Evaluating Parabel in Title-Only Datasets

Angel Seiji Morimoto Burgos
University of Essex

ABSTRACT

Extreme multi-label classification is a complex task requiring to learn a classifier that can automatically annotate a data point with the subset of most relevant labels from a much larger collection of them. Several approaches have been developed across the years, but they have mostly been tested on full-text datasets, which are not always available in real life scenarios. [11] developed XMLC models and trained them using more accessible data (title-only); however, because of the uncommon datasets employed and the validation metric selected (sample-based F1 score), their work became hard to compare with other XMLC techniques.

To put the work of [11] into context, a state-of-the-art XMLC technique known as Parabel was implemented (based on the work of [13]) and tested using their datasets and their validation metric. It was found that Parabel did not perform as well as all of their models in terms of sample-based F1, but it still was competitive. This was partly due to default hyper-parameters being used and the fact that Parabel was designed by its authors to optimize ranking gain functions instead of other multi-labels functions like the sample-based F1 score.

CCS CONCEPTS

• **Computing methodologies → Natural language processing**; **Neural networks**; **Learning in probabilistic graphical models**; **Classification and regression trees**.

KEYWORDS

extreme multi-label classification, machine learning, supervised learning

ACM Reference Format:
Angel Seiji Morimoto Burgos. 2020. Evaluating Parabel in Title-Only Datasets. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Extreme multi-label classification (XMLC) concerns the task of learning a classifier that can annotate a data point with the most relevant subset of labels from a collection of labels that is orders of magnitude larger. It should not be confused with a multi-class classification problem, where only one label from many can be assigned to a data point. Even though not much research has been dedicated to XMLC problems (when compared to the amount of work done in the multi-class domain), it has the potential to be used across many applications. The most obvious one, perhaps, is that of automatic document tagging, in digital libraries for example. However, many other traditional problems can also be reformulated as XMLC tasks, such as suggesting queries in a web search engine based on the last query entered by the user (the queries themselves represent the labels of the XMLC problem) or recommending products to buy based on previous purchases.

The problem with XMLC is that it is a very complex task because of several reasons: the amount of available labels is in the range of thousands or millions, among those labels only a few of them are assigned to each data point, the size of the training dataset is usually big and the computational resources required to reasonably train a classifier are high. Given this, utilizing traditional Machine Learning (ML) techniques in a straightforward manner becomes difficult. Furthermore, some of them are by design only capable of dealing with binary classification problems, like Support Vector Machines (SVMs). Hence, in order to deal with XMLC, a number of approaches have been developed throughout the years and many of them have shown promising results. Nonetheless, in most cases the data used to train the classifiers is full-text data, which according to [6] does not showcase the real-life scenario of digital libraries (one of the main applications of XMLC), where full-text is many times made unavailable for text mining. They argue instead that metadata, such as titles, is much more accessible and should be leveraged to train XMLC classifiers. In fact, they even show in [11] that by using deep learning methods, classifiers built with the available title-only data can achieve a performance on par or even greater than that of classifiers trained with the available full-text data. The problem with their work, though, is that they utilized two datasets and an evaluation metric that is uncommon in the XMLC domain, hence making it difficult to compare their results with that of the other modern approaches.

With the purpose of contextualising the deep learning models developed by [11] with respect to the other techniques in the XMLC area, an algorithm known to be state-of-the-art in XMLC is implemented. This algorithm, known as Parabel, was originally created by [13]. In this paper, it is used with the datasets employed by [11], so that a better insight can be obtained of their methods as well as the suitability of Parabel and similar XMLC techniques to title-only datasets. A slight modification is also made to Parabel's probabilistic model so that higher scores can be obtained in terms of the sample-based F1 score, metric used in the experiments of [11].

The remainder of this paper is structured as follows. In section 2, a survey of the approaches utilized across the years for dealing with XMLC problems is presented. In section 3, a description of the Parabel technique is given. In section 4, the datasets used and the experimental setup are introduced. In section 5, the results of the experiments are presented and discussed. Finally, in section 6, the

Conference'17, July 2017, Washington, DC, USA
© 2020 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

main findings are summarized and directions of future work are given.

2 RELATED WORK

2.1 Naive 1-vs-All approach

Perhaps the simplest way to create an XMLC classifier is to use the 1-vs-All method, which consists of creating one independent classifier for each label to distinguish it from the rest. Naively building the XMLC classifier in this manner, however, has many drawbacks. First, the training time might be absurd as it will scale linearly with the number of labels: for a dataset with 325K labels, training could take up to 96 days [2]. Second, prediction time might also suffer, since every classifier would have to be evaluated [14]. Third, the size of the compund XMLC classifier (which includes every individual binary classifier) can be extremely large, as high as 870 GB for a dataset with 325K labels [2]. Using a naive 1-vs-All approach was considered infeasible and therefore was highly disregarded for a long time.

2.2 Embedding-based approaches

One of the two most popular approaches that have been used to deal with XMLC is that which uses embeddings. Essentially, given a set of n training points $\{(x_i, y_i)\}_{i=1}^{n}$, where the inputs are d-dimensional vectors and the outputs are L-dimensional vectors (i.e. $x_i \in \mathbb{R}^d$ and $y_i \in \{0,1\}^L$), the embedding-based approach consists of reducing the number of labels by projecting the label vectors onto a lower $\hat{L}$-dimensional space [4]. Traditionally, after this compression step is carried out, embedding-methods utilize the 1-vs-All approach to train classifiers to make compressed label predictions. Afterwards, a decompression technique is used to lift the compressed predictions back to the original label space [4]. There are several compression and decompression methods, which explains why many embedding-based approaches exist. Furthermore, they have a number of advantages, including simplicity, solid theoretical support, ease of implementation, capability to handle label correlations and suitability to online and incremental scenarios [4].

Classical embedding approaches work by assuming that the training label matrix is low-rank, which implies that there are correlations between the labels and hence that a small number of factors can be used to make predictions; however, this is not the case in many real-world scenarios [15], where a lot of *tail* labels usually exist and cannot be approximated well in low dimensional linear subspaces [4]. *Tail* labels are those that appear in a very small number of data points (in a popular benchmark dataset known as WikiLSHTC, around 300K out of 325K labels are present in less than 5 data points [4]).

In order to reduce the error induced by the low-rank assumption, other embedding-based methods have been proposed. SLEEC, for example, learns embeddings in a non-linear way that allows for the nearest neighbours to be preserved in the embedding label space [4]. This permits the algorithm to utilize a k-neareast neighbours (kNN) classifier in that space instead of performing a decompression operation, ultimately returning as a prediction the sum of the label vectors of the nearest neighbours of the test point [4]. To speed the prediction time, SLEEC first partitions the dataset by using k-means clustering and then learns an independent embedding and kNN classifier for each cluster [4].

SLEEC was able to outperform the state-of-the-art methods in several benchmark datasets when it was published, obtaining higher values of *precision at k*, P@k (with $k = \{1,3,5\}$), than even the best tree-based method of the moment, FastXML (see section 2.3). Regardless, it still had some problems. For instance, since it used k-means clustering before learning the embeddings, it partitioned the dataset using feature vectors and not the labels' information, which led to the possibility of similar label vectors not being placed in the same partition [15]. Also, its prediction time could have been better: even though it was comparatively faster than traditional embedding-based methods (as far as 300 times faster), it still took 8 milliseconds per test instance in the WikiLSHTC dataset, while FastXML required only 0.5 milliseconds [4]. To compensate for these deficiencies, other embedded-based methods were created. AnnexML, for example, first utilizes an approximation technique for finding the nearest neighbours of the label vectors in a quick way, which it uses to build a k-nearest neighbours graph (kNNG) and partitions it trying to preserve its original structure as much as possible [15]. Each subgraph generated is then projected to its own embedding space [15]. A multi-class classifier is then learnt to predict to which partition a test point belongs. After the partition is predicted, the approximate nearest neighbours of the test point are efficiently found (by looking at the kNNG in the embedding space) and used to label it [15].

When put up against SLEEC, FastXML, PfastreXML (see section 2.3), PD-Sparse (see section 2.4) and an implementation of a Probabilistic Label Tree (PLT), it achieved superior performance in almost every dataset and every P@k measure (with $k = \{1,3,5\}$). Also, it demonstrated that it could be up to 58 times faster in prediction than SLEEC without diminishing its performance [15].

2.3 Tree-based approaches

The other approach apart from embedding-based methods that has been popularily utilized for XMLC throughout the years is concerned with learning a hierarchy over the labels in a tree-like way [14]. In general, doing this was thought to trade-off accuracy for increased speed in predictions when compared to embedding-based methods [2]. However, with the arrival of XMLC classifiers like FastXML, it was found that reasonable results could also be achieved with this paradigm. FastXML improves over its predecessors, like the Label Partitioning by Sublinear Ranking (LPSR) [17] and the Multi-Label Random Forest (MLRF) [1] algorithms, by incorporating several changes, the first one being that it learns a hierarchy over the feature space instead of the label space [14]. The reasoning behind this decision is that usually only a small set of labels is active in a given region of the feature space [14]. A test point is passed down throughout a learnt ensemble of trees and the prediction is created by returning a ranked list of the active labels that occur more often in the ensemble' leaf nodes reached by the data point [14]. FastXML partitions the feature space to create the trees in a way that optimizes a ranking loss function known as normalized Discounted Cumulative Gain (nDCG) [14].

Choosing nDCG as the loss function was perhaps a critical choice in FastXML, as previous methods that optimized rank insensitive measures, like entropy in MLRF, were not suited for the XMLC domain, where predicting a small number of relevant labels is more important than returning many irrelevant ones [14]. In fact, an improved version of FastXML, PfastreXML, was later created with one of its key contributions being the fact that it optimized a propensity scored version of nDCG [9]. Propensity models were proven to boost the prediction accuracy of tail labels, which are vastly present in most real-world datasets and are hard to deal with (as mentioned in section 2.2) [9]. In experiments done with the popular benchmark datasets, PfastreXML was able to achieve a considerably better performance at the propensity scored version of P@k (PSP@k) and nDCG@k (with $k = \{1, 3, 5\}$) than baseline algorithms, FastXML and even SLEEC [9]. Moreover, PfastreXML preserved the scalibility and parallelizable properties of FastXML, having only a minor training and testing overhead, still being able to train on the WikiLSHTC-325K dataset in under 30 minutes in a 16-core computer [9].

2.4 Improved 1-vs-All approaches

In section 2.1 it was mentioned that using the 1-vs-All approach was infeasible due to the extremely long training and prediction time required. However, as research progressed, clever ways of utilizing the 1-vs-All approach were designed, in hopes of achieving better performance than embedding and tree-based methods, which sometimes exchanged accuracy for efficiency. An example is the Primal-Dual Sparse (PD-Sparse) algorithm [20]. It works on the assumption that for each data point there are only a few active labels and that the feature space contains enough information to discriminate between labels [20]. This assumption allows the algorithm to utilize a margin-maximization loss function, so that a very sparse solution in primal and dual is obtained [20], which is beneficial regarding time and memory efficiency [10]. It is able to reduce the training time and model size considerably while keeping an accuracy that often exceeded that of traditional 1-vs-All methods (in datasets where they could scale) and representative embedding and tree-based techniques, SLEEC and FastXML [20]. However, prediction time was still linear with respect to the number of classes, as mentioned by [10].

Another scaled-up version of the 1-vs-All approach is the Distributed Sparse Machines for Extreme multi-label Classification (DiSMEC) [2]. To scale the 1-vs-All approach to the XMLC problem, DiSMEC added two key components: a double layer of parallelization and an ambiguity control mechanism. The former consisted of dividing the training into batches of $\hat{L}$ labels, where $\hat{L}$ represented the number of computing nodes available. In their experiments, [2] had access to 32 nodes, each one with 32 cores, so a total of 1024 labels could be used simultaneously to learn their binary classifiers. Furthermore, the distributed nature of the approach could be leveraged to speed up the prediction, to the point of being close to FastXML [2]. In terms of the ambiguity control mechanism, [2] realized that most values within the learnt weight vectors for any dataset were very close to 0. These weight values did not add decisive information to distinguish one label from another and instead increased the total model size dramatically. Hence, [2] decided to don't store all values that were less than a given threshold (0.01 in their experiments), achieving a total model size of 3 GB for WikiLSHTC-325K (instead of the original 870 GB size mentioned in section 2.1). DiSMEC outperformed the representative models of embedding and tree-based methods when it was published, SLEEC and FastXML, in almost all tests and by around 10% to 15% in three datasets.

The disadvantage of DiSMEC, nonetheless, was that its speed was very dependent on the computational power used. In the case in which only 100 cores were available (an order of magnitude less than the experiments in [2]), it would take a few days to train the classifier on large datasets, like WikiLSHTC-325K [19]. The idea of distributed computing, though, motivated [19] to create PPD-Sparse, a parallelizable version of PD-Sparse. In their experiments, PPD-Sparse was shown to significantly reduce the memory footprint of PD-Sparse and the training time of 1-vs-All approaches (less than 1 hour in all tested datasets using 100 cores), while preserving an accuracy competitive with the then state-of-the-art methods: FastXML, PfastreXML, SLEEC and DiSMEC [19].

2.5 Deep learning approaches

Neural networks have been employed in a lot of fields due to the success of deep learning in extracting features even from high-dimensional data such as images. Not much was done though until recently in the area of XMLC. Possibly one of the first attempts at using deep learning in an XMLC setting was [10], where a Convolutional Neural Network (CNN), XML-CNN, was designed to predict the labels to be assigned to the documents in the popular benchmark datasets. It was based on the ideas of CNN-Kim, one of the first CNNs to be applied on multi-class text classification [10]. Contrary to most embedding and tree-based methods, XML-CNN used pre-trained and concatenated word embeddings to represent the input documents, instead of the Bag-of-Words (BoW) notation which does not take context into consideration [10]. The architecture of XML-CNN consisted of a convolutional layer with several filters, a dynamic max-pooling layer, a fully connected layer and an output layer with nodes equal to the number of labels in each dataset [10]. To test its performance, P@k and nDCG@k measures were evaluated for the benchmark datasets and its results were compared to those of techniques representative of other XMLC approaches (SLEEC, FastXML and PD-Sparse) as well as to successful multi-class text deep learning methods that were adapted to work in the multi-label domain [10]. In most of the cases (11 and 12 out of 18 P@k and nDCG@k measures) XML outperformed every other method, and it was found that it was particularly good in datasets with a high number of training instances per label [10].

Different neural network architectures apart from CNNs have been used to deal with multi-label classification. In [12], for example, the problem is treated as a sequence-to-sequence prediction in which two Recurrent Neural Networks (RNNs) are utilized, one for encoding the input text and one for decoding and predicting labels in a sequential order. Different ways of ordering the labels are explored for obtaining better accuracy [12]; however, as [21] mentions, the assumption of having an order between labels is not rational for the XMLC domain. In [11], CNN and RNN architectures using word embeddings as input are revisited once again and compared with Multi-Layer Perceptrons (MLPs) utilizing a TF-IDF (Term Frequency = Inverse Document Frequency) bag-of-unigrams input representation. A focus is also made on determining how much more training samples are needed to achieve similar or better performance by relying only on document titles instead of the full texts [11]. For this, five title-only sets are made for each of the two datasets used, EconBiz and PubMed (see 4.1), so that each title-only set is a power of 2 in size of the full-text dataset. Interestingly, for the full-text and all the title-only sets of the EconBiz corpus, MLP models achieved the best results [11]. Same occurred in PubMed, except for the 3 title-only sets with most training examples, where the best scores were obtained by an RNN.

To the surprise of the authors of [11], the CNN architecture performed worse than the rest despite having been shown to be competent in XML-CNN. It is important to note that a reason might have been the fact that [11] used a sample-based F1-measure for evaluation instead of the commonly used P@k or nDCG@k measures. They decided to do this because it was commented by [6] that sample-based F1 measures reflected more the real-life process carried out for annotating individual documents. In [7] though, a CNN outperformed an MLP even when a similar measure to sample-based F1 (micro F1) was used. One major difference to the previous work is that in this case the same input representation, word embeddings, was used for both architectures. [7] also suggested that hyperparameter tuning should be done when possible, as datasets with different number of labels and distributions require very different values for optimal results.

2.6 Combined approaches

A number of methods have been developed in recent years for the XMLC setting that cannot be properly placed within one of the categories of the previous sections. One of such methods is Parabel, which can be considered a mixture of 1-vs-All and tree-based techniques [13]. Essentially, Parabel learns a hierarchy of labels in the form of a balanced tree, similar to tree-based approaches, but at each leaf node it has a 1-vs-All classifier for each label present in it [13]. The tree is partitioned in such a way that similar labels end up in the same leaves. To reduce the excessive training time present in naive 1-vs-All implementations, Parabel significantly decreases the number of data points used for training each classifier. Other techniques do something similar by applying traditional methods for subsampling negative training points and which can lead to performance loss if used excessively, like subsampling at random or employing the primal-dual optimization algorithm of PPD-Sparse [13]. Parabel instead bases itself on the idea that accuracy can be preserved if the negative data points used for training a classifier of any given label are only those that have the most similar (confusing) labels to it (i.e. data points from the other labels in the same leaf node) [13].

In Parabel, the 1-vs-All classifiers in the leaves learn to estimate a probability distribution of the labels, somewhat generalizing the well-known Hierarchichal SoftMax (HSM) model often employed for learning word embeddings [13]. In fact, as mentioned by [21], Parabel uses a type of Probabilistic Label Tree (PLT). A PLT is itself a generalized version of HSM to the multi-label domain, as HSM was originally designed for multi-class problems [18]. Parabel was able to achieve to achieve 9% higher accuracy, as well as 20 and 38 times less training time and size than the leading tree-based method, PfastreXML [13]. Additionally, it obtained comparable accuracy to state-of-the-art 1-vs-All techniques, DisMEC and PPD-Sparse, with 600-900 times faster training and 60-13,000 times faster predictions [13].

AttentionXML is another method that utilizes a combination of several XMLC approaches [21]. It works somewhat similar to Parabel in the fact that it builds a PLT using top-down hierarchical clustering; however, it then uses a procedure for compressing the PLT to make it shallow and wide. Additionally, as its classifiers, it uses an attention-aware bidirectional RNN deep learning model at each level of the PLT [21]. Using a multi-label attention mechanism, AttentionXML is able to represent input text differently for each label, so as to capture the most relevant parts of the text for each of them (something that is not considered in most state-of-the-art methods) [21]. The deep learning models in AttentionXML are trained in a top-down fashion and most of the parameters of the model at level d are initialized using the parameters of the previous level $(d-1)$, so that faster convergence is achieved in models of deeper levels within the PLT [21]. Another notable difference is that AttentionXML uses word-embeddings as its input instead of BoW features (which were used in Parabel), since the former provide more semantic context information than the latter [21]. Furthermore, as many other methods, it uses an ensemble (in this case of PLTs) to achieve even better results. When compared against the performance of state-of-the-art methods of every type of approach, such as AnnexML (embedding-based), PfastreXML (tree-based), DiSMEC (1-vs-All), XML-CNN (deep learning), Parabel (tree-based and 1-vs-All) and others, it achieved the best P@k and PSP@k score in every dataset used [21].

Similar to AttentionXML, another XMLC technique known as Scalable LInear extreme ClassifiErs (Slice) was able to outperform every other state-of-the-art method (it was not tested against AttentionXML) in experiments run over several datasets [8]. Slice works by learning a separate linear classifier for each label in a 1-vs-All fashion. Similar to Parabel, it focuses on training each classifier only with the most confusing negative examples, which it estimates using a generative model with the Hierarchichal Navigable Small World Graph (HNSW) algorithm, which is itself a type of Approximate Nearest Neighbour Search (ANNS) [8]. HNSW weakens as the feature space grows in dimensionality. Partly because of this is that a low-dimensional Convolutional Deep Structured Semantic Model (CDSSM) embedding is used instead of a BoW representation [8]. Another reason is that Slice was made to predict subsets of relevant queries from a search engine based on a query given by the user as input, and low-dimensional dense embeddings have been shown to be better for representing queries in the literature [8]. As it is common practice, Slice was compared against many of the state-of-the-art XMLC methods and showed very good performance, for example, it demonstrated to be 13% more accurate than PPD-Sparse while being 10 and 100 times faster at training and prediction respectively. Its main advantage, though, was that it could scale in an efficient and accurate way to a dataset with 100 million labels and 240 million training points, something which had not been nearly possible with any of the previous XMLC classifiers [8].

3 METHODS

Most of the techniques described in section 2 have been compared against each other by means of evaluating their performance on a set of well-known XMLC benchmark datasets, which have been collected in [3]. Those datasets, however, consist mostly of full-text documents, which according to [11] are many times unavailable, particularly in the context of digital libraries, where copyright laws and regulations might be placed to prevent analysis of their text contents. Documents metadata, on the other hand, is often accessible, which is why the authors focused their research on it in [6, 11]. Nevertheless, despite finding interesting results, such as the fact that an MLP (rarely or never utilized before for the XMLC domain) may outperform more complex neural network architectures, their work became hard to compare with the rest of the methods; not only because they utilized different datasets, but also because they evaluated the results with an unusual validation metric for XMLC problems, the sample-based F1-score.

Assessing the results of [11] and putting them into context with respect to the state-of-the-art approaches was desired, so a representative and modern XMLC technique was decided to be implemented, trained and evaluated in this paper using the titles-only datasets and the metrics used by [11]. The technique chosen was Parabel, because it is a contemporary non-deep learning algorithm (so that the deep learning based methods of [11] can be compared against other approaches) which has been shown to achieve great performance in an efficient way over the benchmark datasets. Parabel was already described briefly in section 2.6; however, a more detailed explanation of its underlying mechanisms are explained in this section. Each of the components are described based on the ideas of the original paper [13].

3.1 Label representation

Being a mixture of tree-based and 1-vs-All approaches, Parabel requires a method for creating a tree that partitions the label or feature space so that efficient predictions can be made in a logarithmic way. Even though FastXML demonstrated that partitioning the feature space can be just as good as partitioning the label space while being much faster, Parabel was conceived with a clever way of partitioning the label space. Its fundamental idea consists of representing a label as a feature vector created by averaging the training points that were annotated with that label. More formally, given a dataset of size n, $\{(x_i, y_i)\}_{i=1}^{n}$, with D-dimensional training points $x_i \in \mathbb{R}^D$ and L-dimensional label vectors $y_i \in \{0,1\}^L$, the feature vector of a single label l is expressed as:

$$v_l = \frac{v'_l}{\|v'_l\|_2} \quad \text{where} \quad v'_l = \sum_{i=1}^{n} y_{il} x_i \tag{1}$$

Given this representation, the similarity between two labels, l_1 and l_2, can be computed as follows:

$$s = v_{l_1} \cdot v_{l_2} \tag{2}$$

In conjunction, Equations 1 and 2 reflect that, under Parabel's assumption, two labels are considered similar if they are present in data points that are akin. The main advantage brought by this is that it works reasonably well on tail labels in comparison to other representations, since the similarity measure between two tail labels may be high even if they don't have any training points in common (as long as the training points themselves are similar between them). Also, in the case in which the majority of the labels in a dataset are tail labels (which is often common in real-life applications), it is computationally efficient to compute the feature vector representation of each label, as only a small number of training points has to be averaged for each of them.

3.2 Building the label tree

Iteratively partitioning the label space by directly optimizing a constrained $k = 2$-means objective so that two clusters of the same size are built for each level of the tree is a hard problem. To try to approximate this, Parabel employs an algorithm which converges to a local optimum. This algorithm, known as Hierarchical Spherical Balanced K-Means (with $k = 2$ because each partition creates two clusters, such that a binary tree is formed), works as follows. First, a node is created for representing the root of the tree/hierarchy. This root node contains all the labels in the dataset. Then, for creating its left and right children, the partition procedure is executed over the labels.

The partition process first creates a centroid for each cluster to be generated. Let's call μ_+ and μ_- the centroids of the clusters representing the left child and right child respectively. These centroids are initialized by uniformly sampling from $\{v_1, v_2, ..., v_{L'}\}$, where L' is the number of labels of the parent node. The samples are taken without replacement so that μ_+ and μ_- are not equal. Then, the similarity of each label vector to the centroids is computed by using Equation 2, such that $s_{l+} = \mu_+ \cdot v_l$ and $s_{l-} = \mu_- \cdot v_l$ are the similarities of label l to the left and right centroids. An overall inclination s of each label towards one or another centroid is then computed by substracting the similarity to μ_- from the similarity to μ_+. By doing this, intuitively, if s is positive for a given label, it means that the label is more inclined to the left centroid, whereas if it is negative, it is closer to the right centroid. However, since a balanced tree is desired for the advantages it provides in terms of logarithmic lookup times, labels are not assigned to the centroids directly based on the sign of their s value. Instead, they are first sorted in decreasing order of s, after which the first half of the sorted list is assigned to the left cluster and the other half to the right. If the number of labels is odd and so the clusters cannot be of the same size, the assignment of the label in the middle of the sorted list is done based on the sign of its s value. Once all labels have been assigned to a cluster, the centroids are updated by averaging the label vectors assigned to them, which is expressed as follows:

$$\mu_+ = \frac{\sum_{l \in Y_+} v_l}{\|\sum_{l \in Y_+} v_l\|_2} \quad \text{and} \quad \mu_- = \frac{\sum_{l \in Y_-} v_l}{\|\sum_{l \in Y_-} v_l\|_2} \tag{3}$$

where Y_+ and Y_- are the sets of label vectors assigned to the left and right clusters accordingly. Ideally, the process of finding the similarities, assigning labels to the clusters based on them and updating the centroids is performed repeateadly until there is no change in the assignments of labels. However, in practice, this can sometimes be very costly, that is why a numeric threshold ε_p can be specified to determine when to stop. When this threshold is put in place, instead of computing the differences of labels' assignments, the total similarities of the labels to their assigned clusters is compared between two consecutive iterations. The total similarities of the labels to their assigned clusters for a given iteration is calculated as follows:

$$\sum_{l \in Y_+} s_{l+} + \sum_{l \in Y_-} s_{l-} \tag{4}$$

If the value calculated from Equation 4 does not increase by at least ε_p from one iteration to the next, then the procedure is stopped, with the resulting partitions being the ones obtained from the last iteration executed.

Once the partition is done in the root of the tree and its children have been generated, each with its mutually exclusive subset of labels, the partition process is done once again, but now on each of the children to generate their own children. This is recursively done until the partitions generated have less than some predefined number of labels γ, in which case they are considered leaf nodes. This way, a balanced binary tree is built, representing a hierarchy over the label space.

3.3 Training Parabel

As a 1-vs-All technique, Parabel needs to have a classifier for each label with which the tree was built. These classifiers, however, are placed in the leaves of the tree. They are trained using as positive examples the data points that were tagged with the label they represent and as negative examples the points that were tagged with any of the other labels in the same leaf node. In traditional 1-vs-All approaches, each classifier is trained with all the dataset, making the training process very computationally expensive. Parabel instead leverages the characteristic of the created tree of grouping together similar labels to pick a much smaller set of negative examples which is, in addition, useful for properly tagging a data point with the correct label even when there are many similar (and confusing) but incorrect labels. In the regurarly used datasets with high proportitions of tail labels, the decrease in size of the data used for training each classifier is even more significant. To exemplify, let's assume a label tree with leaf nodes that have around 100 labels and let's also assume that the dataset used to train the algorithm was the WikiLSHTC-325K, which has around 300k (out of 325k) tail labels, each one being active in at most 5 data points (as mentioned in section 2.2). In such case, for any classifier of a tail label that is in a leaf node where all the other labels are also tail, there would have been at most 500 points for training it. Additionally, this is considering that all the tail labels in that leaf node had no common training points, which is hardly the case given that, if they were grouped together by the Hierarchical Clustering K-Means algorithm, they should have had some level of similarity between them (and similarity between labels was defined based on similarity between training points as explained in section 3.1). Even so, comparing a subset of 500 points with the total dataset of around 1.8 million points clearly shows the speed advantage offered by Parabel's architecture.

Recall that even though a balanced tree was built with Parabel so that predictions can be made in logarithmic time by passing a test point through the correct path in the tree, this is not possible in a straightforward way because the tree was built over the label space and not over the input feature space. So, in order for a test point to traverse the tree all the way to the leaves, Parabel first must learn a binary classifier at each internal node. These classifiers use as positive examples all the training points that were tagged with any of the labels belonging to the node. As for negative examples, they utilize the rest of the data points that were used as positive examples in the parent node (i.e. those that were tagged only with the labels of the sibling nodes).

Parabel is a probabilistic model that represents the joint probability that a set of labels is applicable to a data point. For brevity, a formal mathematical description of the probabilistic model is omitted here, but can be looked up in the original paper [13]. It is important to say that because Parabel models a joint probability, any classifier can be used for internal and leaf nodes as long as it can learn the parameters of a probability distribution over the data points. Using the classifiers as is, though, can lead to really big models. For instance, if the classifiers used were Logistic Regressions (LRs) and each point in a dataset of size N was a D-dimensional feature vector with annotations from a set of L labels, then at least DL weights would have to be learnt (because LRs learn a weight for each feature). The memory needed for storing such a model could quickly become unreasonable as L and D increase (recall the 870 GB of the 1-vs-All model for the WikiLSHTC-325 dataset). To cope with this, a threshold ε_w can be set for the learnt weights of each classifier, such that all the weights that are below ε_w are set to 0 and not stored. This idea was originally formulated and tested in the work of [2], where it was found that a vast majority of learnt weights in 1-vs-All classifiers were very close to 0 and did not help in discriminating one label from another while being very taxing in terms of model size.

3.4 Making predictions

The way Parabel makes its predictions is optimized for gain functions that consider the top ranked relevant predictions (such as P@k and nDCG@k) instead of conventional multi-label or multi-class loss functions, as the algorithm's authors argue that ranking functions are usually preferred for XMLC applications, such as tagging and recommendation problems. Hence, predictions are made so that, given a set of labels y and a data point x to tag, the labels are ranked in descending order of their marginal probabilities of being relevant to x, i.e. in descending order of $P(y_l = 1|x)$ where $y_l \in y$. The marginal probability of a given label l (that belongs to leaf node n) being active in data point x is determined by:

$$\mathbb{P}(y_l = 1|x) = \mathbb{P}(y_l = 1|z_n = 1, x) \prod_{\hat{n} \in A_n} \mathbb{P}(z_{\hat{n}} = 1|z_{\mathcal{P}_{\hat{n}}} = 1, x) \tag{5}$$

where z_i represents that node i is a binary variable having a value of 1 if node i was visited and 0 otherwise, A_n is the set of ancestor nodes of node n in the tree (excluding root) and $\mathcal{P}_{\hat{n}}$ is the parent of node $\hat{n}$. The same function can be calculated for an internal node m, but in that case it is not described as the marginal probability of a label l being active in x, but rather as the marginal probability of any label l from a set of labels L' being active in x (or more simply, as the probability of node m being visited):

$$\mathbb{P}(z_m = 1|x) = \mathbb{P}(y_1 = 1 \cup \cdots \cup y_{l'} = 1|x) = \\ \mathbb{P}(y_1 = 1 \cup \cdots \cup y_{l'} = 1|x) \prod_{\hat{m} \in A_m} \mathbb{P}(z_{\hat{m}} = 1|z_{\mathcal{P}_{\hat{m}}} = 1, x) \quad (6)$$

The notation is different in both Equations, yet in practice they behave the same when using 1-vs-All classifiers, since each internal node will represent the set of labels associated with it as a single positive class, i.e. as a single label, reducing Equation 6 to 5.

Even with the above formulations, if no additional mechanism is put in place, still the high costs of predicting over L labels are incurred and, furthermore, the built label tree is under-utilized. To avoid this, a greedy and breadth-first (level-order) traversal is employed, where only the P most probable paths are traversed at each level. At each iteration of the traversal, a level is explored (starting from the root), where the probabilities of point x belonging to each node being explored (the first nodes explored are the children of the root) are calculated using Equation 6. The P most probable nodes are then chosen for exploring its children in the next iteration. The procedure continues until P leaf nodes are found. The marginal probabilities of all the labels (contained in those leaf nodes) being active in point x are then computed using Equation 5. Finally, the labels for which the marginal probabilities were found are ranked and returned.

3.5 Enhancing performance

Like most approaches in the XMLC domain, Parabel can make use of an ensemble to improve performance even further. If an ensemble of size T is utilized, then T different label trees are constructed as different random initializations in the Hierarchical Balanced K-Means algorithm lead to different solutions. In such case, each tree is trained separately by following the procedure described in section 3.3. As for making predictions, a data point x is passed down each tree until the marginal probabilities of the most likely labels are found in the leaf nodes as described in section 3.4. Afterwards, the final score of each label analysed is computed by averaging its marginal probability across all trees $\frac{1}{T}\sum_{t=1}^{T}\mathbb{P}_t(y_l = 1|x)$.

4 EXPERIMENTAL SETUP

4.1 Datasets

As mentioned in section 3, the purpose of this paper was to properly compare the deep learning methods and results used by [11] against those of popularly considered state-of-the-art XMLC techniques like Parabel. For this comparison to be made, the same title-only datasets used in [11] were employed.

The first dataset was EconBiz, which was created by extracting the titles of the English publications that had annotations from the EconBiz online search platform for economics. The annotations were subject headings chosen by experts from a regulated set of terms known as the *Thesaurus for Economics* (STW).

The second dataset was PubMed, which was created by gathering the titles of open access English publications from PubMed Central (a collection of documents covering topics from biomedical and life sciences supplied by the US National Library of Medicine) and the training set of the semantic indexing task of the 2017 challenge organized by the BioASQ organization. All the publications in the PubMed dataset were annotated by human experts using the so-called *Medical Subject Headings* (MeSH).

Table 1: Characteristics of the EconBiz and PubMed title-only datasets. $|X|$ represents the number of data points, $|L|$ indicates the number of labels present, x/l means the average number of documents tagged per label, and l/x is the average number of labels assigned to a document. T_{all} and T_1 represent the biggest and smallest subset of each dataset as described in section 4.2. The information presented in this table was retrieved from [11].

	EconBiz		PubMed	
	T_{all}	T_1	T_{all}	T_1
$\|X\|$	1,064,634	70,619	12,834,026	646,513
$\|L\|$	5,661	4,849	27,773	26,276
x/l	819.1	75.8	5,852.3	331.0
l/x	4.4	5.3	12.6	13.5

Both datasets were provided by [11]. The characteristics of each of them is showcased in Table 1.

4.2 Experiments

For proper comparison with the results of [11], their procedure was replicated as much as possible in this paper but using Parabel as the XMLC classifier. In their work, they split each title-only dataset in several smaller subsets. The smallest subset for each dataset was comprised of only those publications for which the full-text was available (around 5-7% of all the documents for each dataset), because they wanted to compare the performance of XMLC classifiers trained on full-text with that of classifiers trained only on titles. According to their previous study [6], training on full-text was much better, so they wanted to verify how much more data was needed for the title-only classifiers to reach performance comparable to their full-text counterparts (hence the creation of the subsets of different sizes for title-only data).

Hereafter, the smallest subset of each dataset will be referred to as T_1. The subsets which are x number of times bigger than T_1 will be expressed as T_x. The subset that contains essentially all the elements in the original dataset will be named T_{all}. In the end, imitating the procedure of [11], 5 different subsets were used for each dataset: T_1, T_2, T_4, T_8 and T_{all}. In their study, they decided to divide the T_1 subset into 10 folds, so that in each iteration of the cross-validation procedure, 9 folds were chosen for training and 1 for testing. The split of the data into folds was only done for T_1 because they wanted to always test on the data points that also had their equivalent full-text version (for appropriate comparison between the title-only and full-text classifiers). For the cases when any of the other subsets were used (which were themselves supersets of T_1), the cross-validation procedure was performed using the same folds as in T_1, but the extra data not included in T_1 was used as part of the training set at each iteration.

In the present paper, the 10-fold cross-validation strategy of [11] was employed for all the subsets in the EconBiz dataset. For the PubMed subsets, though, only one training procedure was done using folds 2-10 plus the extra data of the subset (no extra data in T$_1$), while a single testing operation was done utilizing fold 1. Originally, the objective was to perform cross-validation on both datasets, but it turned out to be very costly computational-wise, so the single training-testing procedure was used instead. Also, any model that would have taken more than 24 hours to be trained and evaluated was omitted from the results in section 5, to represent in a way the scalability of the Parabel technique. It is also important to note that, even though the experiments were carried out using subsets of the datasets with the same sizes as those utilized by [11], only T$_1$ and T$_{all}$ of each dataset were guaranteed to be exactly the same to the ones used by them. This was because [11] did not specify which data points they added from the original dataset to T$_2$, T$_4$ and T$_8$; they just mentioned that the size of the subset was doubled each time.

For gauging the performance of Parabel every time a testing procedure was executed with a test set, two evaluation metrics were used: sample-based F1 score and P@k (with $k = \{1, 3, 5\}$). The former was employed to be able to compare Parabel's performance with that of the deep learning methods utilized in [11] (whose results were all reported using sample-based F1). The latter was used to observe how Parabel performed in the EconBiz and PubMed datasets in terms of a gain function it has been designed to optimize.

For a collection of n data points X and a ground-truth label set Y, the sample-based F1-score of a classifier h can be defined as follows [22]:

$$P(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)|}{|h(x_i)|} \tag{7}$$

$$R(h) = \frac{1}{n}\sum_{i=1}^{n}\frac{|Y_i \cap h(x_i)}{|Y_i|} \tag{8}$$

$$F_1(h) = \frac{2 \cdot P(h) \cdot R(h)}{P(h) + R(h)} \tag{9}$$

where $P(h)$ and $R(h)$ are the precision and recall of classifier h respectively. As for the P@k metric, the following equation holds [5]:

$$\begin{aligned} P@k = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{k}\sum_{j=1}^{k} rel(rank(j, h(x_i)), Y_i) \quad \text{where} \\ rank(j, L) = L[j] \quad \text{and} \\ rel(l, Y_i) = \begin{cases} 1, & \text{if label } l \in Y_i \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{10}$$

The above equation assumes that $h(x_i)$ returns a set of labels ordered from most to least relevant for point x_i.

4.3 Input representation and preprocessing

In Parabel's original paper [13], the authors stated that the algorithm is able to work regardless of the text input representation used, whether it be low-dimensional, dense embeddings or high-dimensional, sparse BoW vectors. They did mention, however, that even though state-of-the-art results can be obtained through low-dimensional features, high-dimensional features can get an even better performance if the squared hinge loss is used instead of the log loss in the classifiers' optimization problem; hence they mostly used the BoW notation in their experiments. Following their actions, a sparse notation was also chosen for this paper regardless using Logistic Regression (LR) classifiers at the internal and leaf nodes (LRs try to minimize the log loss function). Term frequencies, though, were not used directly for the BoW representation. Instead TF-IDF features were utilized, as they allow reducing the influence or weight of words that are very common across the different documents in a dataset (and hence have little discriminative power). TF-IDF can also be thought of as a mechanism for automatically ignoring stop words without using a specific dictionary of such words. To calculate the TF-IDF value of a word w in a document d from a dataset D of documents, the following equation can be used [6]:

$$\begin{aligned} \text{TF-IDF}(w, d) = \text{TF}(w, d) \cdot \text{IDF}(w, D) \quad \text{where} \\ \text{IDF}(w, D) = 1 + \log \frac{|D| + 1}{|\{d \in D : w \in d\}| + 1} \end{aligned} \tag{11}$$

In Equation 11, $TF(w, d)$ is the number of times word w appears in document d.

It is worth mentioning that [11] utilized a TF-IDF BoW representation for their MLP architectures as well (which were in most cases the best performing models in their experiments), which is also partly why that representation was chosen for this paper. Preprocessing steps were also replicated from their work so that Parabel could be evaluated more properly, reducing the likeliness of the comparison made being influenced by the preprocessing techniques used. These steps were performed before extracting TF-IDF features and consisted of the following. First, all words separated by a hyphen were joined. Then, all non-alphabetic characters were discarded, as well as single-character words like 'a'. Afterwards, all words were lower-cased and lemmatized using the morphological processing of the English lexical database WordNet [16]. Also, the TF-IDF features were only generated for the 25,000 most common words across all the documents in a dataset, exactly like [11] did for their *Base-MLP* model (their other MLP model utilized the 25,000 most common words as well as the 25,000 most common bi-grams, i.e. combinations of two consecutive words).

4.4 Details of Parabel version used

For all the experiments carried out, the same version of Parabel was used, i.e. with the same hyper-parameters. First and foremost, the classifiers used in the internal and leaf nodes were LRs (as mentioned in section 4.3) with the following hyper-parameters: an L2 regularization with a dual formulation was used with C equal to 1 (C is inverse to the regularization strength), a bias was added to the learnt decision function and a maximum number of iterations for the solver to converge was set to 20. Most of these were set by using the default values of Parabel's original implementation [13]. As for the other hyper-parameters exclusive of the XMLC technique itself and not of its underlying 1-vs-All classifiers, T was set to 1 (i.e. a single tree instead of an ensemble), γ to 100, ε_p to 0.0001, ε_w to 0.1 and P to 10. All of these, with the exception of T, were used as the default options in [13], though setting ε_w to 0.1 was used only in the experiments where the square hinge loss function was utilized (0.5 was instead used in when the log loss was chosen). As for the reason why T was set to 1, it was mostly to reduce the computational cost of having to train an ensemble.

In section 3.4 it was explained that Parabel is optimized for ranking gain functions and not for other metrics such as the sample-based F1 score, which is the main comparison metric in this paper. Using Parabel's probabilistic model as is could result in really low F1-based scores because it would be difficult for a label l in a leaf node n to achieve a marginal probability of being relevant to data point x greater than 0.5, given that it would be computed as the multiplication of the probabilities of every node within the path from root to n (see Equation 5). A way to guarantee that the sample-based F1 score is not affected by the probabilistic model of Parabel could be to set a threshold lower than 0.5 for considering a label to be relevant for a data point. Finding the optimal threshold, however, is quite difficult and, moreover, varies depending on the dataset used, as more labels would mean a taller tree with more probability multiplication operations than a smaller tree present when the number of labels is less. Instead of this, what was decided to be done in the end was to consider the marginal probability of a label l in a leaf node n of being relevant to data point x to be given without considering the probabilities of the ancestor nodes. Essentially, Equation 5 was reduced to the following expression:

$$\mathbb{P}(y_l = 1|x) = \mathbb{P}(y_l = 1|z_n = 1, x) \tag{12}$$

Even though this simplification violates the probabilistic model optimized for ranking gain functions, it was considered valid to be performed for the purposes of improving the sample-based F1, as the probability of each 1-vs-All label classifier would now be directly used to evaluate whether the label was relevant to the data point. This way, the common 0.5 value would be preserved as the threshold.

5 RESULTS AND DISCUSSION

The results of the experiments carried out in terms of the sample-based F1 score are shown in Table 2 along with the values reported by [11]. It can be seen from the outcomes that Parabel does not achieve the best performance in any of the tests. However, one thing to note is that its effectiveness relative to the rest of the methods is quite different between the two types of datasets. In EconBiz, for instance, is consistently the worst classifier across all subsets, starting at T_1 and ending at T_{all}, both around 0.60 F1 points less than the second worst classifier (CNN). On the contrary, for the PubMed subsets, it starts as the worst classifier at T_1, but quickly becomes the third and second best model in T_4 and T_8 respectively. Why this occurs is not precisely known, but several reasons are considered. One of them is the fact that the scores in the PubMed subsets are not reliable enough (because they were not obtained using cross validation) and hence cannot be compared directly to the results of the deep learning techniques. Another reason might be that the EconBiz and PubMed datasets have very different data distributions, with the latter being more suitable for the type of problems for which Parabel was designed.

Regardless the finding that Parabel was not able to surpass the sample-based F1 scores of the deep learning techniques of [11], it is by no means a weak classifier, as in most cases (particularly in the PubMed dataset) had scores that are very close to them.

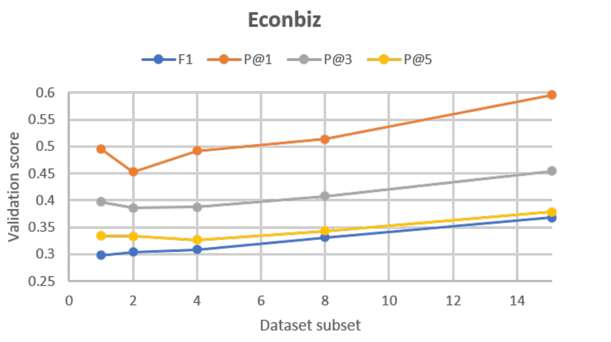

Figure 1: Graphical representation of the performance of Parabel (according to different evaluation metrics) as a function of the size of the subset of the EconBiz dataset used to train the model.

Also, it is important to take into consideration several aspects. First, hyper-parameters were set to default values used in the original implementation, whereas each neural network architecture employed by [11] was specifically tuned for each dataset. Second, a single label tree was built and used for computational reasons, which could have led to lower results because of the variability in the randomness of the Hierarchical K-Means Clustering algorithm (this type of variability is often handled by utilizing an ensemble of trees). Third, as has been mentioned before, the Parabel technique was designed to optimize ranking gain functions, not traditional multi-class or multi-label measures such as the sample-based F1. Although the slight modification of reducing Equation 5 to 12 most likely helped to improve the F1 score, it definitely did not compensate enough to make the algorithm work as if it was designed to optimize that metric from the start.

Speaking of the simplification done, it did not harm the performance of the P@k metric as much as it was initially thought despite Parabel's proven probabilistic model being violated. By looking at Table 3 and 4, it can be seen that the metrics are not bad, considering that they are very similar to the F1-score (slightly above it) in the EconBiz dataset and much higher in the PubMed dataset. From these results, it would seem that increasing the sample-based F1 score would result in an increase in P@k or vice-versa, suggesting that the deep learning methods of [11] might even be able to surpass Parabel in terms of ranking gain functions due to their slightly greater F1-score. This, however, is not entirely true as can be seen from Figure 1, where even though the sample-based F1 score kept continually growing as the dataset size increased, the P@k values decreased when going from T_1 to T_2 and T_4. In Figure 2, yet, this is not seen, since all of the metrics kept getting better.

In addition to the evaluation metrics, the time taken to train and test Parabel was recorded and written in Table 5 and 6. Interestingly, the values obtained were much higher than expected, taking for instance more than 1.5 hours to train on T_{all} of EconBiz and T_1 of PubMed. This was completely unforeseen as the training time reported on much bigger datasets in the original paper was even less. In the WikiLSHTC-325K dataset, for example, a single tree version of Parabel using the squared hinge loss took 0.29 hours

Table 2: Outcomes of the experiments for the sample-based F_1 metric. The values reported for the deep learning models (every row except the last one) are extracted directly from the paper of [11]. The best result for each subset (column) is highlighted in bold. Values marked with an * were obtained only by executing a single training-testing procedure instead of the 10-fold cross validation. Parabel models that were estimated to take more than 24 hours to train and test were not developed and their results are reported with a - symbol.

	EconBiz sample-based F_1 scores					PubMed sample-based F_1 scores				
	T_1	T_2	T_4	T_8	T_{all}	T_1	T_2	T_4	T_8	T_{all}
Base-MLP	**0.391**	**0.419**	**0.442**	0.451	0.472	**0.479**	**0.478**	0.475	0.465	0.485
MLP	0.357	0.396	0.432	**0.453**	**0.500**	0.449	0.456	0.464	0.465	0.504
CNN	0.364	0.382	0.400	0.407	0.426	0.438	0.437	0.431	0.419	0.440
LSTM	0.360	0.392	0.417	0.435	0.466	0.465	0.470	**0.477**	**0.481**	**0.515**
Parabel	0.298	0.304	0.308	0.331	0.368	0.428*	0.452*	0.465*	0.472*	-

Table 3: Results of the experiments using Parabel with the EconBiz subsets in terms of the P@k metric.

	EconBiz				
	T_1	T_2	T_4	T_8	T_{all}
P@1	0.496	0.453	0.492	0.514	0.596
P@3	0.397	0.386	0.387	0.408	0.454
P@5	0.334	0.333	0.326	0.343	0.378

Table 4: Results of the experiments using Parabel with the PubMed subsets in terms of the P@k metric. All these results were obtained from using a single training-test procedure instead of the 10-fold cross validation. A model for T_{all} was not built because it was estimated to take more than 24 hours to train and test.

	PubMed				
	T_1	T_2	T_4	T_8	T_{all}
P@1	0.794	0.814	0.837	0.841	-
P@3	0.710	0.737	0.749	0.752	-
P@5	0.648	0.669	0.683	0.686	-

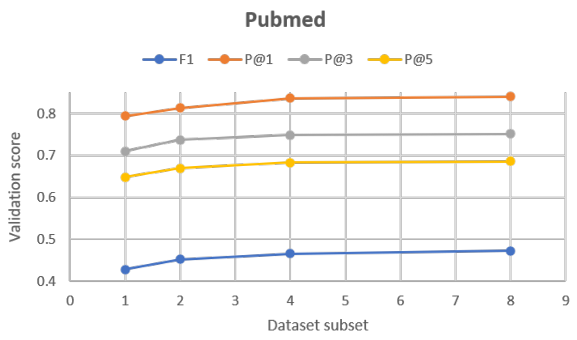

Figure 2: Graphical representation of the performance of Parabel (according to different evaluation metrics) as a function of the size of the subset of the PubMed dataset used to train the model.

to train, despite there being 325K labels and nearly 2 million data points (according to [3]). The main reasons that have been identified to explain the difference in speed are the following. First, the EconBiz and PubMed datasets, contrary to the well-known and utilized XMLC datasets, don't have many tail labels, as can be recognized from the fact that the average number of data points tagged with a given label is 819.1 and 5852.3 respectively (see Table 1), whereas datasets like WikiLSHTC-325K have only about 17.46 average points per label [3]. This causes the vectorization of the labels to become much more expensive (since more input vectors have to be averaged per label) as well as the process of learning the classifiers in the nodes of the tree (much more positive and negative instances are used per classifier), which can be seen by observing that the most demanding tasks in Tables 5 and 6 are those. The second factor that might have affected the efficiency of Parabel in the experiments of the present paper has to do with implementation details. In the original publication, the implementation was done using C++, while in this one the Python programming language was utilized. C++ is well-known to be much faster than Python because of its lower-level and compiled nature. In addition, because of how sparse matrices are handled in SciPy (one of Python's scientific libraries), access time might have been heavily deteriorated. Yet, further investigation needs to be made to ensure the validity of this claim.

6 CONCLUSION

In this paper, an implementation of the Parabel state-of-the-art XMLC technique was developed with the purpose of putting the deep learning methods of [11] in context with the rest of the modern XMLC approaches, as in their work, [11] utilized uncommon datasets and evaluation metrics that made it difficult to compare its results with other techniques. For achieving this, Parabel was trained and evaluated using the datasets of [11] and the sample-based F1 score. This would also serve to gauge the performance of Parabel in the context of digital libraries, where the presence of title-only data is much higher than that of full-text data. The results of the experiments undertaken showed that, even though not surpassing the deep learning methods, Parabel was able to achieve competitive performance in terms of the sample-based F1 score, despite being designed for optimizing ranking gain functions. This was possible partly because of a slight modification that simplified its probabilistic model.

Table 5: Average time in seconds taken to train an evaluate a Parabel model for each subset in the EconBiz dataset. The total time taken to do the 10-fold cross validation for each subset is also included.

		EconBiz				
		T_1	T_2	T_4	T_8	T_{all}
Train	Vectorize labels	34.984	59.787	117.079	266.266	549.239
	Build label tree	141.286	169.394	159.983	178.639	194.810
	Learn classifiers	251.765	480.132	963.076	2197.159	4858.927
	Total	428.034	709.313	1240.137	2642.065	5602.976
Evaluate		1223.146	1304.948	1330.059	1379.473	1457.869
Cross validation		16869.796	20749.451	26890.768	42908.904	76154.170

Table 6: Time in seconds taken to perform the single train and evaluation operation of the Parabel model for each subset of the PubMed dataset. No values are reported for T_{all} since a model was not built for it because it was estimated to take more than 24 houras to train and test.

		PubMed				
		T_1	T_2	T_4	T_8	T_{all}
Train	Vectorize labels	1096.306	2301.217	4956.094	10549.083	-
	Build label tree	1097.138	1363.982	1340.603	1422.995	-
	Learn classifiers	3543.010	7021.071	16030.433	36499.391	-
	Total	5736.453	10686.269	22327.130	48471.470	-
Evaluate		8659.403	8954.808	9028.4442	9266.895	-

Testing a version of Parable with its parameters tuned for each dataset and comparing the results with those of the deep learning models remains an activity to be done in future work. Similarly, an evaluation of the deep learning models in terms of their P@k values is desired for the future, so that a better understanding of the relationship between sample-based F1 and P@k scores can be derived.

REFERENCES

[1] Rahul Agrawal, Archit Gupta, Yashoteja Prabhu, and Manik Varma. 2013. Multi-label learning with millions of labels: recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web (WWW '13)*. Association for Computing Machinery, Rio de Janeiro, Brazil, 13–24. https://doi.org/10.1145/2488388.2488391

[2] Rohit Babbar and Bernhard Schölkopf. 2017. DiSMEC: Distributed Sparse Machines for Extreme Multi-Label Classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining (WSDM '17)*. Association for Computing Machinery, New York, NY, USA, 721–729. https://doi.org/10.1145/3018661.3018741 event-place: Cambridge, United Kingdom.

[3] K. Bhatia, K. Dahiya, H. Jain, A. Mittal, Y. Prabhu, and M. Varma. 2016. The extreme classification repository: Multi-label datasets and code. http://manikvarma.org/downloads/XC/XMLRepository.html

[4] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. 2015. Sparse Local Embeddings for Extreme Multi-label Classification. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 730–738. http://papers.nips.cc/paper/5969-sparse-local-embeddings-for-extreme-multi-label-classification.pdf

[5] Gordon V. Cormack and Thomas R. Lynam. 2006. Statistical precision of information retrieval evaluation. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '06)*. Association for Computing Machinery, Seattle, Washington, USA, 533–540. https://doi.org/10.1145/1148170.1148262

[6] Lukas Galke, Florian Mai, Alan Schelten, Dennis Brunsch, and Ansgar Scherp. 2017. Using Titles vs. Full-text as Source for Automated Semantic Document Annotation. *arXiv:1705.05311 [cs]* (Sept. 2017). http://arxiv.org/abs/1705.05311 arXiv: 1705.05311.

[7] Francesco Gargiulo, Stefano Silvestri, and Mario Ciampi. 2018. Deep convolution neural network for extreme multi-label text classification. 641–650. https://doi.org/10.5220/0006730506410650

[8] Himanshu Jain, Venkatesh Balasubramanian, Bhanu Chunduri, and Manik Varma. 2019. Slice: Scalable Linear Extreme Classifiers Trained on 100 Million Labels for Related Searches. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining (WSDM '19)*. Association for Computing Machinery, Melbourne VIC, Australia, 528–536. https://doi.org/10.1145/3289600.3290979

[9] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme Multi-label Loss Functions for Recommendation, Tagging, Ranking & Other Missing Label Applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, San Francisco, California, USA, 935–944. https://doi.org/10.1145/2939672.2939756

[10] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. 2017. 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 '17)*. Association for Computing Machinery, New York, NY, USA, 115–124. https://doi.org/10.1145/3077136.3080834 event-place: Shinjuku, Tokyo, Japan.

[11] Florian Mai, Lukas Galke, and Ansgar Scherp. 2018. Using Deep Learning for Title-Based Semantic Subject Indexing to Reach Competitive Performance to Full-Text. *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries - JCDL '18* (2018), 169–178. https://doi.org/10.1145/3197026.3197039 arXiv: 1801.06717.

[12] Jinseok Nam, Eneldo Loza Mencía, Hyunwoo J Kim, and Johannes Fürnkranz. 2017. Maximizing Subset Accuracy with Recurrent Neural Networks in Multi-label Classification. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5413–5423. http://papers.nips.cc/paper/7125-maximizing-subset-accuracy-with-recurrent-neural-networks-in-multi-label-classification.pdf

[13] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. 2018. Parabel: Partitioned Label Trees for Extreme Classification with Application to Dynamic Search Advertising. In *Proceedings of the 2018 World Wide Web Conference (WWW '18)*. International World Wide Web Conferences Steering Committee, Lyon, France, 993–1002. https://doi.org/10.1145/3178876.3185998

[14] Yashoteja Prabhu and Manik Varma. 2014. 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 '14)*. Association for Computing Machinery, New York, NY, USA, 263–272. https://doi.org/10.1145/2623330.2623651 event-place: New York, New York, USA.

[15] Yukihiro Tagami. 2017. AnnexML: Approximate Nearest Neighbor Search for Extreme Multi-label Classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. Association for Computing Machinery, Halifax, NS, Canada, 455–464. https:

//doi.org/10.1145/3097983.3097987

[16] Princeton University. 2010. About WordNet. https://wordnet.princeton.edu/

[17] Jason Weston, Ameesh Makadia, and Hector Yee. 2013. Label Partitioning for Sublinear Ranking. In *International Conference on Machine Learning*. http://www.thespermwhale.com/jaseweston/papers/label_partitioner.pdf

[18] Marek Wydmuch, Kalina Jasinska, Mikhail Kuznetsov, Róbert Busa-Fekete, and Krzysztof Dembczynski. 2018. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 6355–6366. http://papers.nips.cc/paper/7872-a-no-regret-generalization-of-hierarchical-softmax-to-extreme-multi-label-classification.pdf

[19] Ian E.H. Yen, Xiangru Huang, Wei Dai, Pradeep Ravikumar, Inderjit Dhillon, and Eric Xing. 2017. PPDsparse: A Parallel Primal-Dual Sparse Method for Extreme Classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. Association for Computing Machinery, Halifax, NS, Canada, 545–553. https://doi.org/10.1145/3097983.3098083

[20] Ian E. H. Yen, Xiangru Huang, Kai Zhong, Pradeep Ravikumar, and Inderjit S. Dhillon. 2016. PD-Sparse: A Primal and Dual Sparse Approach to Extreme Multiclass and Multilabel Classification. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48 (ICML'16)*. JMLR.org, 3069–3077. Place: New York, NY, USA.

[21] Ronghui You, Zihan Zhang, Ziye Wang, Suyang Dai, Hiroshi Mamitsuka, and Shanfeng Zhu. 2019. AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification. *arXiv:1811.01727 [cs]* (Nov. 2019). http://arxiv.org/abs/1811.01727 arXiv: 1811.01727.

[22] Min-Ling Zhang and Zhi-Hua Zhou. 2014. A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering* 26, 8 (Aug. 2014), 1819–1837. https://doi.org/10.1109/TKDE.2013.39 Conference Name: IEEE Transactions on Knowledge and Data Engineering.