Hierarchical Deep Learning for Multi-label Imbalanced Text Classification of Economic Literature

Sanne Lin[a], Flavius Frasincar[a,*], Jasmijn Klinkhamer[a]

[a]*Erasmus University Rotterdam, PO Box 1738, 3000 DR, Rotterdam, the Netherlands*

Abstract

With the vast amount of economic literature available in this day and age, efficient and accurate text classification becomes increasingly important. We propose an extended version of the Hierarchical Deep Learning for Text Classification (HDLTex) approach, called HDLTex++. HDLTex++ applies hierarchical learning using neural networks to classify documents and is adapted for the multi-label classification of class imbalanced data. We use HDLTex++ to assign to economic publications category labels from the Journal of Economic Literature classification system, which has a hierarchical tree structure with three levels. The performance of HDLTex++ is compared to two methods based on Support Vector Machines (SVMs), one where the class hierarchy is fully incorporated, and one where only the tertiary subcategories are taken into consideration. Performance is evaluated using the standard F1-score and a novel hierarchical F1-score that accounts for both class imbalance and class hierarchy. Our findings show that HDLTex++ is more effective in the prediction of primary category labels, compared to both SVM models, and in the prediction of secondary category labels, compared to the hierarchical SVM model.

Keywords: Class Imbalance, Hierarchical Learning, Multi-Label Classification, Neural Networks, Support Vector Machines

1. Introduction

Much academic research has been conducted in the field of economics. Many of these research papers, journal articles, dissertations, and other forms of publications can be found in repositories. Their topics can be determined using the Journal of Economic Literature (JEL) classification codes

*Corresponding author; tel: +31 (0)10 408 1262

Email addresses: sanne.lin33@gmail.com (Sanne Lin), frasincar@ese.eur.nl (Flavius Frasincar), 584059jk@eur.nl (Jasmijn Klinkhamer)

assigned to them, which are part of the JEL classification system. This system is based on a set of categories structured in a class hierarchy with three levels. Whereas browsing and filtering publications is more convenient using JEL codes, assigning classification codes to publications is still a time-consuming process, if done manually. The aim of this paper is to propose an automatic classifier that is able to categorize the publications according to their content and assign the appropriate JEL codes.

The development of methods aimed at solving classification problems and training classifiers has formed a large part of the academic literature in machine learning. Such classification problems can have many different applications, such as diagnosing cancer, e.g., [1], classifying environmental conditions, e.g., [2], detecting fraud, e.g., [3], predicting bankruptcy, e.g., [4], and recognizing music genres, e.g., [5]. A common type of classification is text classification, where text is analysed and assigned labels based on its content. Applications of text classification can be found for instance in email spam filtering, e.g., [6], intent classification, e.g., [7], and sentiment analysis, e.g., [8].

Classification problems can have distinctive characteristics [9]. One such characteristic is the number of categories to which an instance can be classified. In binary classification problems, an instance is classified to one of two categories, such as positive or negative sentiment [10]. Problems where instances can be sorted into one (or more) of multiple categories are called *multi-class* classification problems, for which classification algorithms can often be a natural extension to binary classification techniques [11]. Another characteristic is the number of labels that can be assigned to an instance, which is exactly one in *single-label* classification but can be any number of labels in *multi-label* classification [12]. Inherently, multi-label classification problems are also multi-class problems.

In the context of assigning JEL codes to economic publications, the classification problem is one that is multi-label, since a publication can fit multiple categories. For instance, papers about market structure can be classified under subcategories of both 'Microeconomics' and 'Industrial Organisation'. Due to the tree structure of the JEL classification system, we are also facing a hierarchical classification problem, where we focus on predicting categories from the leaf nodes of the classification hierarchy. [13] approaches a general multi-label hierarchical classification problem. In contrast, we focus on developing a text classifier to assign one or multiple labels to publications. To this end, we use the content of a publication's title and abstract, as well as the pre-defined

hierarchical structure of the JEL classification system.

A complication that can occur in classification problems is *class imbalance* [14], which has been addressed in various domains [15, 16]. The JEL classification system has categories that vary in specificity. For instance, a child category 'General' exists which captures the general instances of the parent category, such as textbooks and surveys, while the remaining child categories capture more specific topics within the parent category. It is even possible for some overlap to occur, e.g., some (but not all) publications are classified to both the 'General' category and one or multiple specific categories, while some other publications belong to multiple specific topics, but not to the 'General' category. Since the 'General' category covers a broader range of publications (including some from sibling categories) and some subcategories are more niche than others, class imbalance arises, with some categories containing more instances of publications than others. Class imbalance can present an issue in training a classifier, since a natural tendency of the classifier will be to assign the more common JEL codes to publications, as this classification will achieve a relatively high accuracy. However, this will also result in the misclassification of publications with more niche topics. Hence, to build a classifier that is able to categorise publications with common and niche topics alike, this class imbalance needs to be addressed.

In this paper, we introduce an extended version of the Hierarchical Deep Learning for Text Classification (HDLTex) approach, a hierarchical multi-class classifier which takes a local classifier approach to text classification by training deep learning classifiers [17] at each parent node of a two-layer class hierarchy [18]. The first level classifier is used to classify documents to primary categories of the classification system. At each primary category, subclassifiers are built which are trained on the subset of documents belonging to the corresponding primary category only. The authors find that their approach outperforms the more traditional classifiers such as Naive Bayes (NB) and Support Vector Machines (SVMs). Our classifier, which we call HDLTex++, extends HDLTex in several ways. First, since the JEL classification system contains three layers, we append an additional layer of deep learning classifiers to the original algorithm. Second, we apply an alternative loss function that allows for multi-label learning. Third, we apply cost-sensitive learning to counter class imbalance, such that a higher cost is attached to the misclassification of documents from smaller categories. To evaluate the results, we use the standard macro-averaged F1-score, which does not account for the hierarchy. We also propose a novel performance measure,

the macro-averaged hierarchical F1-score, which takes into account both the class imbalance and the class hierarchy.

The remainder of this paper is structured as follows. Section 2 gives an overview of related work in multi-label and hierarchical text classification. Section 3 introduces the data used. Section 4 describes our methodology. The results are presented in Section 5. In Section 6, conclusions are provided and future research possibilities are discussed.

2. Related Work

Text classification, also called text categorization and topic spotting, is the task of organizing natural language texts into pre-defined thematic categories [9]. Before a classification algorithm can be applied to assign category labels to a document, its content needs to be converted into a format that the algorithm can interpret. Transforming the raw text data into a structured dataset is done in two steps. First, in the text pre-processing step, the data is cleaned, for example by discarding capitalisation and punctuation marks. Following the data pre-processing, relevant features can be extracted from the cleaned document and mapped to a numerical representation suitable to be used as input data for classification algorithms.

A common approach to extracting features is through weighted words, e.g., [19], where a vector is created in which each word in the document is represented by a weight, commonly the Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF is calculated by taking the product of, as the name suggests, the term frequency (TF) and the inverse document frequency (IDF). The term frequency refers to the number of occurrences of the word in the document, while the document frequency is the fraction of documents in which that word occurs. The inverse document frequency is then calculated by taking the logarithm of the inverse fraction. While easy to calculate, a limitation of TF-IDF is its inability to capture semantic similarity between words, since each word is considered independently [20].

An alternative to weighted words is word embedding, which uses distributed vector space models to learn distributed word representations that capture the semantic and syntactic relations between words. Two popular models that provide pre-trained word embeddings are word2vec and Global Vectors (GloVe). Word2vec uses either the Continuous Bag-of-Words (CBOW) model, which tries to predict the target word using its surrounding words (context), or the continuous

skip-gram model, which tries to predict the context using the current word, e.g., [21]. GloVe incorporates elements from both global matrix factorisation and local context window methods into a weighted least squares problem applied on co-occurrence counts of word pairs [22]. The learned word embeddings can be aggregated (for instance, by taking the (weighted) average or the maximum) to find a text representation for the entire document [23].

While word vectors trained using word2vec and GloVe model the semantic and syntactic characteristics of the words, each word representation is context-independent and thus does not allow for the incorporation of concepts such as polysemy [24]. In response to this shortcoming, several models have been proposed for learning contextual word embeddings, including context2vec [25], CoVe [26], and ELMo [24]. Recently, the Bidirectional Encoder Representations Transformers (BERT) model was proposed by [27], which uses a deep bidirectional architecture using transformers to learn context-dependent word representations. BERT was shown to obtain state-of-the-art results in the General Language Understanding Evaluation (GLUE) benchmark.

2.1. Multi-Label Classification

Classification problems can be divided into *single-label* problems, where each data instance is assigned exactly one label, and *multi-label* problems, where each instance can receive any number of labels. Within single-label classification, problems can be considered *binary-class* (i.e., having two classes) or *multi-class* (i.e., having more than two classes).

A commonly used classification method is the Support Vector Machine (SVM), e.g., [28]. Developed by [29], the (binary-class) support vector machine is used to find a hyperplane that maximizes the margin between two classes. Though originally intended to find linear boundaries, SVMs can be easily adapted to find non-linear boundaries by using different kernel functions. In addition, SVMs can be used to find such functions irrespective of the dimensionality of the problem, making it quite suitable for text classification, for which input vectors can be large [30]. SVM classifiers can be applied in a J-class multi-label setting using the One-vs-Rest (OVR) approach, where a binary-class SVM is trained for each class. Alternatively, the One-vs-One (OVO) approach can be used, where an SVM is trained for each pair of classes, resulting in $J(J-1)/2$ classifiers. A majority voting scheme is used to classify instances to categories. A third option is an SVM classifier that considers all classes using a single optimization problem [31]. A multi-class classifier like this may be adapted for multi-label use by selecting the classes for which the margins surpass a certain

threshold, instead of only the class that classifies the test instance with the greatest margin.

In recent years, the use of deep learning techniques in text classification has become increasingly common. [32] propose the Backpropagation for Multi-Label Learning (BP-MLL) algorithm , a neural network approach that minimizes a pairwise ranking loss function. The BP-MLL algorithm is adapted in [33] by replacing the loss function with the cross-entropy loss and a sigmoid activation function for the output layer. This loss function was later also used in [34], where a Convolutional Neural Network (CNN) approach is proposed for extreme multi-label text classification.

Other classification techniques that can be applied to multi-label problems include k-Nearest Neighbours (kNN) [35], Multi-Label kNN (ML-KNN) [36], an extension to kNN that uses the maximum a posteriori (MAP) principle to predict label sets, FastXML [37], a tree-based classifier for extreme multi-label learning, and BoosTexter [38], a boosting algorithm derived from the AdaBoost algorithm.

2.2. Hierarchical Classification

Hierarchical classification problems describe a subset of classification problems where the categories are arranged in a class hierarchy. There are three approaches to hierarchical classification: flat, local, and global classification [39]. In flat classification methods, the hierarchical structure of the categories is ignored, and only the leaf categories are incorporated, such that classification methods intended for non-hierarchical classification problems can be easily applied. However, flat classifiers have to be able to discriminate between all leaf categories, of which there may be many. Furthermore, potentially valuable information derived from the parent-child relations between the categories is neglected.

Alternatively, global or local classifiers can be used, which do incorporate the hierarchical structure of the categories. A global classifier is used to build a single model that is able to classify instances to all categories in the hierarchy. This approach was taken in [40], where the label set of each instance is appended with the corresponding ancestors of the labels and a boosting procedure is applied on the whole category space. As this approach flattens the hierarchy during the training phase, inconsistencies can occur, where an instance is placed into a category, but not (one of) its corresponding ancestors.

Local classification methods use the hierarchical structure to train flat classifiers for subsets of the categories. Multiple types of local classification approaches exist [39]. The Local Classifier per

Node (LCN) approach is used when training a binary classifier at each node of the hierarchy to predict whether an instance belongs to the class associated with the classifier. [41] propose an LCN approach for classifying hierarchical Web content, using a binary-class SVM at each node of the hierarchy. The authors report that their hierarchical method outperforms the flat baseline SVM model in terms of F1-score.

The Local Classifier per Parent Node (LCPN) approach is used when training a multi-class classifier at every *parent* node to predict which of the child categories the instance belongs to. Each classifier is trained only on the instances belonging to its descendent nodes. By training one classifier for a set of child nodes, this approach can take into account correlations between the nodes, which is an advantage over the LCN approach. An example of an LCPN approach is the Hierarchical Deep Learning for Text Classification (HDLTex) model [18] . This model consists of a combination of neural networks at each parent node in the classification hierarchy, creating a stacked deep learning architecture. Using this structure, the HDLTex model outperforms more traditional classifiers like NB and SVM.

The last and least common type is the Local Classifier per Level (LCL) approach, where a flat classifier is trained at each level of the hierarchy, e.g., [42]. Compared to the other local classification approaches, the LCL approach trains the fewest classifiers. However, at deeper levels of the hierarchy, this requires the classifiers to discriminate between a larger number of categories. Furthermore, inconsistencies can easily arise, when predicted leaf categories do not belong to the predicted internal node categories.

2.3. Class Imbalance

One issue that can arise in classification problems is class imbalance which occurs when the data are not equally distributed among the categories, resulting in some categories being more well-represented than others. In [43], three measures are introduced for determining the level of imbalance in a multi-label dataset. The first is the imbalance ratio per label (*IRLbl*, see (1), where y_i is the label set of observation i). For each class j, this measure divides the class frequency of the most common class by the class frequency of class j. The imbalance ratio will equal one for the largest class, and increase as the imbalance increases.

$$IRLbl(j) = \frac{\max_{j'=1}^{J}(\sum_{i=1}^{N} h(j', y_i))}{\sum_{i=1}^{N} h(j, y_i)}, \tag{1}$$

$$h(j, y_i) = \begin{cases} 1, j \in y_i, \\ 0, j \notin y_i. \end{cases}$$

The mean imbalance ratio (*MeanIR*) calculates the average imbalance ratio over all labels and represents the overall level of imbalance in the data (see (2)).

$$MeanIR = \frac{1}{J} \sum_{j=1}^{J} IRLbl(j). \tag{2}$$

Finally, the coefficient of variation of IRLbl (*CVIR*) is introduced, calculated as the standard deviation of the imbalance ratios per label, divided by the mean imbalance (see (3)). This indicator shows the level of variation between the various imbalance ratios.

$$CVIR = \frac{IRLbl_\sigma}{MeanIR}, \tag{3}$$

$$IRLbl_\sigma = \sqrt{\sum_{j=1}^{J} \frac{(IRLbl(j) - MeanIR)^2}{J-1}}.$$

One approach to handle class imbalance is through cost-sensitive learning, where the cost function is adapted such that misclassifying instances from the minority class(es) result in a higher cost than misclassifying instances from the majority class(es). This forces the classification algorithm to place more weight to correctly classifying minority class instances. An instance of a cost-sensitive learning approach in hierarchical text classification can be found in [44], in which a hierarchical SVM approach is taken, followed by cost-sensitive neural network classifiers on the output scores of the SVM model. The study finds that the hierarchical SVM approach outperforms the non-hierarchical SVM classifier. Furthermore, the cost-sensitive approach is successful in decreasing the number of severe misclassifications. In [16], a cost ratio C^+/C^- (where C^+ and C^- are the cost given to a majority and a minority class, respectively) set to the inverse imbalance ratio is found to yield good performance in SVM models.

Another way to deal with class imbalance is through the under- or oversampling of data. The former refers to the random removal of instances from the majority class(es), e.g., [45], whereas the

latter is the opposite: instances from the minority class(es) are randomly sampled with replacement, e.g., [46]. Instead of sampling with replacement, one can also create synthetic examples from the minority class, for instance using the Synthetic Minority Oversampling Technique (SMOTE) [47]. [48] propose the Multi-Label SMOTE (MLSMOTE) algorithm for generating synthetic observations for minority classes, suitable for multi-label data. This algorithm is adapted from SMOTE, but involves an additional step: the generation of a synthetic label set for the generated instance. Using the MLSMOTE algorithm, Charte *et al.* are able to achieve better results compared to other oversampling techniques and imbalance-aware classifiers. However, the use of resampling methods, whether synthetic or not, presents one drawback. For hierarchical (local) classification, a set of classifiers is used, each trained on a different subsample of the data. The algorithm would need to be applied to each subsample, which could be costly, especially as the number of classifiers increases.

3. Data

The data used in this paper originates from Research Papers in Economics (RePEc), a repository containing a large volume of working papers, journal articles, books, and other forms of economic publications. We extracted the title and abstract of the publications to use as input to our classifiers, restricting the sample to only the publications that have at least one JEL classification code and are written in English. The JEL classification codes belong to a system developed by the Journal of Economic Literature, and is used for the classification of economic publications. The classification system forms a hierarchy with three layers. It consists of twenty primary categories, labelled with letters ranging from A-R and Y-Z. The subcategories at the second level are the secondary categories. They are denoted by the letter corresponding to their parent category, followed by a single digit denoting the subcategory, starting from zero. An example of a more specific category is 'Asset Markets and Pricing', which is denoted by the JEL code G1. This is the second subcategory of category G, 'Financial Economics'. In the same way, the subcategories of the secondary categories, i.e., tertiary categories, are denoted by their parent category code plus an additional digit. For instance, category G11 'Portfolio Choice; Investment Decisions' is the second category within category G1. Figure 1 shows part of the JEL categorisation system: the subtree rooted at primary category G.

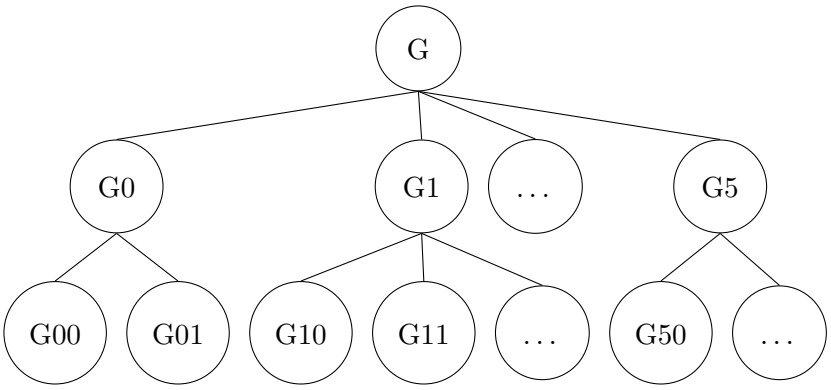

Figure 1: Primary category G and its descendant category nodes.

We exclude a small number of JEL classification codes, namely the category with code 'Y' ('Miscellaneous Categories'), which is used for unclassified documents and contain subcategories that are not topic-bound such as 'Excerpts' and 'No Author General Discussions'. We disregard this category and documents belonging exclusively to this category. For the same reason, we exclude the category with code 'A3' ('Collective Works'), and its corresponding subcategories.

In total, there are 19 primary categories. Each primary category has between three and ten subcategories, with a median of six. Furthermore, there are 129 secondary categories and 834 tertiary categories. The secondary categories have between two and ten tertiary categories as children, with a median of seven.

Our dataset contains 428,827 observations and their tertiary category labels. Within this dataset, approximately 20% of the data (83,445 observations) is reserved as the test dataset for reporting the final results of our models. The dataset is split using an iterative stratification approach developed for multi-label data proposed in [49]. This approach is aimed at maintaining, within each subset of the data, the proportion of positive examples for each category in the complete dataset. The remaining 80% of the data is then split once more using the same procedure, resulting in a training dataset (278,510 observations) and a validation dataset (66,872 observations), which are used for hyperparameter optimization. Table 1 shows the frequency table of the number of (tertiary) labels per observation for each split dataset as well as the complete dataset. Observations with three labels are the most common, representing almost one third of each dataset, followed by observations with two labels. Less than 5% of the data instances have more than five labels.

Table 2 displays descriptive statistics regarding the size of the JEL categories. For the primary and secondary JEL categories, we define the size of a category as the number of instances belonging

Table 1: Frequency table of the number of labels per observation in percentages.

No. of labels	1	2	3	4	5	6+
Train	21.74	22.84	29.59	15.04	6.36	4.44
Validation	12.73	27.12	32.75	16.23	6.54	4.63
Test	9.14	29.86	33.66	15.97	6.72	4.65
All	17.88	24.87	30.87	15.40	6.46	4.51

to the subtree rooted at that category. The table shows that some imbalance already occurs in the primary JEL categories, with the largest category being more than ten times the size of the smallest category. The mean imbalance ratio lies around 4.5, indicating that the majority class is on average 4.5 times larger than the other classes. Moving from the primary categories to the secondary categories show the same pattern as moving from secondary categories to tertiary categories. The average size per label decreases. The smallest group and the largest group both shrink. This is due to the observations being spread out to more categories. This increase in spread affects the imbalance ratios. The mean imbalance ratios increase to about 25 for the secondary categories, and then increases to over 70 for the tertiary categories. This suggests that a secondary/tertiary category is on average 25/70 times smaller than the largest category respectively. The imbalance ratios vary more, as evidenced by the larger CVIR values.

The text data is cleaned as follows. First, we remove the HTML tags, the JEL codes, and the numbers from the text. Then, we remove some specific phrases like “no abstract, this is a discussion paper”, “abstract in English is missing”, and “abstract missing - contribution appeared in the programme”. Last, non-English text is also removed.

Table 2: Descriptive statistics on the number of observations per JEL category.

	Primary			
	Train	Validation	Test	All
Mean	27416.74	6918.32	8785.47	43120.53
Min	4907	1217	1562	7686
Max	63742	16157	20547	100446
MeanIR	4.48	4.51	4.49	4.49
CVIR	4.02	4.07	4.02	4.03
	Secondary			
	Train	Validation	Test	All
Mean	5108.33	1297.45	1648.14	8053.92
Min	28	6	11	45
Max	23961	5996	7462	37419
MeanIR	25.84	26.42	23.08	25.26
CVIR	82.77	93.31	66.76	80.17
	Tertiary			
	Train	Validation	Test	All
Mean	933.14	236.93	300.46	1470.53
Min	7	1	1	12
Max	10402	2601	3251	16254
MeanIR	71.3	75.17	74.43	70.62
CVIR	138.26	173.69	210.92	140.3

4. Methodology

In this section, we describe the methods that we use to classify publications and evaluate said classifications. We start by describing our use of SVMs and our extension of the HDLTex model, HDLTex++ (including its constituent neural networks), before considering the feature extraction approaches we utilize, the loss functions that we use for each classification model, as well as our approach to model evaluation. We then describe the hyperparameter optimization involved in each

classification approach as well as the hardware and software used to perform our analysis.

4.1. Support Vector Machines (SVMs)

In a binary-class SVM, introduced in [29], the objective function is formulated as

$$\begin{aligned} &\min_{\boldsymbol{v},b} \frac{1}{2} \|\boldsymbol{v}\|^2 + C \sum_i \xi_i, \\ &\text{s.t. } y_i[\boldsymbol{v}'\boldsymbol{x}_i + b] \geq 1 - \xi_i, \quad \forall i \\ &\xi_i \geq 0, \quad \forall i \end{aligned} \tag{4}$$

where $\boldsymbol{v}$ is the weight vector, b the bias term, ξ_i the slack variables, C the regularization parameter, $\boldsymbol{x}_i$ the input vector, and $\boldsymbol{y}_i \in \{-1, 1\}$ the class label. Using the Lagrangian primal function and its first order conditions, the optimization problem is given by

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j), \\ \text{s.t.} \quad & \sum_i \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C, \quad \forall i, \end{aligned} \tag{5}$$

where $K(\cdot, \cdot)$ is the kernel function. Solving the optimization problem yields the decision function

$$f(\boldsymbol{x}) = \boldsymbol{v}'\boldsymbol{x} + b = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b. \tag{6}$$

In this paper, we use the linear kernel, where $K(\boldsymbol{x_i}, \boldsymbol{x_j})$ is simply the inner product of the two vectors. SVMs with linear kernels are faster to train, yet have been shown to achieve similar, if not better, performance in text classification than SVMs with non-linear kernels [35, 41, 28].

To adapt the binary-class SVM for multi-label classification with J classes, we take the OVR approach, and train a binary-class SVM for each class, yielding J decision functions $f_{c_1}(\boldsymbol{x}), \ldots, f_{c_J}(\boldsymbol{x})$. We pass the decision function values through the sigmoid function such that the output range is bounded between zero and one. The classes for which the output is above a certain threshold are assigned to the instance. Both the regularization parameter and the threshold values are selected using cross-validation. Training a binary-class SVM for each class against $J - 1$ other classes can introduce class imbalance to the classification problem, since the number of instances with the class label tends to be much lower than the number of instances without. This can result in a skewed class boundary, such that new instances are more likely not to be classified to the class [50].

To account for the class imbalance, we use separate cost parameters for the positive and negative examples of each SVM submodel by multiplying the cost C by a class-dependent weight. The class weights are inversely proportional to the class frequency, with classes containing few positive examples receiving a larger weight.

We train a flat and a hierarchical SVM classifier. The flat classifier ignores the hierarchical structure of the classification system and learns an OVR classifier for each of the leaf categories of the class hierarchy, using the entire dataset for each classifier. For the hierarchical SVM, we take the LCN approach, as has been tried with good results in previous literature [41], and train an OVR classifier at each node of the class hierarchy. All instances belonging to descendant categories of the category node are considered positive examples, whereas all instances belonging to the categories descending from the node's *sibling* categories are used as negative examples.

4.2. HDLTex++

The proposed HDLTex++ model, extended from the HDLTex model [18], uses an LCPN approach to text classification. At each parent node of the class hierarchy, a deep learning architecture is used, either a Convolutional Neural Network (CNN) or a Recurrent Neural Network (RNN). In other words, the highest-level classifier at the root of the classification hierarchy uses the entire training dataset to train, since all categories are descendants of the root, whereas subsequent classifiers only use a subset of the data, using only those training instances assigned labels from categories belonging to the subtree rooted at the classifier node. By training just one model for each set of sibling categories, interdependencies between these categories can be taken into account, an advantage over LCN approaches. Furthermore, inconsistencies (where a document is categorised into a child category but not the corresponding parent category) that might occur in LCL approaches are avoided.

Whereas [18] consider all combinations of three models in both layers of their architecture, specifically the basic fully connected (i.e., dense), feedforward neural networks they call Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs), thereby training nine (3^2) models in total, we only use the latter two, since these architectures present advantages over the basic DNNs. For instance, CNNs are able to extract local features from the input data, while RNNs are useful for handling sequential data. Furthermore, both models contain a fully connected output layer, essentially still incorporating the key feature

of DNNs. By leaving out the DNNs, we train a total of eight (2^3) models (i.e., three levels, with either CNNs or RNNs at each level). In the following subsections, each type of neural network is described, both CNNs and RNNs, as well as the hyperparameters present in each.

4.2.1. Convolutional Neural Network (CNN)

CNNs make use of convolution layers that connect to a subset of the input. Originally developed for visual pattern recognition, CNNs are used to pick up local features within the larger input, such as visual objects present at varying locations in different images [51], but have also been used in natural language processing [52] and text classification [34]. The main idea is to pass over the input data using a sliding window approach and apply a non-linear transformation (also called the convolution filter or kernel) that returns a scalar value for each window. The result is a convolution layer or feature map containing local features. In CNNs using text input, temporal (i.e., one-dimensional) convolutions can be used as follows. Let $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{n_{in}})$ be a $T \times n_{in}$ input matrix, for example a document consisting of n_{in} word embeddings of size T. Moving a sliding window with width k over the data results in $m = n_{in} - k + 1$ windows of size $T \times k$. Applying a filter $\boldsymbol{U}$ typically consists of performing element-wise matrix multiplication of the window and a weight matrix, adding a bias to each resulting element, taking the sum the results, and passing the sum through an activation function such as the sigmoid and ReLU functions. The elements of the resulting convolution layer $\boldsymbol{z} = (z_1, \ldots z_m)$ are given by

$$z_i = g(\tilde{\boldsymbol{X}}_i \odot \boldsymbol{W} + \boldsymbol{B}), \quad \forall i = \{1, \ldots, m\}, \tag{7}$$

where $\tilde{\boldsymbol{X}}_i = (\boldsymbol{x}_i, \ldots, \boldsymbol{x}_{i+k-1})$ is the i^{th} window, $\boldsymbol{W}$ and $\boldsymbol{B}$ are the weight and the bias matrices of the filter, respectively, $\odot$ specifies the element-wise multiplication and $g(\cdot)$ is a non-linear function (e.g., the sum of all elements followed by an activation function).

When multiple filters $\boldsymbol{U}_1, \ldots, \boldsymbol{U}_s$ are applied on the input data, a set of s convolution layers with size $m \times 1$ is created. In the same way, a new set of feature maps can be created using the output from the previous set. Before passing the output from one set of layers to the next, the dimension is reduced using pooling layers, with the intent to extract the most important feature from the feature maps. A common pooling technique that we apply is max pooling. Using max pooling with a pooling size p, a sliding approach is used once more to pass over the elements *within* each convolution layer, this time using a window of width p. For each window, the maximum element is

selected. The final layers of a CNN are typically fully connected and use the final flattened pooled layer as input. Figure 2 shows a diagram containing two convolution filters with width 2 and 3, followed by a pooling layer with filter width 3. The result of the pooling layer is then flattened into a single vector.

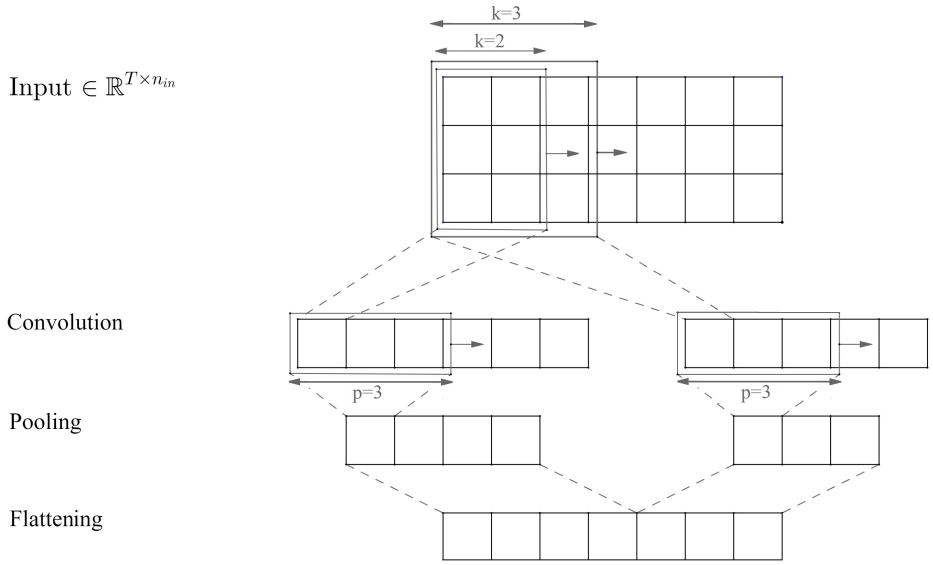

Figure 2: Example of 1D convolution and pooling layers.

4.2.2. Recurrent Neural Network (RNN)

In RNNs, the output from a layer in the network can be reused (with a delay) as the input to the same layer, giving RNNs the ability to handle sequential data. RNNs are characterized by the following recursive operation:

$$\boldsymbol{h}_t = \boldsymbol{W}_{rec}\sigma(\boldsymbol{h}_{t-1}) + \boldsymbol{W}_{in}\boldsymbol{x}_t + \boldsymbol{b}, \tag{8}$$

where $\boldsymbol{h}_t$ and $\boldsymbol{x}_t$ are the hidden state and input vectors at time t, respectively, $\boldsymbol{W}_{rec}$ and $\boldsymbol{W}_{in}$ are the recurrent and input weight matrices, respectively, $\boldsymbol{b}$ is a bias vector, and $\sigma(\cdot)$ is an element-wise non-linear function (for which we use the sigmoid function).

The conventional back-propagation through time method of fitting an RNN can lead to exploding or vanishing gradients [53]. To solve this problem, the long short-term memory (LSTM) model was proposed, which uses gating functions for preserving long-term dependencies [54]. These gates help regulate the flow of information by controlling when information is passed along or forgotten.

A simplified variant of the LSTM model that is less computationally expensive but has comparable performance is the gated recurrent unit (GRU) [55]. Each GRU cell contains two gates, the reset gate and the update gate, which at time t are calculated as

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_r[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_r), \tag{9}$$

$$\boldsymbol{u}_t = \sigma(\boldsymbol{W}_u[\boldsymbol{x}_t, \boldsymbol{h}_{t-1}] + \boldsymbol{b}_u), \tag{10}$$

where $\boldsymbol{b_r}$ and $\boldsymbol{b_u}$ are bias vectors, $\boldsymbol{W}_r$ and $\boldsymbol{W}_u$ are a weight matrices, and $\sigma(\cdot)$ is an element-wise sigmoid function. The hidden units $\boldsymbol{h}_t$ are then calculated as

$$\boldsymbol{h}_t = \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \odot \tilde{\boldsymbol{h}}_{\boldsymbol{t}}, \tag{11}$$

where $\tilde{\boldsymbol{h}}_t$ is the candidate value given by

$$\tilde{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{\tilde{h}}[\boldsymbol{x}_t, \boldsymbol{r} \odot \boldsymbol{h}_{t-1}] + \boldsymbol{b}_{\tilde{h}}). \tag{12}$$

The reset gate, each element of which can take on a value between zero and one, thus determines to what extent the previous hidden state is used to calculate the candidate cell values. The update gate determines to what extent the previous hidden state values are retained and, at the same time, what proportion of the candidate values is used to calculate the new cell values.

4.3. Feature Extraction

To extract the relevant features from each document, we use word embeddings through (1) GloVe, which provides context-independent word representations, and (2) BERT, for which the word vectors are context-dependent, both of which are among the best performing word embeddings in their category. Both GloVe and BERT provide pre-trained models. For the baseline SVM models, we use the pre-trained GloVe embeddings with length 100, as well as the word embeddings derived from the BERT-Base model, which provides embeddings with length 768. Before converting the text to word vectors, all words are converted to lower case. Furthermore, before vectorising words using GloVe embeddings, we remove punctuation marks from the text, since they by themselves do not carry semantic meaning. However, for the BERT word embeddings, punctuation marks are preserved, since they can provide contextual meaning. Using GloVe and BERT word embedding techniques, we obtain a word vector for each word in the document. A document $\boldsymbol{X}_i$ can be represented as

$$\boldsymbol{X}_i = (\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,n_i}) \in \mathbb{R}^{T \times n_i}, \tag{13}$$

where T is the dimension of each word vector $\boldsymbol{x}_{i,m}$, and n_i is the number of words in document $\boldsymbol{X}_i$. We aggregate the word vectors using the coordinate-wise average or maximum to obtain the input vectors for the SVMs. This results in vectors $\boldsymbol{x}_i^{(avg)}$ and $\boldsymbol{x}_i^{(max)}$, such that

$$x_{i,j}^{(avg)} = \frac{\sum_m x_{i,m,j}}{n_i}, \tag{14}$$

$$x_{i,j}^{(max)} = \max_m \ x_{i,m,j}. \tag{15}$$

For HDLTex++, we use only pre-trained GloVe embeddings as input. Since the pre-trained GloVe embeddings are context-independent, there are only a fixed number of GloVe embeddings, i.e., the size of the vocabulary on which it was trained. These embeddings form a matrix that can be used as a layer in the deep learning models.

4.4. Loss Functions

In SVMs, the loss function is defined as the average hinge loss over the observations, where the hinge loss for any observation i is given by

$$l^{hinge}(\boldsymbol{x}_i; \boldsymbol{v}, b) = \max\{0, 1 - y_i(\boldsymbol{v}'\boldsymbol{x}_i + b)\}. \tag{16}$$

To account for class imbalance, we use the weighted average to calculate the loss function. The loss function for an SVM submodel for category c is given by

$$\mathcal{J}_c^{hinge}(\boldsymbol{v}_c, b_c) = \frac{1}{\sum_i w_{i,c}} \sum_i w_{i,c} l_c^{hinge}(\boldsymbol{x}_i; \boldsymbol{v}_c, b_c), \tag{17}$$

where $w_{i,c}$ is a class-dependent weight defined as the inverse class frequency divided by the number of classes. In other words, $w_{i,c}$ is defined as

$$w_{i,c} = \begin{cases} \frac{N_c}{J_c * N_c^+}, & \text{if } y_{i,c} = 1, \\ \frac{N_c}{J_c * N_c^-} & \text{otherwise,} \end{cases} \tag{18}$$

where N_c is the total number of training examples, N_c^+ and N_c^- are the number of positive and negative examples, respectively, and J_c refers to the number of classes.

In [18], HDLTex was applied on a multi-class single-label text classification problem with the use of a categorical cross-entropy loss function. However, we train the model to classify multi-class multi-label data, for which this loss function is unsuitable. Hence, we make use of the binary cross-entropy (BCE) loss function, and use a sigmoid activation function for the output layer, similar to

the approach taken by [33] and [34], such that each output node returns a value between zero and one.

We also add weights to the function to correct for class imbalance, using the same calculation as the weights in the SVM models, such that misclassification of observations from classes with fewer instances are penalized more heavily. The weighted binary cross-entropy (WBCE) loss function at each parent node p then looks as follows:

$$\begin{aligned}\mathcal{J}_p^{WBCE}(\boldsymbol{\theta}) &= \sum_i \sum_{c \in child(p)} \mathcal{J}_c^{WBCE}(\boldsymbol{\theta}) \\ &= -\sum_i \sum_{c \in child(p)} [w_{i,c}(y_{i,c} \log(\hat{y}_{i,c}(\boldsymbol{\theta})) \\ &\quad + (1 - y_{i,c}) \log(1 - \hat{y}_{i,c}(\boldsymbol{\theta})))], \end{aligned} \tag{19}$$

where $y_{i,c}$ is an indicator for whether instance i belongs to category c, $\hat{y}_{i,c}(\cdot)$ is the predicted probability of instance i belonging to category c, and $\boldsymbol{\theta}$ are the parameters.

4.5. Evaluation Measures

To evaluate the performance of the models, we use two performance measures. The first is the macro-averaged "flat" F1-score, calculated as the arithmetic mean of the F1-scores of each category, where the F1-score is the harmonic mean of the precision and recall. This F1-score thus gives equal weight to each category, regardless of category size. However, it does not take into account the class hierarchy. Each misclassification is weighted equally, even though misclassification at a lower level in the hierarchy might be considered less severe than a misclassification at the top level. Therefore, we propose a second evaluation metric, the macro-averaged hierarchical F1-score, based on the hierarchical performance measures proposed in [40]. The macro-averaged hierarchical F1-score not only incorporates the predicted and target label sets, but also their respective ancestor category label sets. Let ℓ_i and $\hat{\ell}_i$ denote the target and predicted label sets of instance i, respectively. The extended label sets, shown in (20) and (21), append to each label set the ancestor categories of each leaf category (excluding the root $\mathcal{R}$).

$$Anc_{\ell_i} = \{\cup_{c_k \in \ell_i} Ancestors(c_k) \setminus \mathcal{R}\}, \tag{20}$$

$$Anc_{\hat{\ell}_i} = \{\cup_{c_k \in \hat{\ell}_i} Ancestors(c_k) \setminus \mathcal{R}\}. \tag{21}$$

Using this notation, we can define the hierarchical precision, recall, and F1-score per category as

$$hP_{c_k} = \frac{\sum_{i:c_k \in \ell_i} \left| Anc_{\ell_i} \cap Anc_{\hat{\ell}_i} \right|}{\sum_{i:c_k \in \ell_i} \left| Anc_{\hat{\ell}_i} \right|}, \tag{22}$$

$$hR_{c_k} = \frac{\sum_{i:c_k \in \ell_i} \left| Anc_{\ell_i} \cap Anc_{\hat{\ell}_i} \right|}{\sum_{i:c_k \in \ell_i} |Anc_{\ell_i}|}, \tag{23}$$

$$hF1_{c_k} = \frac{2hP_{c_k} hR_{c_k}}{hP_{c_k} + hR_{c_k}}. \tag{24}$$

The macro-averaged hierarchical F1-score is calculated, similar to the macro-averaged flat F1-score, as the arithmetic mean of each category's hierarchical F1-score.

Using hierarchical F1-score instead of the flat F1-score has the advantage of penalizing less the mistakes made for nodes in the same branch of the classification tree. With respect to Figure 1, for example, misclassifying a document under the label G11 instead of using the correct label G10 incurs a smaller penalty than misclassifying under the label G50. The reason is that the node labeled G50 is in a different branch than the nodes labeled G10 and G11 in the classification tree. In the considered JEL classification hierarchy, misclassifying for nodes that are in the same branch with the correct node is less often an issue as the global context of the document is still accurately captured. This ensures the usability and the reliability of the JEL taxonomy.

4.6. Hyperparameter Optimization

To find the optimal hyperparameters of each model, we use the holdout method and select the hyperparameters that maximize performance on the validation set. However, the relative performance depends on the performance measures used. Hyperparameters that maximize the flat F1-score focus only on the correct prediction of samples into the tertiary categories, without minimizing the severity of misclassifications. On the other hand, hyperparameters maximizing the hierarchical F1-score focus on minimizing the severity of misclassifications, possibly at the cost of fewer predicted paths that are completely correct. In other words, there is a trade-off between maximizing correct predictions and minimizing the severity of misclassifications. Since we are ultimately interested in the correct prediction of economic publications to the tertiary categories, we optimize the hyperparameters with respect to the flat F1-score. The performance of the final model is evaluated on the test set.

For the SVM models, we optimise two hyperparameters: the regularisation parameter C and the threshold value(s). The regularisation parameter determines the strength of regularisation. A large C penalises misclassifications more heavily, resulting in a hyperplane with a smaller margin which will correctly classify more training examples, but may also overfit. A small C allows for a higher number of misclassified training examples. The threshold value acts as a cut-off point for the output of the SVM, i.e., the decision function values after being passed through the sigmoid function. By adjusting the threshold value, making positive predictions (i.e., classifying an instance to a particular category) can be made easier (when the threshold value is decreased) or harder (when the threshold value is increased). For the flat SVM model, we use a global threshold value that is used for all classifiers. On the other hand, the hierarchical SVM model contains three threshold values, one for each level, and find the combination of threshold values that yields the best performance.

Table 3: Hyperparameters used in the SVM and HDLTex++ models.

	Hyperparameter	Meaning	Value
Flat SVM	Kernel	Kernel function	Linear
	C	Regularisation parameter	*To be optimised*
	Threshold value	Threshold value for the SVM output	*To be optimised*
Hierarchical SVM	Kernel	Kernel function.	Linear
	C	Regularisation parameter	*To be optimised*
	Threshold value (level 1)	Threshold value for the primary level SVM output	*To be optimised*
	Threshold value (level 2)	Threshold value for the secondary level SVM output	*To be optimised*
	Threshold value (level 3)	Threshold value for the tertiary level SVM ouput	*To be optimised*
HDLTex++	Max sequence length	Maximum number of word embeddings per sample.	500
	Batch size	Number of samples per computation	128
	Epochs	Number of epochs	1
	Optimiser	Algorithm for optimising the model	Adam
	Learning rate	Learning rate of the optimiser	0.001
	Decay rates	Decay rates of the optimiser	{0.9, 0.999}
	Dropout rate	Fraction of nodes to drop	0.25
	GRU units	Number of GRU units	100
	Convolution filter width (first set)	Window width of the first set of convolution layers, connected to the input layer	{3,4,5,6,7}
	Convolution filters (first set)	Number of convolution filters per layer, for the first set	{128, 128, 128, 128, 128}
	Max pooling filter width (first set)	Window width of the first set of pooling layers, following each convolution layer in the first set	{5, 5, 5, 5, 5}
	Convolution filter width (second set)	Window width of the second set of convolution layers, connected to the concatenated result of the first set of convolution and pooling layers	{5, 5, 5}
	Convolution filters (second set)	Number of convolution filters per layer, for the second set	{128, 128, 128}
	Max pooling filter width (second set)	Window width of the second set of pooling layers, following each convolution layer in the second set	{5, 5, 30}
	Threshold value (level 1)	Threshold value for the primary level output	*To be optimised*
	Threshold value (level 2)	Threshold value for the secondary level output	*To be optimised*
	Threshold value (level 3)	Threshold value for the tertiary level output	*To be optimised*

For the HDLTex++ models, there are a great number of hyperparameters than can be varied, such as the batch size, learning rate, and the number and sizes of the convolution filters. To limit computations, however, we fix these hyperparameters to the values used in the original paper, and optimise only C and the threshold values at each level of the models. For C we have considered values in the set $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$ and for the thresholds we have considered values between 0 and 1 with a step of 0.01 for the SVM models and 0.001 for the HDLTex++ models. For the hyperparameter optimization we have used a grid search approach. Table 3 lists the hyperparameters used per classifier.

4.7. Hardware and Implementation

Training for the flat and hierarchical SVM models was done using an *Intel® Core™ i7-8565U* CPU (1.8 GHz) with 4 cores and 8 GB RAM. The HDLTex++ models were trained using an *NVIDIA Tesla T4* GPU provided by Google Colaboratory. The models were implemented in Python, with help from the Sci-Kit Learn, TensorFlow, and Keras libraries. Our code and data are made available at: `https://github.com/SanneLin/HDLTex_plus_plus`.

5. Results

In this section, we present the results of the various classification methods discussed in Section 4. We begin with a discussion of each approach's efficiency before continuing with the results of the SVM models and HDLTex++ models.

5.1. Efficiency

Table 4 shows the training times of each level of the SVM and HDLTex++ models. For each classifier, the left column ("Total") reports the average total time to train all subclassifiers within one hierarchical level. The right column ("Average") reports the average total time divided by the number of subclassifiers within each level of the hierarchy. In terms of both total and average training time, the hierachical SVM models rank first. Nevertheless, the average training time per HDLTex++ subclassifier decreases significantly down the hierarchy, which may be attributed to the subclassifiers using a subset of data to train, instead of the entire dataset.

Table 4: Training times for all models in seconds.

	Flat SVM		Hierarchical SVM		HDLTex++	
Level	Total	Average	Total	Average	Total	Average
1	85	4.8	77	4.1	6180	6180
2	683	5.3	44	<1	11940	628.4
3	5337	6.4	50	<1	4770	37.0

5.2. Baseline SVM

Table 5 shows the results of the baseline SVM models as well as the optimal parameters of each SVM approach found through the holdout method for each type of word embedding. Note that the flat SVM approach treats each level in the class hierarchy separately, such that the models at each level are trained independently from one another, on the entire training dataset. On the other hand, the hierarchical SVM models incorporate the class hierarchy. While the submodels are still trained independently using subsets of data belonging to the same parent category, predictions at the lower levels depend on the predictions made by the top level classifiers. For the primary level models, the flat and hierarchical F1-scores are equal, as there is no hierarchy to take into account. For the secondary and tertiary level models, the hierarchical F1-score always appears to be higher than the flat F1-score.

Table 5: Optimal parameters and results of the baseline SVM models.

		Flat SVM				Hierarchical SVM			
Embedding	Level	C	Threshold	Flat F1	Hier. F1	C	Threshold	Flat F1	Hier. F1
GloVe (Average)	1	1	0.57	0.470	0.470	10^{-1}	0.50	0.425	0.425
	2	10^{-1}	0.67	0.252	0.366		0.50	0.210	0.329
	3	10^{-4}	0.75	0.112	0.233		0.56	0.109	0.245
GloVe (Max)	1	10^{-2}	0.55	0.339	0.339	10^{-2}	0.50	0.313	0.313
	2	10^{-3}	0.62	0.159	0.261		0.50	0.122	0.217
	3	10^{-3}	0.69	0.066	0.166		0.56	0.053	0.169
BERT (Average)	1	1	0.60	0.504	0.504	10^{-2}	0.50	0.448	0.448
	2	10^{-4}	0.67	0.277	0.400		0.50	0.226	0.350
	3	10^{-4}	0.75	0.130	0.266		0.62	0.119	0.262
BERT (Max)	1	10^{-2}	0.60	0.453	0.453	10^{-2}	0.50	0.400	0.400
	2	10^{-4}	0.67	0.238	0.355		0.50	0.184	0.301
	3	10^{-4}	0.75	0.103	0.227		0.62	0.086	0.214

We see that the optimal regularisation parameter for the flat SVM models is largest for the primary level models (i.e., 1 for the models using averaged embeddings, and 10^{-2} for the models using max-aggregated models), and smaller for the secondary and tertiary level models, indicating that the primary level models are the least regularised. In contrast, the optimal threshold level

is lowest for the primary level models and largest for the tertiary level models. Additionally, for hierarchical SVM models, each model has an optimal threshold for the first two levels of 0.5, while the optimal decision threshold for the third level is slightly higher at 0.62 for models using BERT embeddings and 0.56 for models using GloVe.

Comparing model performance across different levels, we find that both F1-scores are highest when predicting the primary level category labels, with performance decreasing with each subsequent level. Overall, the models using averaged BERT embeddings perform best. The results also indicate that the BERT models using the averaged embeddings and the max-aggregated embeddings both outperform their GloVe counterparts. Furthermore, models using averaged embeddings perform better than using max-aggregated embeddings.

5.3. HDLTex++

Table 6 shows the results of the eight HDLTex++ models, where each level of the model consists of either a CNN architecture or a RNN architecture. For each model, the optimal threshold levels are shown, which indicate the minimum probability an observation needs to have of belonging to a category before the model will classify the observation to that category. For each model, we find that the optimal threshold value is highest for the first level and decreases for each subsequent level (except for model 7, where the primary and secondary levels have the same threshold value). This shows that with each subsequent level, it becomes easier to classify an observation to a class.

At the primary level, the models using CNNs perform slightly better, achieving an F1-score of 54.3%, whereas the models using RNNs reach an F1-score of 52.5%. Furthermore, models using at least one RNN architecture at the primary and secondary level perform worse than the models with two CNN architectures, in terms of both flat and hierarchical F1-score. When looking at the tertiary level performance, model 2 performs best in terms of flat F1-score, while model 1 performs best in terms of hierarchical F1-score, though the differences are small (less than 1%). Excluding the tertiary level, models 1 and 2 outperform all SVM models except the flat SVM model using averaged BERT embeddings, in terms of both F1-scores.

Table 6: Results of the HDLTex++ models.

				F1	
Model	Level	Architecture	Threshold	Flat	Hierarchical
1	1	CNN	0.750	0.543	0.543
	2	CNN	0.675	0.254	0.407
	3	CNN	0.525	0.065	0.254
2	1	CNN	0.750	0.543	0.543
	2	CNN	0.650	0.259	0.412
	3	RNN	0.550	0.067	0.253
3	1	CNN	0.750	0.543	0.543
	2	RNN	0.700	0.223	0.376
	3	CNN	0.525	0.057	0.236
4	1	CNN	0.750	0.543	0.543
	2	RNN	0.675	0.232	0.386
	3	RNN	0.550	0.061	0.236
5	1	RNN	0.750	0.525	0.525
	2	CNN	0.675	0.245	0.397
	3	CNN	0.525	0.063	0.248
6	1	RNN	0.750	0.525	0.525
	2	CNN	0.650	0.251	0.402
	3	RNN	0.550	0.065	0.249
7	1	RNN	0.725	0.524	0.524
	2	RNN	0.725	0.208	0.360
	3	CNN	0.525	0.053	0.227
8	1	RNN	0.725	0.524	0.524
	2	RNN	0.700	0.217	0.371
	3	RNN	0.550	0.057	0.229

Including the tertiary level models, the HDLTex++ models perform worse in terms of flat F1-score, except compared to the GloVe (Max) SVM models. However, comparing the hierarchical F1-scores, differences between the SVM models and the HDLTex++ models 1 and 2 are much

smaller, and only the flat and hierarchical SVM models using averaged BERT embeddings perform better.

All in all, while the HDLTex++ models tend to produce better predictions for the primary and secondary levels, they do not perform as well in the tertiary level, which is why the flat performance measures of the tertiary level are quite low compared to the flat and hierarchical SVM models. Nevertheless, since the primary and secondary level submodels perform relatively well and each classifier within the tertiary level submodel can only classify observations to child categories of one class, observations that are misclassified tend to end up in sibling categories of the correct categories. Since sibling categories share the same path in the hierarchical tree structure, this can explain why the hierarchical performance remains comparatively high.

5.4. Comparison to Models Using Unweighted Loss

We also take a look at the performance of the flat SVM, hierarchical SVM, and HDLTex++ models, trained using unweighted loss functions. These models, similar to the weighted models, use the hyperparameters that maximize the flat F1-score.

5.4.1. Baseline SVM

Examining the optimal parameters of the unweighted flat and hierarchical SVM models, we see from Table 7 that, compared to the weighted models, the unweighted secondary and tertiary level models of the flat SVM as well as the hierarchical SVM on the whole consistently show higher optimal values for the regularisation parameter, indicating a lower level of regularisation. Furthermore, the optimal threshold value is lower across the board, making it easier to classify instances to categories.

Table 7 further displays the results of the unweighted SVM models. The flat SVM models using averaged BERT embeddings perform best in terms of both the flat and the hierarchical F1-score. On the other hand, the models using averaged GloVe embeddings perform poorest out of all unweighted flat SVM models, even though the weighted counterparts performed second-best. In terms of flat F1-score, each unweighted model performs worse than its weighted counterpart, whereas, in terms of hierarchical F1-score, the unweighted secondary and tertiary models using BERT embeddings perform slightly better.

For the hierarchical SVM models, we find that most models except those using max-aggregated GloVe embeddings show an improvement in both flat and hierarchical F1-score compared to the

weighted models. As this improvement occurs in all three levels, it is possible it stems from increased predictive performance of the primary and secondary levels, which are not as heavily impacted by class imbalance as the tertiary level categories.

Table 7: Optimal parameters and results of the unweighted SVM models.

		Flat SVM				Hierarchical SVM			
Embedding	Level	C	Threshold	Flat F1	Hier. F1	C	Threshold	Flat F1	Hier. F1
GloVe (Average)	1	10^{-4}	0.29	0.168	0.168	10^{1}	0.38	**0.432**	**0.432**
	2	10^{2}	0.27	0.042	0.086		0.38	**0.217**	**0.374**
	3	10^{-1}	0.29	0.025	0.039		0.44	0.099	**0.287**
GloVe (Max)	1	10^{-4}	0.29	0.211	0.211	10^{3}	0.32	0.133	0.133
	2	10^{-1}	0.29	0.120	0.234		0.32	0.031	0.117
	3	10^{1}	0.29	0.037	0.141		0.32	0.006	0.045
BERT (Average)	1	1	0.38	0.498	0.498	1	0.38	**0.497**	**0.497**
	2	1	0.33	0.261	**0.429**		0.38	**0.257**	**0.413**
	3	10^{1}	0.31	0.124	**0.280**		0.44	**0.123**	**0.322**
BERT (Max)	1	10^{-1}	0.38	0.441	0.441	10^{-1}	0.38	**0.440**	**0.440**
	2	10^{1}	0.31	0.204	**0.359**		0.38	**0.207**	**0.366**
	3	10^{2}	0.29	0.086	**0.251**		0.44	**0.086**	**0.276**

Values that are larger compared to the weighted models are emphasized in bold.

5.4.2. HDLTex++

Table 8 shows the results of the unweighted HDLTex++ models. We find once more that the optimal threshold levels are lower compared to the weighted models. Comparing the performance of the different HDLTex++ models, we see that, at each level, the models using CNN submodels perform better than the models using RNN submodels, such that the model using CNNs at every level performs best in terms of the flat and hierarchical F1-score. Moreover, the models using CNNs as the tertiary level submodels achieve higher flat F1-scores than their weighted counterpart models, which suggests that the cost-sensitive learning approach taken when training the weighted HDLTex++ does not result in noticeable improvement in the prediction of the minority classes. In terms of hierarchical F1-scores, the secondary and tertiary level models surpass not only the weighted HDLTex++ counterparts, but also the weighted SVM models.

Table 8: Results of the unweighted HDLTex++ models.

				F1	
Model	Level	Architecture	Threshold	Flat	Hierarchical
1	1	CNN	0.325	0.532	0.532
	2	CNN	0.350	**0.259**	**0.445**
	3	CNN	0.250	**0.072**	**0.324**
2	1	CNN	0.325	0.532	0.532
	2	CNN	0.350	0.259	**0.445**
	3	RNN	0.250	0.052	**0.309**
3	1	CNN	0.325	0.532	0.532
	2	RNN	0.325	0.223	**0.431**
	3	CNN	0.250	**0.066**	**0.315**
4	1	CNN	0.325	0.532	0.532
	2	RNN	0.325	0.223	**0.431**
	3	RNN	0.250	0.046	**0.300**
5	1	RNN	0.300	**0.528**	**0.528**
	2	CNN	0.350	**0.255**	**0.446**
	3	CNN	0.250	**0.070**	**0.324**
6	1	RNN	0.300	**0.528**	**0.528**
	2	CNN	0.350	**0.255**	**0.446**
	3	RNN	0.250	0.050	**0.311**
7	1	RNN	0.325	0.523	0.523
	2	RNN	0.325	**0.219**	**0.429**
	3	CNN	0.250	**0.064**	**0.312**
8	1	RNN	0.325	0.523	0.523
	2	RNN	0.325	**0.219**	**0.429**
	3	RNN	0.250	0.045	**0.299**

Values that are larger compared to the weighted models are emphasized in bold.

6. Conclusion

In an age where the body of academic literature grows ever larger, efficient and accurate classification of scientific publications becomes increasingly important. We explore different classifiers that can be used to categorize economic publications. These classifiers can be used to categorize also publications outside the economics domain, as they are domain agnostic. The classification problem has three main characteristics. First, the classification problem is multi-label: each publication can be classified to one or more categories. Second, the categories to which the publications can be assigned are part of a classification system that contains a tree structure. Third, not all categories contain an equal number of publications, causing a class imbalance problem that, if ignored, can cause classifiers to favour assigning publications to majority classes.

In this paper, we present the HDLTex++ approach, consisting of a neural network architecture at each of the three levels of the class hierarchy and extending the original HDLTex classifier by adapting the algorithm to be suitable for hierarchies consisting of three levels, rather than two. Furthermore, HDLTex++ uses the weighted binary cross-entropy loss function, to take into account both multi-label classification and class imbalance. We compare the HDLTex++ models to one flat and one hierarchical SVM classifier. Looking at the effectiveness of the models, we find that, while not as effective in the prediction of the tertiary category labels, the HDLTex++ models are more effective in predicting primary category labels, compared to the flat and hierarchical SVM models, and outperform the hierarchical SVM models in the prediction of secondary category labels. By exploiting the hierarchy when training the HDLTex++ models, it was possible to compensate for the inferior performance of the tertiary level submodels using the superior performance in the first two levels, allowing the HDLTex++ models to surpass six out of eight flat and hierarchical SVM models in terms of hierarchical performance. In terms of efficiency, the HDLTex++ models take more time to be trained, while the hierarchical SVM models can be trained the fastest.

Performance of the classifiers in this paper may be improved by training models using a wider range of hyperparameters and (deep learning) architectures, in order to learn which factors improve predictive capability. In the case of the SVM models, prediction may be improved using the Binarization with Boosting and Oversampling (BBO) framework, which is based on the OVR framework and applies boosting to classify hard-to-learn instances and oversampling to combat class imbalance [56]. Other solutions for class imbalance include (synthetic) under- and oversampling,

through methods such as MLSMOTE [48], and the use of focal loss [57], which may improve prediction of instances to minority classes. Also, we would like to fine-tune the BERT model for the economic domain, as we expect that this will further improve the performance of HDLTex++ for the JEL taxonomy classification. Last, we would like to experiment with hierarchical transformers for classification [58], which are particularly well-suited for large documents.

References

[1] A. Statnikov, C. F. Aliferis, I. Tsamardinos, D. Hardin, S. Levy, A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis, Bioinformatics 21 (5) (2005) 631–643.

[2] M. Pérez-Ortiz, S. Jiménez-Fernández, P. A. Gutiérrez, E. Alexandre, C. Hervás-Martínez, S. Salcedo-Sanz, A review of classification problems and algorithms in renewable energy applications, Energies 9 (8) (2016) 607.

[3] A. Shen, R. Tong, Y. Deng, Application of classification models on credit card fraud detection, in: Proceedings of the 2007 International Conference on Service Systems and Service Management (ICSSSM 2007), IEEE, 2007, pp. 1–4.

[4] P. C. Pendharkar, A threshold-varying artificial neural network approach for classification and its application to bankruptcy prediction problem, Computers & Operations Research 32 (10) (2005) 2561–2582.

[5] Y. M. Costa, L. S. Oliveira, A. L. Koericb, F. Gouyon, Music genre recognition using spectrograms, in: Proceedings of the 18th International Conference on Systems, Signals and Image Processing (IWSSIP 2011), IEEE, 2011, pp. 1–4.

[6] E. Blanzieri, A. Bryl, A survey of learning-based techniques of email spam filtering, Artificial Intelligence Review 29 (1) (2008) 63–92.

[7] H. Purohit, G. Dong, V. Shalin, K. Thirunarayan, A. Sheth, Intent classification of short-text on social media, in: Proceedings of the 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity 2015), IEEE, 2015, pp. 222–228.

[8] K. Mouthami, K. N. Devi, V. M. Bhaskaran, Sentiment analysis and classification based on textual reviews, in: Proceedings of the 2013 International Conference on Information Communication and Embedded Systems (ICICES 2013), IEEE, 2013, pp. 271–276.

[9] F. Sebastiani, Machine learning in automated text categorization, ACM Computing Surveys 34 (1) (2002) 1–47.

[10] P. Melville, W. Gryc, R. D. Lawrence, Sentiment analysis of blogs by combining lexical knowledge with text classification, in: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009), ACM, 2009, pp. 1275–1284.

[11] M. Aly, Survey on multiclass classification methods, Tech. rep., Caltech (2005).

[12] G. Tsoumakas, I. Katakis, Multi-label classification: An overview, International Journal of Data Warehousing and Mining 3 (3) (2007) 1–13.

[13] J. Duan, X. Yang, S. Gao, H. Yu, A partition-based problem transformation algorithm for classifying imbalanced multi-label data, Engineering Applications of Artificial Intelligence 128 (2024) 107506.

[14] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, N. Seliya, A survey on addressing high-class imbalance in big data, Journal of Big Data 5 (1) (2018) 1–30.

[15] Y. Qian, M. Aghaabbasi, M. Ali, M. Alqurashi, B. Salah, R. Zainol, M. Moeinaddini, E. E. Hussein, Classification of imbalanced travel mode choice to work data using adjustable SVM model, Applied Sciences 11 (24) (2021) 11916.

[16] R. Akbani, S. Kwek, N. Japkowicz, Applying support vector machines to imbalanced datasets, in: European Conference on Machine Learning (ECML 2004), Vol. 3201 of LNCS, Springer, 2004, pp. 39–50.

[17] Q. Li, H. Peng, J. Li, C. Xia, R. Yang, L. Sun, P. S. Yu, L. He, A survey on text classification: From traditional to deep learning, ACM Transactions on Intelligent Systems and Technology 13 (2) (2022).

[18] K. Kowsari, D. E. Brown, M. Heidarysafa, K. J. Meimandi, M. S. Gerber, L. E. Barnes, HDLTex: Hierarchical deep learning for text classification, in: Proceedings of the 16th IEEE International Conference on Machine Learning and Applications (ICMLA 2017), IEEE, 2017, pp. 364–371.

[19] L. Chen, L. Jiang, C. Li, Using modified term frequency to improve term weighting for text classification, Engineering Applications of Artificial Intelligence 101 (2021) 104215.

[20] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, D. Brown, Text classification algorithms: A survey, Information 10 (4) (2019) 150.

[21] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, in: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, Vol. 26, Curran Associates, Inc., 2013.

[22] J. Pennington, R. Socher, C. D. Manning, GloVe: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), ACL, 2014, pp. 1532–1543.

[23] C. De Boom, S. Van Canneyt, T. Demeester, B. Dhoedt, Representation learning for very short texts using weighted word embedding aggregation, Pattern Recognition Letters 80 (2016) 150–156.

[24] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, in: Proceedings of the 2018 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2018), ACL, 2018.

[25] O. Melamud, J. Goldberger, I. Dagan, context2vec: Learning generic context embedding with bidirectional LSTM, in: Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning (CoNLL 2016), ACL, 2016, pp. 51–61.

[26] B. McCann, J. Bradbury, C. Xiong, R. Socher, Learned in translation: Contextualized word vectors, in: Proceedings of the 31st Annual Conference on Neural Information Processing Systems (NIPS 2017), Curran Associates, Inc., 2017, pp. 6294–6305.

[27] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: Proceedings of the 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2019), ACL, 2019, pp. 4171–4186.

[28] W. Zhang, T. Yoshida, X. Tang, Text classification based on multi-word with support vector machine,

Knowledge-Based Systems 21 (8) (2008) 879–886.

[29] V. Vapnik, A. Y. Chervonenkis, A class of algorithms for pattern recognition learning, Avtomatika i Telemekhanika 25 (6) (1964) 937–945.

[30] T. Joachims, Text categorization with support vector machines: Learning with many relevant features, in: Proceedings of the 10th European Conference on Machine Learning (ECML 1998), Vol. 1398 of LNCS, Springer, 1998, pp. 137–142.

[31] J. Weston, C. Watkins, Multi-class support vector machines, Technical report CSD-TR-98-04, Royal Holloway University of London (1998).

[32] M.-L. Zhang, Z.-H. Zhou, Multilabel neural networks with applications to functional genomics and text categorization, IEEE Transactions on Knowledge and Data Engineering 18 (10) (2006) 1338–1351.

[33] J. Nam, J. Kim, E. L. Mencía, I. Gurevych, J. Fürnkranz, Large-scale multi-label text classification—revisiting neural networks, in: Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2014), Vol. 8725 of LNCS, Springer, 2014, pp. 437–452.

[34] J. Liu, W.-C. Chang, Y. Wu, Y. Yang, Deep learning for extreme multi-label text classification, in: Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2017), ACM, 2017, p. 115–124.

[35] Y. Yang, X. Liu, A re-examination of text categorization methods, in: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 1999), ACM, 1999, pp. 42–49.

[36] M.-L. Zhang, Z.-H. Zhou, ML-KNN: A lazy learning approach to multi-label learning, Pattern Recognition 40 (7) (2007) 2038–2048.

[37] Y. Prabhu, M. Varma, FastXML: A fast, accurate and stable tree-classifier for extreme multi-label learning, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2014), ACM, 2014, pp. 263–272.

[38] R. E. Schapire, Y. Singer, BoosTexter: A boosting-based system for text categorization, Machine Learning 39 (2-3) (2000) 135–168.

[39] C. N. Silla, A. A. Freitas, A survey of hierarchical classification across different application domains, Data Mining and Knowledge Discovery 22 (1-2) (2011) 31–72.

[40] S. Kiritchenko, S. Matwin, R. Nock, A. F. Famili, Learning and evaluation in the presence of class hierarchies: Application to text categorization, in: Proceedings of the 19th Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI 2006), Vol. 4013 of LNCS, Springer, 2006, pp. 395–406.

[41] S. Dumais, H. Chen, Hierarchical classification of web content, in: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000), ACM, 2000, pp. 256–263.

[42] B. C. Paes, A. Plastino, A. A. Freitas, Improving local per level hierarchical classification, Journal of Information and Data Management 3 (3) (2012) 394–394.

[43] F. Charte, A. Rivera, M. J. del Jesus, F. Herrera, A first approach to deal with imbalance in multi-label datasets, in: Proceedings of the 8th International Conference on Hybrid Artificial Intelligence Systems (HAIS 2013), Vol.

8073 of LNCS, Springer, 2013, pp. 150–160.

[44] E. Seymour, R. Damle, A. Sette, B. Peters, Cost sensitive hierarchical document classification to triage pubmed abstracts for manual curation, BMC bioinformatics 12 (1) (2011) 482.

[45] G. G. Sundarkumar, V. Ravi, A novel hybrid undersampling method for mining unbalanced datasets in banking and insurance, Engineering Applications of Artificial Intelligence 37 (2015) 368–377.

[46] J. Dou, Z. Gao, G. Wei, Y. Song, M. Li, Switching synthesizing-incorporated and cluster-based synthetic oversampling for imbalanced binary classification, Engineering Applications of Artificial Intelligence 123 (2023) 106193.

[47] N. V. Chawla, K. W. Bowyer, L. O. Hall, W. P. Kegelmeyer, SMOTE: synthetic minority over-sampling technique, Journal of Artificial Intelligence Research 16 (2002) 321–357.

[48] F. Charte, A. J. Rivera, M. J. del Jesus, F. Herrera, MLSMOTE: Approaching imbalanced multilabel learning through synthetic instance generation, Knowledge-Based Systems 89 (2015) 385–397.

[49] P. Szymański, T. Kajdanowicz, A network perspective on stratification of multi-label data, in: Proceedings of the First International Workshop on Learning with Imbalanced Domains: Theory and Applications (LIDTA 2017), PMLR, 2017, pp. 22–35.

[50] G. Wu, E. Y. Chang, Adaptive feature-space conformal transformation for imbalanced-data learning, in: Proceedings of the 20th International Conference on Machine Learning (ICML 2003), AAAI Press, 2003, pp. 816–823.

[51] A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet classification with deep convolutional neural networks, in: F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25 (NIPS 2012), Curran Associates, Inc., 2012, pp. 1097–1105.

[52] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, Journal of Machine Learning Research 12 (2011) 2493–2537.

[53] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166.

[54] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Computation 9 (8) (1997) 1735–1780.

[55] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014), ACL, 2014, pp. 1724–1734.

[56] A. Sen, M. M. Islam, K. Murase, X. Yao, Binarization with boosting and oversampling for multiclass classification, IEEE Transactions on Cybernetics 46 (5) (2015) 1078–1091.

[57] T. Lin, P. Goyal, R. B. Girshick, K. He, P. Dollár, Focal loss for dense object detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 42 (2) (2020) 318–327.

[58] R. Pappagari, P. Zelasko, J. Villalba, Y. Carmiel, N. Dehak, Hierarchical transformers for long document classification, in: Proceedings of the 2019 IEEE Automatic Speech Recognition and Understanding workshop (ASRU 2019), IEEE, 2019, pp. 838–844.