. As discussed previously, each node on the map is represented by a codebook vector that has the same dimension as the vectorized documents. The U-Matrix is a useful tool for visualization of the distances between neighboring nodes on the map. Furthermore, it facilitates the depiction of the map’s cluster structure. By observing the map colors, one can discriminate between clusters and cluster borders. More specifically, nodes of similar colors correspond to clusters, whereas nodes with different or dark colors correspond to cluster borders. Similar colors also indicate similarity between vectors in the same area, while dark colors indicate large distances and, thus, less similar codebook values in the input space. The U-Matrices that are presented in Table 6 have numbers attached to each node. The numbers denote the categories that were assigned to the nodes after the training, namely, the labels that were selected using the majority voting algorithm (0 – Physics, 1 – Medicine, 2 – Engineering, 3 – Mathematics, and 4 – Computer science).

Maps of smaller size tend to have more labels attached to each node, whereas larger maps have fewer or none. This can be better illustrated using hitmaps, as shown below. In this view, every node has a number assigned to it, which is the number of samples that are allocated to the node after the training. Hitmaps can facilitate understanding of how the samples are spread on the grid. Finally, the third column of Table 6 presents the cluster view of SOM for each map size. In this case, 5 clusters have been created, 1 for each category. However, as SOM contains non-delimited clusters, there may be more clusters present that cannot be shown in this view. Hence, the U-Matrix view is a better guide for identifying the clusters.

2) 2nd ROUND: LSTM, CNN+LSTM – KERAS TOKENIZER

In the second round of experiments, we substituted SOM with 2 DNNs: an LSTM and a CNN+LSTM. The Keras tokenizer was utilized for the ToC representation to assess its performance in efficiently representing the ToCs of ebooks in the Springer corpus. The obtained results are summarized in the following figures.

Fig. 12 presents the micro-average F1-score evolution during training. Both the 26 and 5 categories tend to produce similar results in terms of the scores between the RMSProp and Adam optimizers. Furthermore, RMSProp yields better results than Adam in both settings (26 and 5 categories) after the 3\textsuperscript{rd} training epoch. In this case, it seems that the scores do not fluctuate as the training evolves. The same is
TABLE 7. Overall best micro- and macro-average F1-scores of TF-IDF SOM, KERAS LSTM, and KERAS CNN+LSTM.

| Method            | Micro-F1 | Macro-F1 | Micro-F1 | Macro-F1 |
|-------------------|----------|----------|----------|----------|
| TF-IDF/SOM        | 0.62     | 0.62     | 0.62     | 0.27     | 0.27     | 0.26     |
| Keras/LSTM RMSProp| 0.67     | 0.67     | 0.67     | 0.31     | 0.29     | 0.29     |
| Keras/LSTM Adam   | 0.64     | 0.64     | 0.64     | 0.28     | 0.27     | 0.27     |
| Keras/CNN+LSTM RMSProp | 0.65 | 0.65 | 0.65 | 0.26 | 0.28 | 0.26 |
| Keras/CNN+LSTM Adam | 0.65 | 0.65 | 0.65 | 0.27 | 0.27 | 0.27 |
| TF-IDF/SOM        | 0.75     | 0.75     | 0.75     | 0.74     | 0.73     | 0.73     |
| Keras/LSTM RMSProp| 0.80     | 0.80     | 0.80     | 0.78     | 0.77     | 0.78     |
| Keras/LSTM Adam   | 0.77     | 0.77     | 0.77     | 0.74     | 0.75     | 0.74     |
| Keras/CNN+LSTM RMSProp | 0.80 | 0.80 | 0.80 | 0.79 | 0.77 | 0.78 |
| Keras/CNN+LSTM Adam | 0.79 | 0.79 | 0.79 | 0.77 | 0.77 | 0.77 |

deduced from the macro-average scores (Fig. 13); however, in this case, the differences between the two optimizers are negligible.

Similarly, Fig. 14 and Fig. 15 present the F1-scores that were obtained with the CNN+LSTM architecture for the micro- and macroaverages, respectively.

In this case, the F1-scores are lower for the initial training epochs compared to those of the LSTM architecture and tend to stabilize after the 9th epoch for the 26 categories and after the 11th epoch for the 5 categories. The RMSProp optimizer produces better results in this case. Moreover, the differences in the scores that were obtained in each training epoch between the optimizers are more significant in this case, but in the end, the scores tend to converge.

The best scores that were realized by all methods are summarized in Table 7. For the 26 categories, the LSTM with the RMSProp optimizer produces the best scores, namely, 67% for the micro-average and 29% for the macro-average, followed by CNN+LSTM, which scores 2% lower, and SOM, which scores 3% lower.

For the 5 categories, the LSTM and the CNN+LSTM with RMSProp realize a micro-average of 80%, whereas
Adam realizes 3% and 1% lower, respectively. In this case, SOM realizes 5% lower (75%). In terms of the macro-average, the best method is the CNN+LSTM with RMSProp (79%), followed by the LSTM RMSProp (78%), whereas Adam achieves scores of 77% and 74%, respectively, and SOM 73%.

VI. CONCLUSION
In this paper, we investigated the use of an unsupervised NN (SOM) and two DNN architectures to assess the classification performances of the methods in a multiclass setting. The feature selection approach is as lightweight and generic as possible to ensure its applicability to other datasets with minor modifications. The vector construction leverages information that is extracted from the ToC of each ebook using the TF-IDF weighting scheme in the SOM case and the Keras tokeniser in the DNN (both LSTM and CNN+LSTM) case. We have shown that by using the proposed feature selection process, we can capture the most significant features of the dataset while maintaining a relatively small vector size, thereby increasing the classification efficiency.

Furthermore, the InterSECT algorithm for label selection that was proposed in [8] has also been utilized in this paper to assess whether it can be applied in a multiclass setting instead of a multilabel setting successfully. Majority voting was also applied as an alternative label selection method for SOM. We observed that majority voting outperforms InterSECT in all settings; hence, InterSECT is more suitable for multilabel scenarios only. The proposed approach was evaluated under various configuration settings on a multiclass corpus of approximately 56000 ToCs that were extracted from a set of multidisciplinary ebook titles from Springer. Via comparison to the results that were presented in [8], we deduce that the approach generalizes well as we have been able to reuse it on a different dataset with minor changes. By exploiting the similarity of ebooks that belong to the same cluster, we could identify similar ebooks for each category that could be proposed as alternatives when selecting ebooks for each course.

In this paper, we extended the approach by utilizing two DNN architectures to assess their classification performances and compare the results with those obtained using SOM.

Furthermore, two optimizers, namely, RMSProp and Adam, were utilized. The results suggest that both DNN architectures yield better results by 5% compared with the unsupervised NN. However, the DNNs require significantly more resources in terms of computing power and memory but are faster than SOM when run on a GPU. The results demonstrate that both the LSTM and the CNN+LSTM were efficient in classifying a multiclass corpus by utilizing information from the ToCs with minor hyperparameter tuning. In addition, considering that the CNN+LSTM architecture is more complex and, thus, computationally more expensive compared to that of the bidirectional LSTM and since both architectures have the same performance, we consider the latter more suitable.

By applying the methodology that is presented in this study and leveraging the ToC information, we can obtain a better view of the book’s contents while being able to determine which books are similar even though they may not share the same keywords in their ToCs or even the same subject label. In this problem, we do not expect 100% classification accuracy as the labels for each sample are assigned by librarians and, thus, the label allocation is subjective. Moreover, there are labels that are highly correlated, for example, ‘Computer science’ and ‘Engineering’, and even though the system may classify a book as belonging to ‘Engineering’, it can also belong to ‘Computer science’.

VII. FUTURE WORK
The methodology that is described here can be further extended in multiple ways. A potential extension of this study would be to extend the classification to identify hierarchies within categories by using more specific categories as labels and retraining the networks. In this approach, to obtain better accuracy in the multiclass problem, SOM can be used to identify the general categories, similarly to the approach presented in this study and then for each cluster in the map a new SOM classifier could be trained using the more specific labels. Finally, based on the cluster that each testing sample is assigned to, the corresponding SOM model will be used to decide on the more specific label.

Siamese neural networks or contrastive supervised learning could also be exploited to address the class imbalance problem. In addition, pretrained language models such as Transformers, BERT, XLNet and GPT-2 can be utilized. Via this approach, we can leverage already learned embeddings instead of conducting costly training operations from scratch. Networks of this type could be used as an alternative to siamese neural networks or contrastive supervised learning to address the class imbalance problem in the Springer dataset and other real-world datasets. This could be realized by applying transfer learning from supervised data. To the best of our knowledge, Transformers have not yet been applied in content recommendation systems. Hence, this would be an interesting extension to evaluate their applicability in multiclass classification by utilizing information on the ebook ToCs. Finally, the proposed methodology can be applied to other datasets that are available through the Hellenic Academic Libraries’ subscription to aid professors and students in searching for similar content.

REFERENCES
[1] M. Ostendorff, T. Ruas, M. Schubotz, G. Rehm, and B. Gipp, “Pairwise multi-class document classification for semantic relations between wikipedia articles,” 2020, arXiv:2003.09881 [Online]. Available: http://arxiv.org/abs/2003.09881
[2] B. A. Rabut, A. C. Fajardo, and R. P. Medina, “Multi-class document classification using improved word embeddings,” in Proc. 2nd Int. Conf. Comput. Big Data (ICCBD), 2019, pp. 42–46.
[3] H. T. Sueno, “Multi-class document classification using support vector machine (SVM) based on improved Naïve bayes vectorization technique,” Int. J. Adv. Trends Comput. Sci. Eng., vol. 9, no. 3, pp. 3937–3944, Jun. 2020.
ELENI GIANNOPOULOU graduated from the Department of Computer Science and Biomedical Informatics, University of Thessaly (UTH), in 2008. She received the M.Sc. degree in advanced informatics systems from the University of Piraeus in 2010. She is currently pursuing the Ph.D. degree with the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA). She has been involved in both European (FP7, H2020) and national research projects working as a Solution Architect since March 2009. Her research interests include machine learning and information extraction from textual data. She also focuses on the application of machine learning techniques on big data for creating interesting insights in various domains.

NIKOLAS MITROU (Member, IEEE) received the Diploma degree in electrical engineering from NTUA, in 1980, the M.Sc. degree in systems and control from UMIST, Manchester, in 1982, and the Ph.D. degree in electrical engineering from NTUA, in 1986.

From 1982 to 1985, he was with the Nuclear Research Centre Demokritos, Athens, where he was involved in signal processing projects. From 1986 to 1988, he worked with the National Defense Research Centre, Athens, for the development of a low-bit-rate voice coding system. He is currently a Full Professor with the School of Electrical and Computer Engineering, National Technical University of Athens (NTUA), Greece. His research interests include digital communication systems, networks and networked applications and services, sensor networks, knowledge and semantic Web technologies, with emphasis on the architecture, modeling, performance evaluation and optimization of systems and networks, as well as on end-user application and service development, having more than 45 journal articles in the above fields. He is also active in the areas of sensor networks and distributed knowledge systems with applications in the fields of Geographic Information and multimedia content management.