Neural Computing and Applications (2022) 34:937–950
https://doi.org/10.1007/s00521-021-06053-z

SPECIAL ISSUE ON ADVANCES OF NEURAL COMPUTING PHASING CHALLENGES IN THE ERA OF 4TH INDUSTRIAL REVOLUTION

Thesaurus-based word embeddings for automated biomedical literature classification

Dimitrios A. Koutsomitropoulos[1] · Andreas D. Andriopoulos[1]

Received: 6 December 2020 / Accepted: 15 April 2021 / Published online: 11 May 2021

Abstract
The special nature, volume and broadness of biomedical literature pose barriers for automated classification methods. On the other hand, manually indexing is time-consuming, costly and error prone. We argue that current word embedding algorithms can be efficiently used to support the task of biomedical text classification even in a multilabel setting, with many distinct labels. The ontology representation of Medical Subject Headings provides machine-readable labels and specifies the dimensionality of the problem space. Both deep- and shallow network approaches are implemented. Predictions are determined by the similarity between extracted features from contextualized representations of abstracts and headings. The addition of a separate classifier for transfer learning is also proposed and evaluated. Large datasets of biomedical citations are harvested for their metadata and used for training and testing. These automated approaches are still far from entirely substituting human experts, yet they can be useful as a mechanism for validation and recommendation. Dataset balancing, distributed processing and training parallelization in GPUs, all play an important part regarding the effectiveness and performance of proposed methods.

Keywords Classification · Indexing · Word embeddings · Thesauri · Ontologies · Doc2Vec · ELMo · BERT · MeSH · Deep learning

1 Introduction

In the light of recent public health emergencies, the sheer volume of biomedical literature is continuously increasing and puts excessive strain on manual cataloguing processes: For example, the US National Library of Medicine experiences daily a workload of approximately 7000 articles for processing [19].

Current research in the automatic indexing of literature is constantly advancing and various approaches are recently proposed, a fact that indicates this is still an open problem. These approaches include multilabel classification using machine learning techniques, training methods and models from large lexical corpora as well as semantic classification approaches using existing thematic vocabularies. To this end, the Medical Subject Headings (MeSH) is the de-facto standard for thematically annotating biomedical resources [28]. At the same time, natural language processing (NLP) meets a rejuvenation owing to the success and milestone achievements of related state-of-the-art machine learning models.

In this paper we propose and evaluate various approaches for automatically annotating biomedical articles with MeSH terms. We are interested in the performance of current state-of-the-art algorithms based on contextualized word representations and word embeddings, including Doc2Vec [14], ELMo [23] and BERT [3]. We suggest producing vectorized word and paragraph representations of articles based on context and existing thematic annotations (labels). Consequently, we seek to infer the most similar terms retrieved by the model. We evaluate and

✉ Dimitrios A. Koutsomitropoulos
koutsomi@ceid.upatras.gr

Andreas D. Andriopoulos
a.andriopoulos@upatras.gr

[1] Department of Computer Engineering and Informatics, School of Engineering, University of Patras, Patras, Greece

compare both pretrained and transfer learning models for multilabel classification (see Sect. 3.4). Moreover, we combine these algorithms with structured semantic representations in Web Ontology Language format (OWL), such as the implementation of the MeSH thesaurus in OWL Simple Knowledge Organization Systems (SKOS) [30]. We investigate the effect and feasibility of employing distributed data manipulation for dataset preprocessing as well as dataset balancing. Finally, we examine the scalability of training for transferring learning occurring within a pretrained language model to an attached classifier.

A preliminary version of this paper appears in [11]. In addition to the original paper we have built an additional multilabel classification model based on ELMo embeddings. We also investigate additional transfer learning language models including BERT.

We evaluate our models, including vector space word models (Doc2Vec) and perform a comparative analysis of results. Depending on the model evaluated and the total number of classes, our experiments yield results with *F*-score values ranging from 0.34 to 0.77. These outcomes improve or are on par with current state of the art, especially considering the small number of classes for multilabel classification employed in current benchmarks. For example, the GLUE benchmark [6] includes the Corpus of Linguistic Acceptability (COLA) dataset and the downstream task is to classify each sentence in only two classes, depending on whether it is a grammatical English sentence. When MeSH headings are considered, classification scores range between 0.61 and 0.69 (e.g. [17, 33]).

We also report comparison between pretrained model and transfer learning for downstream tasks such as classification, for which there are currently mixed indications. We conclude that transfer learning through pretrained models is preferred for our problem domain and that appending a fully connected classification layer to a pretrained model can substantially improve the classification accuracy.

For the sake of performance and fast dataset consolidation, in our work data comprises only of papers' titles and abstracts. This is in contrast to the full text of the publications which is utilized in related work (e.g. [2, 33]). As a result, training lasts only for a few hours, a process which would be useful for online inference and constantly updating data.

To the best of our knowledge, ontology-based MeSH indexing using deep contextualized representations, such as ELMo and BERT embeddings for titles and abstracts plus MeSH labels themselves, has not been investigated before. Further, this is the first comparative evaluation of performance between the various inference modes of word embedding models for MeSH classification.

The rest of this paper is organized as follows: in Sect. 2 we review relevant literature in the field of biomedical text classification; in Sect. 3 we summarize current word embedding approaches as the main background and identify the major inference modes that can be used with language models; Sect. 4 presents our methodology and approach, by outlining the structure and use of the MeSH vocabulary, the notion of cosine similarity for classification and the indexing procedure designed. Section 5 describes the datasets used and explains the design, configuration and implementation of the various models we have used to perform automated classification. Optimizations such as dataset balancing, and distributed pre-processing are also discussed. Section 6 contains the results of the various experiments and their comparative analysis, while Sect. 7 outlines our conclusions.

2 Related work

A variety of studies have attempted to tackle with the problem of automatic MeSH indexing. To this end, they mainly combine word embeddings with classifiers. Typical cases are discussed next.

MeSH Now [17] indexes MeSH terms in three steps: obtains an initial list of MeSH candidates, ranks all main headings and selects top-ranked terms. To do this, it uses k-NN and Support Vector Machine (SVM) algorithms, thus achieving a 0.61 *F*-score.

DeepMeSH [20] attempts to examine firstly the frequency characteristics of the MeSH terms with deep semantic representation called D2V-TFIDF and secondly the semantics of the citations with a classification framework. Candidate MeSH headings are rated by a *k*-NN classifier. This system achieves an *F*-score of 0.63.

Another approach [25] for solving this issue uses the cosine similarity metric and a representation of the thesaurus as a graph database. It starts with the use of Elastic Search to convert texts into vectors, then with the cosine similarity metric it identifies the most similar texts. By deriving the terms from these texts and calculating the frequency of occurrence in conjunction with similarity, an evaluation function which classifies documents is defined. This implementation yields a *F*-score of 0.69.

The BioWordVec [34] system combines sub-word information from biomedical texts with MeSH terms in two steps: firstly, it constructs MeSH graph data and samples the MeSH term sequences and, secondly, it employs the FastText sub-word embedding model to learn the distributed word embeddings based on text sequences and MeSH term sequences. In this way, the value of the *F*-score metric is improved to 0.69 and 0.72 for CNN and RNN models respectively.

MeSHProbeNet [32] is a self-attentive deep neural network, which is able to predict a set of MeSH terms for a biomedical article. It consists of three main components: a Bidirectional RNN where the full body of text is led to, a self-attentive MeSH probe and a Multi-view classifier at the output. It achieves micro F-score of 0.69.

FullMeSH [2] takes advantage of the availability of full text of publications from PubMed Central and assigns the entire 29K MeSH headings. This model, starting from segments of an article, extracts their representations with Word2Vec, calculates the candidate terms and finally selects the top ranked ones with AttentionCNN, achieving a micro F-score of 0.67. Training takes 5 days on GPU-powered hardware.

BertMeSH [33] uses BERT in the first layer to obtain deep contextual representations for each word. Then, the next layer concatenates all the outputs to represent a given article. Next, AttentionCNN uses multilabel attention to capture the most relevant parts of an article's full text with each label. Finally, it uses fully connected layers to obtain the predicted score for all 29K MeSH headings, achieving an F-score of 0.69. However, it relies on full text rather than title and abstract for training; the latter takes 4 days on GPU-powered hardware.

3 Word embedding models

3.1 Vector space models for natural language

Vector space models are widely used to represent textual information. Particularly in information retrieval, the Term Frequency—Inverted Document Frequency (TF-IDF) is a well-known method for the computation of real-valued vectors representing documents and queries. However, for general language processing tasks, most traditional methods end up with very sparse matrices of high dimensionality [1]. In addition, inverted indexing does not account for the semantic similarity of text elements, but rather focuses on their common co-occurrence in text corpora.

On the other hand, recent techniques based on neural networks attempt to capture the meaning of a word or sentence based on the distributional hypothesis [5]: *a word can be known by the company it keeps.* These techniques encode text elements into vectors of some constant dimension N, known as *word embeddings* [15]. Neural networks are one tool for this implementation using a training process. They manage to adjust the vector representations of words to close values, when they refer to conceptually close words. Thus, these vectors will be characterized by a high probability of similarity.

In the following, we briefly review some major models and architectures in this category, that have had significant impact in contemporary NLP tasks, including word and paragraph classification. These models are used as the basis for our approach to biomedical literature indexing. Moreover, we examine the various inference approaches taken by these models, which affect our methodology and are also evaluated in our experiments.

3.2 Shallow neural network

Shallow neural networks are simple architectural structures limited to a maximum of two fully interconnected levels. However, this lack of structural complexity does not account for weak computational capacity. These models can successfully tackle simple and well-structured problems, in the field of natural language processing. In many cases, part of the architecture will produce the word representations vector. As observed in Fig. 1, a structure is directly involved in word embedding techniques. Specifically, at the input layer, one-hot vectors represent each word of the body text, the output layer includes the prediction, while at the hidden layer the vector word representations are produced as the connection weights.

Considering specific solutions, the start has been made with the Word2Vec algorithm [18] where unique vector word representations are generated by means of shallow neural networks and the prediction method. Its explicit extension Doc2Vec (Document to Vector) [14] is capable of computing unique vector representations for entire sentences or paragraphs.

A related method is the Global Vectors (GloVe) algorithm [22], which manages to transfer the words into a vector space by making use of an enumeration method, efficiently leveraging statistical information. Then, the FastText algorithm [10] achieves not only the management

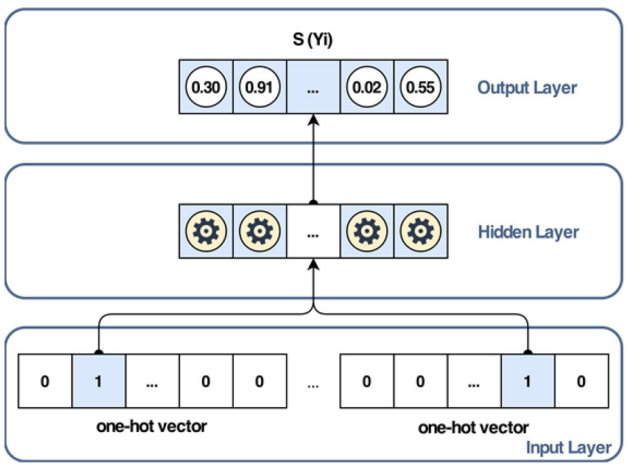

Fig. 1 Shallow neural network architecture for word embedding models

of a large corpus in optimal time but also better word embeddings due to the use of syllables.

3.3 Contextualized representations

Training a shallow neural network would produce the single best embedding for each word. However, the same word can often have multiple meanings depending on its use. This can usually be resolved by looking at the word's context but cannot be captured into a single vector. To accommodate for this *polysemy* more complex structures are required—consisting of many layers (more than 3) [15]. In this case, these models are able to solve complex problems efficiently, but at the same time they have increased computational power requirements. The multiple levels which make up such a structure, with information passing through many layers, require specialized training. Nevertheless, this does not prevent their involvement in the field of natural language processing, with the creation of new word vector representations. Such an architecture related to word embedding is illustrated in Fig. 2. We can observe one-hot vectors for each word at the input layer, the embedding layer straight after that, and then the intermediate layers which eventually lead to the output layer or otherwise to the prediction.

Common representatives in this category include the ELMo algorithm [23], which uses deep neural networks, LSTMs, and different vector representations for a word the meaning of which it differentiates; also BERT [3], which also generates different representations of a word according to its meaning, but instead of LSTMs it uses bidirectional transformer elements [31]. Finally, GPT-2 [24] is a large-scale unsupervised language model. It uses transformers and has the ability to generate synthetic text samples. Also, it can be applied to tasks such as machine translation, summarization, question answering and others.

The above algorithms are summarized in the following Table 1.

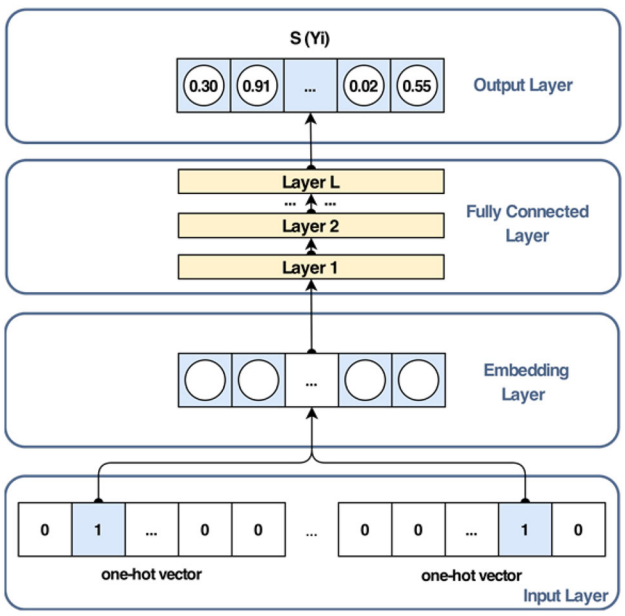

Fig. 2 Deep neural network architecture for word embedding models

3.4 Inference approaches

All architectures discussed above can be used to produce suitable embeddings for words or sentences that attempt to capture their underlying semantics. The question then arises as to how we can tap into these embeddings to produce meaningful results, such as inferences for downstream tasks, including classification, entity recognition, relation extraction and so on. There are mostly three approaches appearing in the literature:

3.4.1 Pretrained model

This approach corresponds to utilizing an existing embeddings model that has been trained in advance, 'as is.' In this way, features out of inputs are extracted, which, in the case of language models, are the values of the vectors that embed words and sentences. Because of this, this method is also referred as *feature-based* in [3]. An example of this is ELMo [23]. Usually a model is trained with large, unlabeled word and sentence datasets, sometimes specializing in different domains, such as biomedicine (e.g., BlueBERT [21]). The model receives no further training. The output used is an average pooling of the word embedding vectors produced or the output of a single summarizing token or the output of some intermediate layers in case of deep architectures. For example, in the case of BERT, authors report that best results are achieved by concatenating the output of the last 4 layers [3]. Elsewhere it is claimed that it is better to use the output of the second to last hidden layer or the hidden state of the first

Table 1 Models and architectures used for word embeddings

Algorithms	Year	Researchers	ANN	Vector per
Word2Vec	2013	Mikolov	Shallow	Word
Doc2Vec	2014	Mikolov	Shallow	Word and Paragraph
GloVe	2014	Stanford	–	Word
FastText	2016	Facebook	–	Syllable
ELMo	2018	Allen Institute	Deep	Character
BERT	2019	Google	Deep	Sub-word
GPT-2	2019	OpenAI	Deep	Sub-word

token, provided the model has been finetuned first (https://github.com/hanxiao/bert-as-service). Nevertheless, it appears that the selection of an appropriate output remains subjective and task dependent.

3.4.2 Finetuning

In the finetuning approach, a single layer is added to the pretrained model, and all parameters are jointly finetuned on a downstream task. Fine tuning involves a much smaller dataset than the one used for pretraining. Therefore, although all parameters are updated, it can be considered a lightweight version of the transfer learning approach discussed below. This is different than the previous approach in that the entire resulting model (pretrained + shallow layer) gets trained once more, but usually with fewer training data. In addition, the output layer added allows to tailor the output for the specific task, e.g., classification, sentence similarity etc. However, there are mixed results on the suitability of finetuning vs. the pretrained model approach, especially for sentence similarity tasks [8].

3.4.3 Transfer learning

Finetuning retains some of the benefits of the pretrained model approach because it gets trained with only so much additional training cycles. However, its added value depends solely on a single simple layer which actually finetunes the model for the downstream task at hand. On the contrary, the transfer learning approach embodies the very idea of transfer learning: an already trained network with all weights configured is reused as a basis for training another model [7]. This way, many underlying word and text features are already learned, and the new model need not train from scratch.

Usually an additional dense layer is appended to the original pretrained model. The latter's weights are frozen, and training commences to configure the parameters of the dense layer. This often ends up with a final SoftMax layer or sigmoid layer, depending on whether it is about single-label multiclass classification or multilabel classification, respectively.

This is different to finetuning, because: (a) the model is trained with more data, (b) a dense layer (deeply connected NN) is used instead of a single shallow layer, (c) the original pretrained model parameters get frozen, whereas they do get updated in finetuning. It also differs from the pretrained approach: An entire dense layer is attached, and additional training takes place, while in the pretrained approach the model is used 'as is.'

4 Classification methodology

Manual indexing of biomedical literature is a hard and time-consuming task. It usually takes 2–3 months to incorporate new articles and the cost for each article is approximately $10 [17].

Our methodology for automated indexing is as follows: First we consider a formal vocabulary to draw our classes from. This is offered by the Medical Subject Headings (MeSH), an official thesaurus which holds some important properties, such as a large number of classes, which differentiate from current classification approaches. Then we look into an efficient procedure to determine which MeSH terms or classes are appropriate for each paper and offer a concrete example and workflow. The design and built of our inference models are discussed in the next section.

4.1 MeSH ontology

Medical Subject Headings (MeSH) is a controlled vocabulary of biomedical terms in a hierarchical structure that is produced and maintained by the United States National Library of Medicine (NLM). MeSH vocabulary is used for indexing journal articles and books in the field of life sciences. It is used by MEDLINE/PubMed article database and NLM's catalog of book holdings. It is also exploited by ClinicalTrials.gov registry [27] to classify which diseases are studied by trials registered in ClinicalTrials.

MeSH is a large and dense thesaurus consisting of 29,917 concepts (called *Descriptors*) as of 2021 [28]. Descriptors are used to index publications for topical headings. In addition, a set of *Qualifiers* are defined, whose aim is to confine the subject to a particular aspect of the main headings. All terms in MeSH are hierarchically organized with most general terms higher in the taxonomy than most specific terms. Moreover, MeSH headings are also organized in a hierarchical tree, where a term can appear in different subtrees. The MeSH tree is composed of 16 main branches (subtrees). Therefore, any term can have some broader relative terms and several narrower terms, as shown in Fig. 3.

The indexing of records in MEDLINE is made by using appropriate MeSH terms, each of which is in the format of combining one main heading and one or multiple Qualifiers. For example, the use of qualifier 'drug therapy' to the subject heading 'Osteoporosis,' will lead to the retrieval of document records that discuss drug therapies for the treatment of osteoporosis.

The MeSH vocabulary is available in different formats—like XML, OWL/RDF—and has been implemented in SKOS [30]. Figure 3 shows an overview of the MeSH structure as well as an example of this ontological

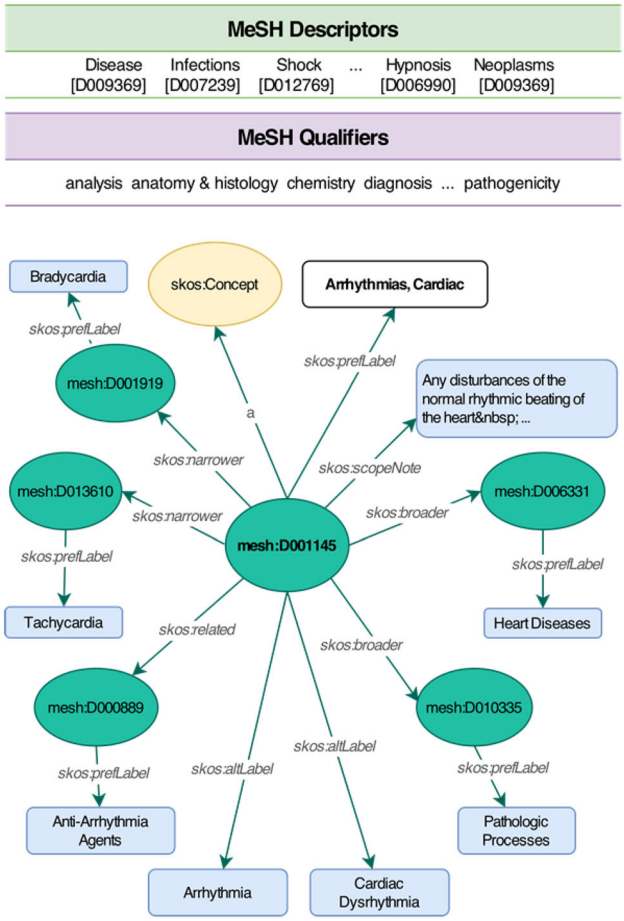

Fig. 3 An example of a MeSH heading and its relations

implementation that we tap into to retrieve the structure and properties of the MeSH headings.

4.2 Cosine similarity and classification

Cosine similarity is a widely used method for retrieving information by checking the similarity of elements such as words, sentences, and even whole texts. It accepts two vectors as arguments and returns their similarity value, which is a real number belonging to the closed space between zero and one. The higher this value, the better the similarity between them.

Our current hypothesis is that thesauri terms that match the meaning of a paper will demonstrate a higher degree of similarity than other, arbitrary terms. The closer a term is to a body of text, the higher its similarity score is to be expected.

However, subject describing terms are indeed not arbitrary; rather, they are contained in a finite, although dense, set, those of MeSH descriptors. They also often represent closely related and fine-grained aspects of the domain of interest. This means that first, the problem dimensionality greatly expands with the total number of terms available for classification. Second, the embeddings produced by any model should likewise be extremely fine-grained, for the model to be able to capture the semantic differences between adjacent terms. Because hardly is this a fact with existing pretrained language models, already trained on a different dataset or different tasks, we already conjecture that at least a model finetuning or the addition of a dense classification layer may be necessary to produce more accurate results.

In any case, the result is a list of terms hopefully corresponding to the subjects of the paper per se. This indexing problem is therefore one of *multiclass classification*. In addition, it is also *multilabel*, markedly an even harder task.

4.3 Approach for biomedical resources classification

Given the various inference modes discussed in Sect. 3, in our approach we consider and compare both using pretrained models and transfer learning, with the addition of a dense classification layer.

In the first case, we rely on the cosine distance to determine the semantic similarity between the embeddings produced by the model and the various MeSH terms. The idea of thresholding has been thoroughly investigated previously by the authors [13]. By considering a certain similarity threshold above which values are classified as positive, we can improve the quality of thematic suggestions. In Sect. 6 we present evaluation results for various values of the decision threshold. The models we build and compare for this scenario include: training Doc2Vec for scratch; using a pretrained ELMo model; and using a pretrained BERT model.

In the second case, we attach a multilabel classifier to a pretrained ELMo architecture. We train this classifier with a biomedical dataset. Decisions are considered positive when the classifier output exceeds 0.5 for a particular class (MeSH term). Further, we relax this requirement by considering the top scoring classes, no matter whether they exceed this threshold or not.

To prepare our training and test datasets we collect metadata from bibliographic resources coming from open-access repositories, such as PubMed [29], EuropePMC [4] and ClinicalTrials [27], together with their manually indexed MeSH terms. Particularly for the Doc2Vec model training, we also take into account the MeSH ontology to facilitate learning MeSH terms that may appear infrequently or not at all in the biomedical datasets. The scopeNote field for each term within the ontology contains a brief explanation about the term and can be used as a training paragraph. In addition, we take care to train the

model as evenly as possible for each MeSH term that is, the training dataset to be equally balanced for each class. We confirm experimentally the positive effects balancing has for evaluation measures (precision and recall) in Sect. 6.

We parse the metadata record for each item and retrieve the title and abstract of the item (body of text). This body of text is then given as input to each model and its vector similarity or decision score is computed against the list of MeSH terms available in the vocabulary. In the example shown in Fig. 4 a paper with https://doi.org/10.1148/radiol.2020200905 related to research about the COVID-19 pandemic is fed to our model. As a result, the model comes up with a set of suggestions together with their score. MeSH terms such as D001185: 'Artificial Intelligence' are included in the manual annotations of the item and are correctly predicted by the model. It is worth noting however that the model can make also suggestions that are not included in the current expert-indexed terms but are otherwise closely related to the topic of the publication. This is evident for example with the term D000658:'Amoxicillin' which, although not picked by the human indexer at the time, it is an antibiotic widely discussed for the treatment of COVID-19.

For evaluation purposes, we should compare the output of our model to some baseline, i.e. the ground truth. The ground truth is offered by the manual annotations inserted by expert indexers, that are included in our reference PubMed dataset, described in Sect. 5.

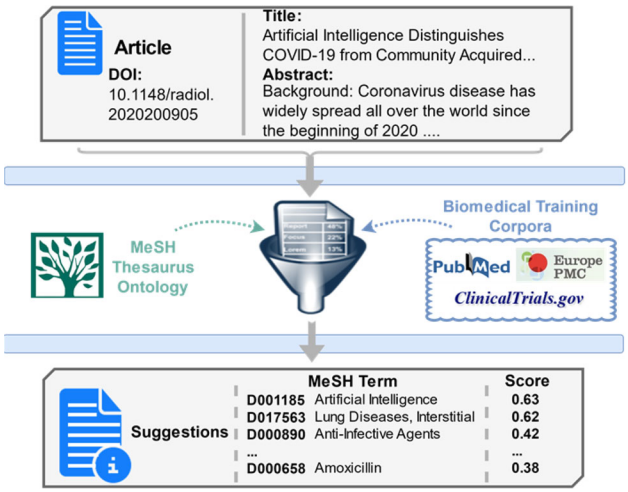

Fig. 4 A specific item gets subject annotations and their similarity scores are computed

5 Model building for biomedical indexing

In this section we explain the design, configuration and implementation of the various models we have used to perform automated classification. In accordance to the arguments made in Sects. 3 and 4, two main methods are considered: (a) directly using a model (pre-) trained for our problem domain, where we configure models based on Doc2Vec, ELMo and BERT and (b) transfer learning employed in a (pre-) trained model as input to a classifier (dense layer), fully trained on the specific biomedical classification task, where we reuse the pretrained ELMo architecture. Therefore, a total of 4 different models are being implemented and evaluated. First, however, we describe the assembly and characteristics of initial data for training and testing, identify their sources, and present the data preparation and balancing procedure.

5.1 Dataset

A dataset from the PubMed repository is used, with records including biomedical citations and abstracts. NLM is responsible for this material being updated daily with new insertions, revisions and deletions. All of these updates are incorporated into a basic file with 30 M records once a year (in December) and they are made freely available to researchers.

The data are available in XML format [26]. A sample PubMed record is displayed in Fig. 5. The elements finally used for training are *ArticleTitle, AbstractText* that represent the body text; and, from the *MeshHeadingList*, the *DescriptorName* with *MajorTopicYN* = *"Y"* or the *DescriptorName* that includes at least one *QualifierName* with *MajorTopicYN* = *"Y"*. The information contained in the *DescriptorName* is a unique ID of the format, e.g.,

```
<?xml version = "1.0" encoding = "UTF-8" ?>
<ArticleTitle> Demonstration of tumor inhibiting properties...
</ArticleTitle>
<Abstract>
  <AbstractText> A report is given on the recent discovery of
immunological properties...
  </AbstractText>
</Abstract>
<MeshHeadingList>
  <MeshHeading>
    <DescriptorName UI = "D007069" MajorTopicYN = "Y" > Ifosfamide
    </DescriptorName>
    <QualifierName UI = "Q000494" MajorTopicYN = "N" > pharmacology
    </QualifierName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName UI = "D007109" MajorTopicYN = "N" > Immunity
    </DescriptorName>
    <QualifierName UI = "Q000187" MajorTopicYN = "Y" > drug effects
    </QualifierName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName UI = "D007165" MajorTopicYN = "N" >
Immunosuppression </DescriptorName>
  </MeshHeading>
</MeshHeadingList>
```

Fig. 5 Sample from a PubMed article metadata record

D007069 and it represents the labels of the text (thesaurus terms).

Another source is EuropePMC, which also provides free access to full texts as well as 5M other related resources. Furthermore, the ClinicalTrials repository also contains 359K records of biomedical research studies. We consolidate records from these different sources aiming at variability and dataset diversity.

Each record in the dataset contains information related to biomedical literature such as the title and the summary of the article, as well as a list of subject headings from the MeSH thesaurus. The selection and integration of the appropriate MeSH heading is carried out by human indexers, who usually choose 10–12 terms to describe a publication. However, in our test datasets, only few are present, typically 1 or 2 or 3 (Table 2), because (a) we only select terms designated as major topics, (b) we conduct experiments with fixed number of total classification labels, thus ignoring some annotations and (c) the balancing process may remove some annotations to keep uniform occurrence of terms.

5.2 Data preprocessing

5.2.1 Using a distributed file system

For the collection of data from the various repositories we first follow a serial approach [12]. For example, PubMed offers a File Transfer Protocol (FTP) service which provides access to 1015 zip files (until December 2019). Each file is inserted into the preprocessing stage individually and as a result the completion of the dataset construction shows some time delays. Moreover, any disconnections to the repositories may add additional delays.

Algorithm 1: Dataset preparation procedure

input: XML files from repository
output: two CSV files
Step 1. **for each** file ∈ repository **do**
Step 2. connect to FTP server
Step 3. get file to local disk
Step 4. store file as line in RDD
Step 5. delete file from local disk
Step 6. **end for**
Step 7. parse file - useful information is extracted
Step 8. convert RDD to DataFrame
Step 9. write useful information to CSV files

Another approach, aimed at reducing preprocessing time, is the use of a distributed infrastructure, the Apache Spark framework. The addition is that now all the information in the XML files is stored in a DataFrame. In detail, we convert each XML file, per line, into a Resilient Distributed Dataset (RDD). Then, after parsing the file, we convert the RDD into a DataFrame, from which we finally extract all the information into CSV files (Algorithm 1). In this way, by transferring part of the processing and extraction of useful information into a distributed infrastructure, we speed up the data preprocessing process.

Table 2 Details of dataset

	Unbal	Test	Bal	Test
Total items	1.0 M	13 K	100 K	10 K
Total annotations	3.2 M	41 K	140 K	14 K
Average # terms per item	3	3	1.4	1.4
Thesaurus coverage rate	70%	25%	4%	4%

5.2.2 Dataset balancing

The next step in the dataset preparation is the adequate coverage of the MeSH thesaurus. That is, the model must learn each term with a sufficient and fixed number of examples. The aim is to avoid bias toward terms with many samples compared to others with only a few. Therefore, when a term is integrated into the set, the number of its samples is limited by an upper limit (Algorithm 2). If there are fewer samples, the term is ignored. If there are more, the search is stopped. Finally, there may not be full coverage of the thesaurus, as shown in Table 2. However, the terms contained are represented by a sufficient and uniform number of samples, resulting in a balanced dataset for training.

Algorithm 2: Dataset balancing procedure

input: two CSV files, thesaurus terms
output: two CSV files with balanced dataset
Step 1. **for each** term ∈ thesaurus **do**
Step 2. compute # of term occurences in CSV
Step 3. **if** number >= 100 **then**
Step 4. collect first 100 samples
Step 5. add samples in two lists
Step 6. **end if**
Step 7. **end for**
Step 8. write lists to CSV files

5.3 Pretrained models: Doc2Vec, ELMo and BERT

5.3.1 Doc2Vec

With this model, we embed words and paragraphs into a lower-dimensional vector space using a shallow neural network. There are no relevant pretrained weights available

for Doc2Vec, so we train our own. Main concept is to have, along with the word vectors, at least one vector for the paragraph. All these vectors participate in the training simultaneously, therefore they gradually adapt to their final state. To do this, we used the Distributed memory (PV-DM) method of Doc2Vec. Specifically, given a set of words and the paragraph id, it predicts the next word. To see the effect of balancing, we train and evaluate two separate models, one with the balanced dataset and another with the unbalanced dataset (Table 2).

By examining the structure of the model, it is observed that it commences with the input which has a size equal to the size of the dictionary. The full body of text is fed to this layer, with each word represented by a "one hot" vector. Next, the hidden layer, which includes 100 nodes with linear activation functions, has the same size as the vector of the word representation. At the output layer, which has a size equal to the dictionary, that is 34,024, the nodes have softmax activation function. Finally, with training the meaning of words is captured and put into the corresponding vector.

The training of the Doc2Vec model is carried out with the help of the Gensim library. Extensive experiments led to the selection of the following parameters: *train epochs* 100, *size vector* 100, *learning parameter* 0.025 and *minimum number* 10. The text body, just before being directed to the input of the model, passes the Gensim utils.simple_preprocess method with a view to convert words into tokens. At the same time, one-character words are removed, as well as those which maintain less than 10 appearances in the text, thus creating better and faster vector representations.

Tests have shown that when there are only a few samples per term, a greater number of epochs can compensate for the sparsity of the training samples. In order to avoid this situation, a balanced dataset was constructed and used. This model, with the adopted weights of the resulting synapses, essentially is a dictionary. In total, it contains the complete set of words from the text used in the training, the vectors of these words, and finally a vector for each full text, respectively.

5.3.2 ELMo

With this model, we move words to the vector space using contextualized representations. Specifically, taking into account the fact that a word can have a different meaning depending on its position in the text, we extract different vectors for each case. This task is completed by utilizing a deep bidirectional language model (biLM) already trained on a specific biomedical large text corpus, such as PubMed. Particularly, this pretrained model is openly available by AllenNLP (https://allennlp.org/elmo) and is included in two files, a JSON options file and a hdf5 weight file, which incorporate the structure and the weights of the model, respectively.

Concerning the model's structure [23], it uses 2 biLSTM with 4096 units and 512-dimension projections. It has also connections from the first to the second layer. The context representation uses in order, 2048 character n-gram convolutional filters, followed by two highway layers and a linear projection with 512 output dimensions. As a result, the model provides three representation levels for each input token, including the character input. In contrast, traditional word integration methods such as Word2Vec give only one layer of representation for tokens.

The model is trained for 10 epochs on domain specific data, specifically 10M abstracts of PubMed articles [9]. For the output, we have selected the linear weighted combination of the 3 ELMo layers.

5.3.3 BERT

BERT is another model which, just like ELMo, can learn different embeddings for different senses of the same word, depending on its context. In contrast to ELMo, it does not operate on the raw characters level but on subwords (called *tokens*). In addition, it does not rely on LSTMs for bidirectional encoding, but uses a *transformer*—an attention-based mechanism with positional encodings to represent word positions. BERT and its variants currently outperform other architectures in recent NLP benchmarks (https://gluebenchmark.com/leaderboard).

We use a specific BERT model pretrained on PubMed abstracts, known as BlueBERT [21]. Its training corpus includes approximately 4000 M words extracted from PubMed. Its base version comprises 12 hidden layers of stacked transformers, each of 768 hidden units, 12 attention heads and 110M parameters in total. As a result, the output embedding size for each token is 768.

The BERT model receives a fixed length of sentence as input: for out-of-vocabulary tokens, there is a limit of 512. Following the inference approaches discussed in Sect. 3, in our experiments we adopt the following strategy: We refrain from using the last layer hidden-state of the first token as output, because the model is not finetuned and is considered not a good summary of the semantic content of the input (https://huggingface.co/transformers/model_doc/bert.html#tfbertmodel). Rather, we concatenate the output of the last 4 hidden layers, producing a tensor of (max 512, 3072) dimension and perform average pooling on the sentence token embeddings. Next, we compute similarity between the 3072-sized vector of an item's body of text and the 3072-sized vector of the MeSH term embedding. To find the MeSH embedding, we use both the term preferred label (the name of the term) as well as its scope note, which is usually larger and more descriptive. The scope

similarity should have a major weight in determining a semantic relation, but label-only similarity can also offer some boost in case of unambiguous terms. The final similarity score is therefore computed as:

$$0.7 * \text{scope\_similarity} + 0.3 * \text{label\_similarity}$$

5.4 Transfer learning for building an embeddings classifier

For implementing the transfer learning approach, a pretrained ELMo model is utilized. Vectors produced by the pretrained model are led to a single dense layer, while an output layer is responsible for making label predictions. This structure, with the help of training, utilizes the primary knowledge of the representations by transferring learning captured by the ELMo model to the attached classifier. To construct our classifier, we use the Keras library and get the set of weights and parameters of an ELMo model trained on the 1 Billion Word Benchmark, available at TensorFlow Hub.[1]

The overall architecture of the ELMo-based classifier is shown in Fig, 6. First, we have the input with the body of text. This is passed to the pretrained ELMo model, which produces a vector of size 1024 for each word in the input text. The next layer performs average pooling of the ELMo word embeddings, thus producing a vector representation of the entire text. Then, there is the dense classification layer, containing 256 nodes with the ReLU activation function. Finally, the output layer contains sigmoid units, one for each label, for our multilabel classification task.

During training, a Tensor Flow session is used with the following parameters: epochs 10 and batch_size 10, defining in this way a limit to the number of training epochs, as well as to the packages transferred. The pretrained model parameters remain frozen and the weights of the dense and output layer are updated accordingly. After the completion of the training, the final weights are stored so that the system is immediately available for utilization in various tasks.

6 Comparative evaluation

6.1 Evaluation methodology

Metrics commonly used in categorization problems are precision, recall, and *F*-measure. For the application of these metrics the existence of pre-categorized data also constitutes a required condition, a process performed by experts. In detail, for the evaluation of each system described in the previous section, measurements are executed after examining the entire test set. Thus, having the total values, such as total number of relevant, suggested and correct terms we can compute an average for each metric. For our purposes, a single information need is defined as a single-term match with an item (1–1) and we report the mean value of these metrics over all these matches, i.e. they are micro-averaged.

To define these metrics, let y_i^l and $\hat{y}_i^l$ the ground truth and the prediction for sample i and for some label l respectively. Their domain is limited to the set $\{0, 1\}$ while the entire label set is denoted by L. To derive the final value for each metric, the enumeration method, of these binary elements, is applied in the following formulas.

Precision (P) checks how many terms suggested by the system are correct, i.e. they coincide with the terms suggested by the experts.

$$P_{\text{micro}} = \frac{\sum_i \sum_{l \in L} y_i^l \cdot \hat{y}_i^l}{\sum_i \sum_{l \in L} \hat{y}_i^l} \tag{1}$$

Recall (R) checks how many terms proposed by the system are contained in the list of terms suggested by the experts, i.e. how many correct terms the system can retrieve.

$$R_{\text{micro}} = \frac{\sum_i \sum_{l \in L} y_i^l \cdot \hat{y}_i^l}{\sum_i \sum_{l \in L} y_i^l} \tag{2}$$

As a general evaluation metric, we also report on the *F*-measure (F), which is the harmonic mean of precision and recall. This is obtained immediately after calculating precision and recall by applying the following formula:

$$F_{\text{micro}} = 2 \cdot \frac{P_{\text{micro}} \cdot R_{\text{micro}}}{P_{\text{micro}} + R_{\text{micro}}} \tag{3}$$

Experiments and results presented in the following evaluate the 4 different models discussed in Sect. 5, either pretrained or utilizing transfer learning for classification.

6.2 Experimental results of pretrained models

6.2.1 Doc2Vec

The behavior of the Doc2Vec model is examined by comparing its performance against both the balanced and unbalanced test sets. In each case the classification task is to correctly identify correct MeSH labels out of a total of 1000 available terms, a demanding multilabel assignment.

A body of text (title and abstract) is given as input to the model and a vector representation is created. The model then searches for the most similar vectors in the vector set it has already incorporated from the training. At most 10 suggestions are produced by the model. The threshold

[1] https://tfhub.dev/google/elmo/3.

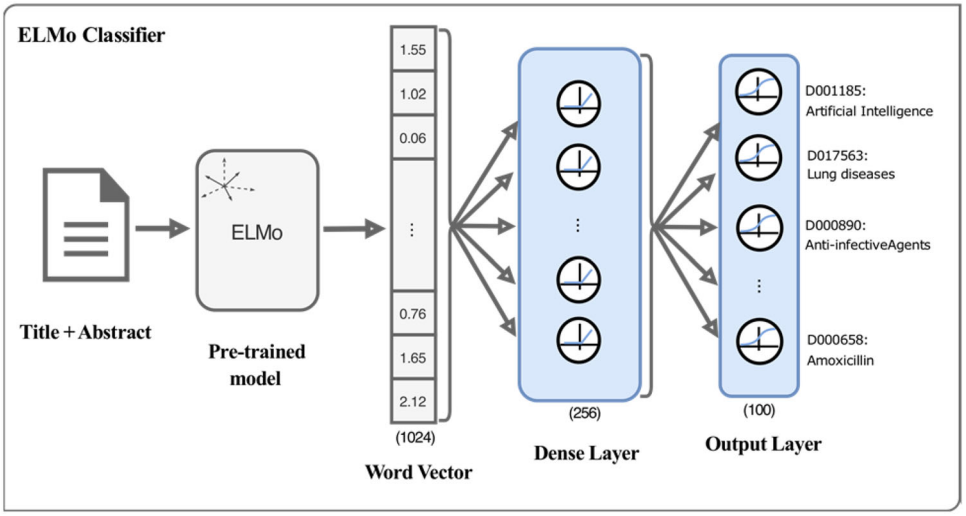

Fig. 6 A multilabel classifier built by concatenating a pretrained ELMo model with a dense classification layer of sigmoid activations

value reported is the similarity score above which suggestions are classified as positive.

Best results are observed when considering only the first 3 suggestions for the unbalanced dataset (rank@3); and only the first suggestion for the balanced dataset (rank@1). This is consistent with the average number of terms appearing in the respective test sets, which is 3 and 1.4, respectively. Figure 7 shows the micro-averaged *F*-score for each case. Obviously, the balanced model demonstrates better scores that its unbalanced counterpart. This is because none of the 1000 terms involved gets any training bias; conversely, no term gets underfitted.

6.2.2 ELMo

In order to evaluate the ELMo approach, a total of 1000 samples are used, distributed evenly between 100 unique labels (10 samples per label). This test set is also balanced and significantly reduced compared to the corresponding test set of the previous model. We have used a reduced number of classes due to the increased computing resources required by the model for vectors inference. We also examine the model's behavior with only 10 classes, still above the current practice in state-of-the-art models.

A body of text (title and abstract) is given as input to the model. The output is the average pooling of the ELMo embeddings for each word in the body of text. Vectors for the MeSH terms are likewise computed, using the term label plus its scope note as input. The model then searches through the set of the term vectors to find those that are most similar to the generated one. Best scores are achieved when considering only the first prediction (rank@1), i.e. the most similar term produced, because the test set in both 10- and 100-labels cases contains on average no more than two terms for each item. For the same reason, *P* and *R* are reported equal or close enough (Table 3).

6.2.3 BERT

As with ELMo, also for the pretrained BERT model we measure effectiveness with 100 and 10 distinct MeSH labels. The model returns the best matching terms, as described in Sect. 5.3, along with their similarity score. Table 3 compares retrieval effectiveness of the various pretrained models.

For our problem domain it turns out that pretrained ELMo outperforms BlueBERT, with an almost double *F*-score. A possible reason is that the deeply bidirectional architecture of the BERT transformers vs. the concatenation of the left-to-right and right-to-left representations of ELMo is not effective in this setting: The text corpus comprises highly specialized biomedical writings in a single language (English). Therefore, concept ambiguity, for which deep bidirectional transformers are all about, is rare. In addition, BERT may be hampered by a higher rate of out-of-vocabulary words in respect to its training set. ELMo, operating on the character level, is much less affected by this situation.

Overall, it appears that Doc2Vec performs best, as it yields acceptable scores even for 1000 distinct classification labels, for which it has been specifically trained in a supervised manner. In contrast, in the two previously pretrained models the training was carried out with an unlabeled dataset, without considering MeSH labels specifically. We conclude that some finetuning for the MeSH classification task and the different number of labels is necessary for the other models to achieve their potential.

6.3 Experimental results of transfer learning model

Our ELMo-based classifier implements the transfer learning approach described in Sect. 5.3, as opposed to using a

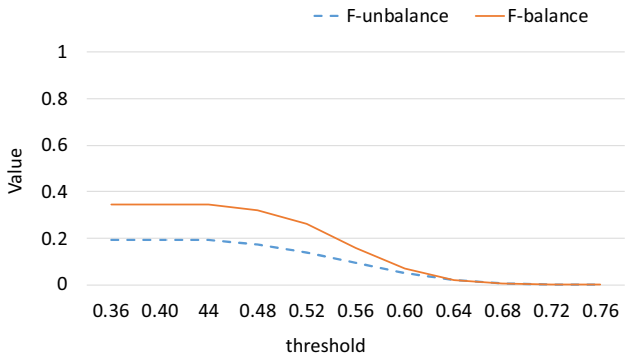

Fig. 7 Doc2Vec *F*-score results for various values of the decision threshold regarding a balanced and an unbalanced dataset

Table 3 Precision, recall and *F*-score for the 3 pretrained approach models with increasing number of labels

	Labels	*P*	*R*	*F*-score
Doc2Vec	1000	0.41	0.29	0.34
ELMo	10	0.70	0.70	0.70
	100	0.31	0.30	0.30
BERT	10	0.42	0.42	0.42
	100	0.16	0.13	0.15

trained model directly. We train the classifier on the 10-labels and 100-labels dataset and evaluate precision and recall on the corresponding test sets (Table 2). We use 0.5 as a decision threshold, meaning that values with sigmoid output > 0.5 are classified as positive.

Figure 8 demonstrates the effect of increasing the (balanced) training set size in each case. When only a small number of labels is selected (10), metrics improve, but generally the classifier performs well even when trained with 100 samples per class. Where it is favored is when allowing 100 different classes: Results considerably improve with the larger dataset, but performance is overall low, due to the high dimensionality. When considering 1000 classes, training could not complete beyond 100 samples per label because of out-of-memory errors.

To compensate for potential underfitting and ill-calibration of the classifier outputs, a common approach is to consider other thresholding techniques [16]. In our case, rather than using any single cutoff, we proceed with a fixed number of the highest outputs, i.e. we consider the sigmoid probability as a similarity score. Given that the ground truth for the 100-label dataset contains mostly one label per item, we take only the 1st highest value as positive and compute precision ($P@1$) and recall ($R@1$). These values are shown in the charts of Fig. 9, in comparison with the scores achieved with the 0.5 thresholding method.

We notice a slight improvement in *F*-score in the case of 10 or even 100 different classes, thus validating this choice

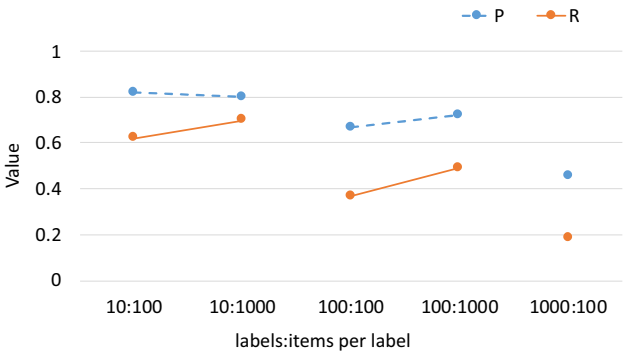

Fig. 8 ELMo Classifier results

(Fig. 9, model trained with 100 samples per label). However, for the 1000-label test set, precision and recall significantly drop. This is a clear indication of severe underfitting, because for a dimensionality that high, only 100 samples per label are insufficient. In addition, the output values are quite below the decision threshold of 0.5 and only few suggestions pass this mark, leading to better precision scores in the first case (0.5 cutoff).

A performance summary of the 4 different implemented models is shown in Table 4 for various classification cardinalities. For completeness, we report the *F*-score of every model in all cardinalities we could produce results.

Apparently, the ELMo classifier, having the benefits of transfer learning for embeddings as well as the discriminative power of a dense classification layer, outperforms other methods. However, this comes at the cost of separate training with 100 or 1000 samples per label. At the same time, only the 'lightweight,' shallow-network Doc2Vec approach could produce meaningful results for 1000 labels classification, while its scores for fewer labels are up-close.

6.4 Scalability

Training can be a particularly demanding process in computing resources as it depends directly on the architecture of the model. This is a valid reason why it is recommended

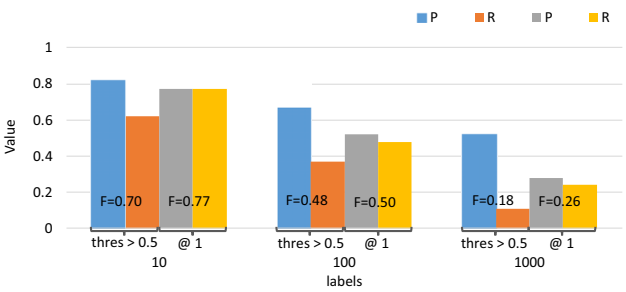

Fig. 9 ELMo Classifier results for different classification cardinalities and decision thresholds

Table 4 *F*-score for the various classification models

	10	100	1000
Doc2Vec	0.70	0.53	**0.34**
ELMo Pretrained	0.70	0.30	0.15
BERT Pretrained	0.42	0.15	0.04
ELMo Classifier (100 samp.)	**0.77**	0.50	0.26
ELMo Classifier (1000 samp.)	0.75	**0.60**	–

Bold indicates highest score per number of classes

to avoid training language models for scratch and adopt the benefits of pretrained models.

In the case of shallow neural networks, such as Doc2Vec, processing power requirements are limited. Training can be completed without significant delays, even with 100 epochs. However, even in the case of transfer learning, such as with our ELMo Classifier, the requirements change. The increased complexity of these models along with the need for sufficient training, increase demands in computing power. Consequently, the utilization of parallel processes is required in order to improve training time as well as to complete the effort itself.

There are cases of executing algorithms with extended datasets and/or many epochs which can lead to either prohibitive completion times or even interruption of their execution. For example, training the ELMo Classifier model with a small set of 1000 samples (10 labels with 100 samples each) takes about 20 minutes, on average commodity hardware (Intel i7, 2.6 GHz, 4-core CPU with 16 GB RAM).

Based on this concern, we perform experiments on high-performance infrastructures with a large number of graphics processing units (GPUs). The GPUs allow for performing calculations at a much higher rate than conventional CPUs. We use a server with two Intel Xeon processors, including 12 cores, 32 GB of RAM and Nvidia V100 GPU. The V100 has 32 GB of memory and 5120 cores and supports CUDA v. 10.1, an API which allows for the parallel use of GPUs by machine learning algorithms. With the execution transferred to the GPU, the 1,000-sample dataset takes only 30 s to be trained. This faster execution (40x) ensures completion of experiments and can assist scalability with larger datasets that would otherwise be impossible to achieve.

Figure 10 presents training times for the ELMo Classifier variants in a logarithmic scale. We notice that it is not only the total number of samples but also the architecture of the model that can affect scalability. The 100-label experiments involve 10X more output units thus increasing the total number of parameters and hardening the training process. For example, training the 1000-label model with more than 100 samples per label could not conclude within a reasonable timeframe, even on the V100 hardware.

7 Conclusions

Recent models featuring context-based word- and sentence- representations appear to revolutionize traditional NLP and make Natural Language Understanding (NLU) a commonplace reality even for everyday tasks. When combined with evolving hardware configurations, they can offer efficient solutions to text processing tasks that would be otherwise impossible to perform on a large scale. However, training such models from scratch is hard and very computationally expensive.

Having implemented and evaluated different models and approaches, we have shown that one can reap the benefits of transfer learning when using available pretrained models in conjunction with a classifier. This can lead to state-of-the-art results, also for biomedical indexing with MeSH labels, even when considering titles and abstracts only. On the other hand, using base models directly and classifying according to semantic similarity cannot compete the added complexity of a classification layer, except maybe when training is conducted for the specific task at hand. This also holds even when using specialized models pretrained on domain datasets. We have experimentally confirmed our earlier hypothesis that, in our domain of interest, the complexity of authoritative controlled vocabularies necessitates extremely fine-grained embeddings, and these can only be achieved with some form of finetuning and/or training. Particularly, utilizing lightweight training in a pretrained model, thus customizing it for the specific dataset at hand, is crucial especially for online decision making and constantly updating data.

Additionally, a careful dataset balancing as well as the capability of deep networks to leverage distributed GPU architectures are demonstrated beneficial and should be exercised whenever possible. Finally, we see improvements in the way classification suggestions are being

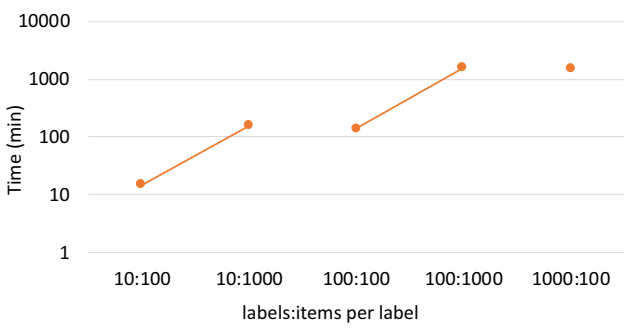

Fig. 10 Training times for the ELMo Classifier

offered, especially in view of the density of the thesaurus used: other ontological relations, such as generalization or specialization of concepts can be taken into account in order to discover and prune hierarchy trees appearing in the recommendations list.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

1. Christopher DM, Prabhakar R, Hinrich S (2008) Introduction to information retrieval. Cambridge University Press, Cambridge
2. Dai S, You R, Lu Z, Huang X, Mamitsuka H, Zhu S (2020) FullMeSH: improving large-scale MeSH indexing with full text. Bioinformatics (Oxford, England) 36(5):1533–1541. https://doi.org/10.1093/bioinformatics/btz756
3. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Proceedings of NAACL-HLT 2019, pages 4171–4186 Minneapolis, Minnesota, June 2–June 7
4. Europe PMC Consortium. Metadata of all Full-Text Europe PMC articles. [Online]. Available: https://europepmc.org/ftp/pmclitemetadata/
5. Firth JR (1957) A synopsis of linguistic theory 1930–55, 1952–59, 1–32. Blackwell, Oxford
6. General Language Understanding Evaluation. GLUE [Online]. https://gluebenchmark.com
7. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. The MIT Press, Cambridge
8. Hedtke, J., Petrov, S (2019) Semantic similarity search. Machine Learning—Projects Fall 2019–CS229, cs229.stanford.edu
9. Jin Q, Dhingra B, Cohen W. W, Lu X (2019) Probing Biomedical embeddings from language models. arXiv:1904.02181 [cs.CL], NAACL
10. Joulin A, Grave E, Bojanowski P, Mikolov T (2016) Bag of tricks for efficient text classification. arXiv:1607.01759v3 [cs.CL]
11. Koutsomitropoulos D, Andriopoulos A (2020) Automated MeSH indexing of biomedical literature using contextualized word representations. In: Proc. of the 16th Int. Conference on Artificial Intelligence Applications and Innovations (AIAI), IFIP AICT vol. 583, pp. 343–354. Springer 2020
12. Koutsomitropoulos D, Andriopoulos A, Likothanassis S (2019) Subject classification of learning resources using Word embeddings and semantic thesauri. In: IEEE Innovations in Intelligent Systems and Applications (INISTA), Sofia, Bulgaria
13. Koutsomitropoulos D, Andriopoulos A, Likothanassis S (2020) Semantic classification and indexing of open educational resources with word embeddings and ontologies. Cybern Inf Technol 20(5):95–116
14. Le QV, Mikolov T (2014) Distributed representations of sentences and documents. In: 31st International Conference on Machine Learning, ICML, Beijing, China
15. Li Y, Yang T (2017) Word embedding for understanding natural language: a survey. Springer, NewYork
16. Lipton Z, Elkan C, & Naryanaswamy, B (2014) Optimal Thresholding of classifiers to maximize F1 measure. Machine learning and knowledge discovery in databases. In: European Conference, ECML PKDD proceedings. ECML PKDD (Conference). 8725. https://doi.org/10.1007/978-3-662-44851-9_15.
17. Mao Y, Lu Z (2017) MeSH now: automatic MeSH indexing at PubMed scale via learning to rank. J Biomed Semant 8(1):1–9. https://doi.org/10.1186/s13326-017-0123-3
18. Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. In: ICLR Workshop
19. Mork JG, Jimeno-Yepes A, Aronson, AR (2013) The NLM medical text indexer system for indexing biomedical literature. In: Conference and Labs of the Evaluation Forum 2013 (CLEF 2013), Valencia, Spain
20. Peng S, You R, Wang H, Zhai C, Mamitsuka H, Zhu S (2016) DeepMeSH: deep semantic representation for improving large-scale MeSH indexing. Bioinformatics 32(12):i70–i79. https://doi.org/10.1093/bioinformatics/btw294
21. Peng Y, Yan S, Lu Z (2019) Transfer learning in biomedical natural language processing: an evaluation of BERT and ELMo on ten benchmarking datasets. In: Proceedings of the Workshop on Biomedical Natural Language Processing (BioNLP).
22. Pennington J, Socher R, Manning CD (2014) GloVe: global vectors for word representation. In: Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1532-1543, Doha, Qatar
23. Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. arXiv:1802.05365v2 [cs.CL], NAACL
24. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I (2019) Language models are unsupervised multitask learners. OpenAI Blog 1(8):9
25. Segura B, Martínez P, Carruan MA (2017) Search and graph database technologies for biomedical semantic indexing: experimental analysis. JMIR Med Inform 5(4):e48. https://doi.org/10.2196/medinform.7059
26. U.S. Department of Health & Human Services, MEDLINE®–PubMed® XML Element Descriptions and their Attributes, 2018. [Online]. Available: https://www.nlm.nih.gov/bsd/licensee/elements_descriptions.html
27. U.S. National Library of Medicine. ClinicalTrials.gov [Online]. Available: https://clinicaltrials.gov
28. U.S. National Library of Medicine. Medical Subject Headings, 2019. [Online]. Available: https://www.nlm.nih.gov/mesh/meshhome.html
29. U.S. National Library of Medicine. PubMed.gov [Online]. https://www.nlm.nih.gov/databases/download/pubmed_medline.html
30. Van Assem M, Malaisé V, Miles A, Schreiber G (2006) A method to convert thesauri to SKOS. In: The Semantic Web: Research and Applications: 3rd European Semantic Web Conference, ESWC, Proceedings (Vol. 4011, p. 95), Springer, Budva, Montenegro
31. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Advances in Neural Information Processing Systems, pp. 6000–6010
32. Xun G, Jha K, Yuan Y, Wang Y, Zhang A (2019) MeSHProbeNet: a self-attentive probe net for MeSH indexing. Bioinformatics 35(19):3794–3802. https://doi.org/10.1093/bioinformatics/btz142
33. You R, & Liu Y, Mamitsuka Zhu S (2020) BERTMeSH: Deep contextual representation learning for large-scale high-performance MeSH indexing with full text. DOI: https://doi.org/10.1101/2020.07.04.187674
34. Zhang Y, Chen Q, Yang Z et al (2019) BioWordVec, improving biomedical word embeddings with subword information and MeSH. Sci Data 6:52. https://doi.org/10.1038/s41597-019-0055-0