Kalina Kobus

Efficient algorithms for extreme multi-label classification

Doctoral Dissertation

Submitted to the Discipline Council
of Information and Communication Technology
of Poznań University of Technology

Advisor: Krzysztof Dembczyński, Ph. D., Dr. Habil.

Poznań · 2020

Kalina Kobus

Efektywne algorytmy dla wieloetykietowej klasyfikacji ekstremalnej

Rozprawa doktorska

Przedłożono Radzie Dyscypliny
Informatyka Techniczna i Telekomunikacja
Politechniki Poznańskiej

Promotor: Krzysztof Dembczyński, Ph. D., Dr. Habil.

Poznań · 2020

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computing Science.

Kalina Kobus
Laboratory of Intelligent Decision Support Systems
Faculty of Computing and Telecommunications
Institute of Computing Science
Poznań University of Technology
kalina.jasinska@cs.put.poznan.pl

Typeset by the author in LaTeX.

This dissertation and associated materials can be downloaded from:
http://www.cs.put.poznan.pl/kjasinska

Institute of Computing Science
Poznań University of Technology
Piotrowo 2, 60–965 Poznań, Poland
http://www.cs.put.poznan.pl

The research presented in this dissertation was partially funded from National Science Centre funds granted under grant no. 2017/25/N/ST6/00747.

Computational experiments have been performed in Poznan Supercomputing and Networking Center.

Abstract

Extreme multi-label classification (XMLC) is a learning task of tagging instances with a small subset of relevant labels chosen from an extremely large pool of possible labels. Problems of this scale can be efficiently handled by organizing labels as a tree, like in hierarchical softmax used for multi-class problems. In this dissertation, we propose and thoroughly investigate probabilistic label trees (PLTs), suitable for estimating the conditional probabilities of labels, that can be treated as a generalization of hierarchical softmax for multi-label problems.

We introduce the PLT model and discuss training and inference procedures. We present a general training scheme given a label tree structure in advance. Furthermore, we consider a problem of training PLTs in a fully online setting, without any prior knowledge of training instances, their features, or labels. In this case, both node classifiers and the tree structure are trained online. We prove a specific equivalence between the fully online algorithm and an algorithm with a tree structure given in advance. We describe three inference procedures allowing prediction with PLTs suitable for various optimized performance metrics.

We prove the consistency of PLTs for a wide spectrum of performance metrics. To this end, we first analyze the performance metrics used in extreme multi-label classification and derive the Bayes classifiers for them. This way we show for which performance metrics the optimal decisions can be determined through the conditional probabilities of labels. Then, we upper-bound the regret with respect to these metrics by a function of surrogate-loss regrets of node classifiers.

We analyze the computational complexity of training and prediction procedures of PLTs and show that under certain assumptions it is sublinear in the number of labels. Finally, we discuss several implementations of PLTs, empirically evaluate their performance, and demonstrate their competitiveness to the state-of-the-art methods.

Acknowledgments

I would like to express sincere gratitude to my advisor, Dr. Krzysztof Dembczyński, for giving me the opportunity to pursue my scientific interests, sharing with me the ins and outs of doing research, and all his help throughout the years.

I would also like to extend my special thanks to Dr. Wojciech Kotłowski, who got me interested in machine learning in the first place, and also for many insightful comments and discussions. I also thank Dr. Nikos Karampatziakis, my internship mentor, for the great opportunity of doing research at Microsoft, and for his guidance, kindness, and help. I am truly grateful to Prof. Roman Słowiński and all the members of the Laboratory of Intelligent Decision Support Systems for their kindness and support, and also for showing me how the world-class research is carried out in Poznań. I thank my fellow PhD student Marek Wydmuch, who collaborated with me on much of this work. I also thank my other fellow PhD students, with whom I shared the toil of pursuing a PhD, for many good laughs and their unshakable belief in my research.

Last, but certainly not least, I wish to thank my family and friends. In particular, I wish to thank from the bottom of my heart my parents, Anita and Waldek, for their love and teaching me how to be curious, and my dearest husband Tadeusz, who walked this path before me, for his love, support and never-ending encouragement, without which this dissertation would not exist.

List of publications

Journal papers:

K. Jasinska-Kobus, M. Wydmuch, K. Dembczyński, M. Kuznetsov, and R. Busa-Fekete. Probabilistic label trees for extreme multi-label classification. *Journal of Machine Learning Research (in review)*, 2020a,

K. Jasinska, K. Dembczyński, and N. Karampatziakis. Extreme classification under limited space and time budget. *Schedae Informaticae*, 2017. doi: 10.4467/20838476SI.16.001.6182.

Conference papers:

K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. Extreme f-measure maximization using sparse probability estimates. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1435–1444, New York, USA, 2016. PMLR,

M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *Advances in Neural Information Processing Systems 31*, pages 6355–6366. Curran Associates, Inc., 2018,

K. Jasinska-Kobus, M. Wydmuch, D. Thiruvenkatachari, and K. Dembczyński. Online probabilistic label trees. *AISTATS 2020 (in review)*, 2020d.

Workshop and arxiv papers:

K. Jasinska and K. Dembczyński. Consistent label tree classifiers for extreme multi-label classification. In *The ICML Workshop on Extreme Classification*, 2015,

K. Jasinska and N. Karampatziakis. Log-time and log-space extreme classification. *CoRR*, abs/1611.01964, 2016,

K. Jasinska. Efficient exact batch prediction for label trees. In *Extreme Multilabel Classification for Social Media at The Web Conference*, 2018,

K. Jasinska and K. Dembczyński. Bayes optimal prediction for ndcg@k in extreme amulti-label classification. In *From Multiple Criteria Decision Aid to Preference Learning Workshop*, 2018,

R. Busa-Fekete, K. Dembczynski, A. Golovnev, K. Jasinska, M. Kuznetsov, M. Sviridenko, and C. Xu. On the computational complexity of the probabilistic label tree algorithms. *CoRR*, abs/1906.00294, 2019,

K. Jasinska-Kobus, M. Wydmuch, D. Thiruvenkatachari, and K. Dembczyński. Online probabilistic label trees. *CoRR*, abs/2007.04451, 2020c,

K. Jasinska-Kobus, M. Wydmuch, K. Dembczyński, M. Kuznetsov, and R. Busa-Fekete. Probabilistic label trees for extreme multi-label classification. *CoRR*, abs/2009.11218, 2020b.

Contents

1 Introduction

Machine learning is a broad and fast-growing field in between computer science, information theory, statistics, and even neuroscience. The development of machine learning focuses on not only the theoretical aspects but also the empirical results and applications. As it has proven to be successful in many applications, the number of such applications is growing, which leads to the emergence of new research directions. Machine learning is divided into three main subfields: supervised learning, unsupervised learning, and reinforcement learning.

Let us focus on classification, being the interest of supervised learning. In classification problems, the task is to assign, or predict, a label, or a number of them, to an instance based on its features as correctly as possible. The correctness of the prediction is evaluated using a loss function, which defines the solved task. To solve the classification problem, one usually trains a classifier, being a function assigning labels to instances based on their features. Such classifier is selected from a class of possible classifiers in a process of training. The training is done with a learning algorithm using a data set of training observations. The classifier is selected during training based on its loss, not necessarily the same as the task loss. The quality of the classifier is evaluated based on the task loss computed on its predictions, obtained in the process of prediction or inference, and observed ground truth labels. A desirable property of a learning algorithm is that given an infinitely growing sample it selects a classifier with task risk converging to the lowest possible task risk. We refer to such property as consistency.

In standard classification problems, the number of possible output labels is moderate. However, in many modern machine learning applications, the output space can be enormous, as it may contain even millions of labels. As an example, let us consider the task of tagging Wikipedia articles. In this case, each article is an observation, words appearing in its text are the instance's features, and the categories to which it is assigned are the labels. By creating a data set from the current content of Wikipedia, we end up with an enormous classification problem not only with millions of observations and features but also with over one million labels. Problems of this scale are often referred to as *extreme classification*. We distinguish extreme multi-class classification and extreme multi-label classification, depending on the number of relevant labels assigned to an instance: exactly one in the multi-class case, or any number in the multi-label case.

1.1 Extreme classification

Some notable examples of extreme classification problems are image and video annotation for multimedia search [Deng et al., 2011], tagging of text documents [Dekel and Shamir, 2010], online advertising [Beygelzimer et al., 2009a, Agrawal et al., 2013], recommendation of bid words for online ads [Prabhu and Varma, 2014], video recommendation [Weston et al., 2013], or prediction of the next word in a sentence [Mikolov et al., 2013]. To understand how to frame these problems as classification consider the following examples. In the case of tagging of text documents, as in the aforementioned Wikipedia example, the text of the document is the instance's features, and the tags are the labels. In the case of recommendation of bid words for online ads, the instance represents the landing page of the ad, and the labels are the relevant bid words. In the case of video recommendation, either user may be represented as an instance with videos being the labels, or the other way round. Notice that the number of labels in all these applications is extremely large.

In extreme multi-label classification, one usually needs to retrieve a small number of relevant labels or create a ranking of these labels. The quality of such predictions is usually measured with precision@k, being the fraction of relevant labels among k retrieved labels, or NDCG@k, being a ranking quality measure assigning rank-diminishing gains to relevant predictions. Other measures used in extreme classification include recall@k and macro-F_1-measure. However, because of the size of the output space, the computational efficiency of training and prediction algorithms plays a much more important role than in the case of non-extreme problems. In extreme classification, even algorithms scaling linearly with the number of labels may be prohibitively slow. As an example consider a standard naive solution to multi-label problems, training an independent model, e.g. a linear classifier, for each label individually. This approach, usually referred to as 1-VS-ALL, has time and space complexity linear in the number of labels. Such complexity is too costly in practical applications. Consider a problem with 10^6 labels and assume that a single binary classifier is trained in 1 second. In such a case, training 10^6 binary classifiers would take over 11 days. Mind you, that given a large number of training observations and features, the training time is likely to be much longer. Also, prediction with 1-VS-ALL, which requires evaluation of all 10^6 classifiers, is time-consuming. If you consider also the model size, under the assumption that there exist 10^5 features, you easily get a model of the size of the order of hundreds of GB. Hence there is a need for more advanced solutions characterized by both good predictive performance and sublinear complexity.

Besides computational challenges, extreme classification poses statistical ones, not present in standard learning problems. For example, for many labels, only a small number of training observations is usually available. These are so-called long-tail labels. In the ultimate case, we face the problem of zero-shot learning, where there are no training observations for some labels in the training set. Furthermore, training data might be of low quality as no one could go through all labels to verify whether they have been correctly assigned even to a single training observation. The training information is therefore usually obtained from implicit feedback and we often deal with the so-called counterfactual learning. However, in the dissertation, we solely focus on predictive quality and computational performance.

1.2 Related work

Extreme classification methods apply various techniques to reduce the training and prediction times and increase the predictive performance of the classifiers. Classically, the efficiency of prediction and good predictive performance is obtained by decision trees. However, the direct use of standard decision tree algorithms can be very costly [Agrawal et al., 2013] in extreme classification, mainly due to the need to compute the split criterion. The FASTXML algorithm [Prabhu and Varma, 2014] reduces the cost of computation of splits of the feature space by use of, instead of simple conditions, the linear classifiers resulting from alternating optimization of a multi-criteria objective. To improve accuracy, FASTXML builds an ensemble of many decision trees. The idea of FASTXML was further applied by PFASTREXML [Jain et al., 2016] and CRAFTML [Siblini et al., 2018], which modify the optimized criterion or method of creation of node classifiers, respectively. Online tree building method is considered in [Choromanska and Langford, 2015], which proposes LOMTREE for multi-class problems. Recently, Majzoubi and Choromanska [2019] introduced LDSM for multi-label problems, building the tree semi-online node-by-node. For a long time since the introduction, the decision-tree-based algorithms were the state-of-the-art of extreme multi-label classification, characterized by competitive predictive and computational performance.

Another approach adopted to solve multi-label classification are the methods based on embeddings. They reduce the original output space to a space with smaller dimensionality [Tai and Lin, 2012] and build regression models in this low-dimensional space. Methods adopting this approach differ in the compression and decompression schemes. Bhatia et al. [2015] have proposed sparse local embeddings (SLEEC) that use a k-nearest neighbor classifier in the embedding space to decompress the final prediction. This method performs better than decision-tree-based methods on some benchmark data sets, and worse on other ones. It is also characterized by large model sizes and long training and prediction times. AnnexML [Tagami, 2017] improved the efficiency of prediction compared to SLEEC by use of k-nearest neighbor graph of label vectors and obtained state-of-the-art predictive performance. Recently, Guo et al. [2019] introduced GLaS, taking advantage of co-occurrences of labels and fast maximum inner product search inference, and improving the predictive performance of the embedding-based methods.

The predictive performance of 1-vs-All has long been considered as hard to beat by less costly approaches. The smart 1-vs-All approaches do train a single binary classifier per label, but reduce the computational costs of naive methods by use of distributed computation and weight pruning (DiSMEC, Babbar and Schölkopf [2017]), exploring different optimization and prediction methods (PD-Sparse, Yen et al. [2016]; PPDSparse, Yen et al. [2017]), or both (ProXML, Babbar and Schölkopf [2019]). These methods obtain state-of-the-art predictive performance. However, their total training and prediction times are significantly longer than the times reached by other methods.

Jasinska et al. [2016] proposed the PLTs model, the first *label-tree*-based extreme multi-label classification approach. Label-tree-based methods allow to efficiently approximate the 1-vs-All, and hence are characterized by shorter training and prediction times. They differ significantly from decision trees, as in a label tree each tree leaf corresponds to exactly one label, not to a part of the feature space. Several PLT based models can be used in an ensemble, with an appropriate inference procedure, to increase the overall performance. The PLT-based approach was further adopted by extremeText Wydmuch et al. [2018], Parabel [Prabhu et al., 2018], Bonsai Tree [Babbar and Schölkopf, 2019], or AttentionXML [You et al., 2019], allowing these methods to reach state-of-the-art predictive performance and shorter prediction times than the decision-tree-based or smart 1-vs-All methods. An important aspect of label-tree-based methods is the choice of the label tree structure. Prabhu et al. [2018] have proposed an algorithm based on hierarchical balanced 2-means clustering on representations of labels, being the average features of observations with a given label. A variant of this method using k-means clustering is used by Wydmuch et al. [2018] and Khandagale et al. [2019]. Optimal structure with respect to the cost of training is considered by Busa-Fekete et al. [2019], while Zhuo et al. [2020] consider the optimal tree structure for approximate beam-search-based inference. In the case of multi-class label trees, Beygelzimer et al. [2009a] proposed an online tree building procedure for CPTs. fastText [Joulin et al., 2017] uses hierarchical softmax with a Huffman tree built on the label frequencies. Jernite et al. [2017] have introduced a Learned Tree algorithm, which combines hierarchical softmax with a hierarchical clustering that reassigns labels to paths in the tree in a semi-online manner. In this dissertation, we focus on PLTs and analyze them in detail.

Deep-learning-based methods have also been applied to extreme multi-label classification. These methods, when applied to textual data, use the raw text representations of data, while the previously described methods use sparse representations. Therefore we cannot directly compare their predictive performance to the results of previously described methods. Also, their training and prediction times are incomparable, as these methods use GPUs instead of CPUs. The first deep learning method applied to extreme multi-label classification is XML-CNN, introduced in [Liu et al., 2017]. It under-performs in terms of predictive performance and needs long training times. The already mentioned ATTENTIONXML [You et al., 2019] uses a shallow PLT and a multi-label attention mechanism. This way, ATTENTIONXML obtains state-of-the-art predictive performance. On the other hand, X-BERT [Chang et al., 2019], which also can be viewed as a PLT, uses a BERT-based [Devlin et al., 2018] probabilistic classifiers in the internal PLT nodes, and linear models in the leaves.

Many methods besides PLTs have been proposed to tackle extreme multi-label classification challenges. These methods usually allow to compute the predictions more efficiently than a naively applied 1-VS-ALL approach and frequently achieve good empirical results in terms of popular performance metrics. However, most of these methods are analyzed neither in terms of their statistical consistency with respect to the optimized performance metric nor in terms of their computational complexity.

1.3 Motivations

The motivation of the proposed probabilistic label trees model is a simple observation: it turns out that the optimal decisions, i.e. the Bayes classifiers, for precision@k and many other extreme multi-label classification metrics can be determined through the conditional probabilities of labels. Therefore estimating these conditional probabilities of labels and plugging-in a suitable decision rule might guarantee the statistical consistency of methods designed under this principle. From this perspective, the problem of extreme multi-label classification appears to be a problem of efficient estimation of the conditional probabilities of labels and efficient inference.

To estimate the conditional probabilities of labels efficiently, one can organize labels as a tree in which each label corresponds to one and only one path from the root to a leaf. Such methods have been applied to multi-class problems. A prominent example of such label tree model is hierarchical softmax (HSM) [Morin and Bengio, 2005], often used with neural networks to speed up computations in multi-class problems, for example, in natural language processing [Mikolov et al., 2013]. Interestingly, similar algorithms have been introduced independently in many different research fields. In statistics, they are known as nested dichotomies [Fox, 1997], in multi-class regression as conditional probability trees (CPTs) [Beygelzimer et al., 2009b], and in pattern recognition as multi-stage classifiers [Kurzynski, 1988]. However, probabilistic variants of label trees have not been used in extreme multi-label classifications before.

1.4 Aim and scope

Given the motivations presented above, the hypothesis of this dissertation is formulated as follows:

> There exists a class of statistically consistent learning algorithms for extreme multi-label classification whose computational complexity is sublinear with the number of labels.

The following points describe the main contribution of this dissertation.

1.4.1 Bayes classifiers

We review the Bayes classifiers for predictive performance metrics relevant to extreme multi-label classification. We prove that the Bayes prediction for precision@k is a set of k labels with the highest conditional probabilities. We also prove the forms of Bayes classifiers for DCG@k and NDCG@k. Following the literature, we include the results for generalized classification performance metrics [Kotłowski and Dembczyński, 2017], and recall@k [Menon et al., 2019]. This way, we show which metrics used in extreme multi-label classification can be optimized using conditional probabilities of labels.

1.4.2 Probabilistic label trees model

We introduce and exhaustively discuss the *probabilistic label trees* (PLTs) model. PLTs use a label tree to factorize conditional label probabilities by applying the chain rule along the paths in the tree. This way, they reduce the original multi-label problem to a number of binary classification (estimation) problems. From this point of view, PLTs follow the learning reductions framework [Beygelzimer et al., 2016]. PLTs use class probability estimators in all nodes of the tree to estimate the relevant factors being specific conditional probabilities. The estimate of the conditional probability of a label associated with a leaf is defined then as a product of the probability estimates on the path from the root to that leaf. For efficient prediction PLTs follow a proper tree search procedure.

We consider two learning setups of PLTs: incremental or batch training with label tree structure given in advance, and online with tree structure created simultaneously with the training. We prove a specific equivalence of those training methods. Prediction-wise, we consider three tree search procedures. The first one retrieves labels with the estimated conditional probability exceeding the given threshold. The second one, based on the uniform-cost search, retrieves the given number of highest scoring labels. The third, based on beam search, is an approximate method of retrieval of the highest scoring labels.

Let us relate the proposed approach to literature. Similar label tree approaches, HSM, nested dichotomies, or CPTs, have already been used for solving multi-class problems. PLTs can be treated as a proper generalization of these approaches to the problem of multi-label estimation of conditional probabilities of single labels. There also exist other label tree approaches which mainly differ from PLTs and other methods mentioned above in that they do not use the probabilistic framework. The most similar to PLTs method out of these is HOMER introduced by Tsoumakas et al. [2008]. Other, less similar are Tournament-based or filter trees, which have been considered in [Beygelzimer et al., 2009b] and [Li and Lin, 2014] to solve respectively multi-class and multi-label problems. Another example are label embedding trees introduced in [Bengio et al., 2010]. None of these methods, however, has been thoroughly tested in the extreme classification setting, but initial studies suggest that they neither scale to problems of this scale nor perform competitively to PLTs. Interestingly, probabilistic classifier chains [Dembczyński et al., 2010], suited for multi-label problems under the subset $0/1$-loss, can also be interpreted as a specific label tree, but with paths corresponding to subsets of labels. This way the size of the tree is exponential in the number of labels, while the size of the PLT tree grows linearly with the number of labels.

Since the first articles [Jasinska and Dembczyński, 2015, Jasinska et al., 2016] the PLT model has been used in many other algorithms such as PARABEL [Prabhu et al., 2018], BONSAI TREE [Khandagale et al., 2019], EXTREMETEXT [Wydmuch et al., 2018], and ATTENTIONXML [You et al., 2019], allowing these methods to reach state-of-the-art predictive performance and shorter prediction times than other methods. This emphasizes the importance of the proposed PLTs model in the area of extreme multi-label classification.

1.4.3 Consistency and regret bounds

The theoretical results regarding PLTs concern their consistency for a wide spectrum of performance metrics, outlined before. We show the consistency following the learning reductions framework [Beygelzimer et al., 2016]. We upper-bound the regret of a PLT with respect to these performance metrics by a function of surrogate regrets of node classifiers. To this end, we first bound the L_1 estimation error of conditional probability of a single label by L_1 errors of node classifiers on a path from the root to the label's leaf. This result is similar to the result obtained by [Beygelzimer et al., 2009a] for multi-class problems, however, presented proofs seem to be simpler and the bounds tighter. Then we apply the bound of the L_1 error of node classifiers by their regret with respect to a strongly proper composite loss [Agarwal, 2014]. This way we connect the regret of the node classifiers with the L_1 error of estimation of conditional probabilities of labels. These results are the building blocks for all other results regarding metrics for which the optimal predictions may be determined through the conditional probabilities of labels.

Using the bound for the L_1 estimation error, we show the regret bounds for generalized performance metrics of the linear-fractional form. This class of functions contains among other Hamming loss, micro-, and macro- F-measure. The bounds for PLTs are based on results obtained by [Kotłowski and Dembczyński, 2017] for the 1-VS-ALL approach. Then we consider precision@k. By using the definition of the regret and the bound for the L_1 estimation error, we obtain conditional and unconditional bounds for precision@k. This result combined with the priority-queue-based inference shows that PLTs are well-tailored for this metric. The last metric considered is DCG@k. For this metric we report analogous results to precision@k.

We also study the relation of PLTs to HSM. We show that the former is indeed a proper generalization of the latter to multi-label problems. Namely, for multi-class distribution PLTs reduce to the HSM model. Moreover, we show that other popular generalization of HSM to multi-label problems using the pick-one-label heuristic, used for example in FASTTEXT [Joulin et al., 2017] and LEARNED TREE [Jernite et al., 2017], is not consistent in terms of L_1 estimation error and precision@k.

1.4.4 Computational complexity of training and prediction

We analyze the computational complexity of training and prediction with PLTs. We show that training of node classifiers (with a tree structure given in advance) can be done in logarithmic time under additional assumptions concerning the tree structure and the maximum number of labels per training observation. Moreover, with additional assumptions on estimates of conditional label probabilities also the prediction time is logarithmic in the number of labels. This result is not trivial as a prediction method needs to use a tree search algorithm that, in the worse case, explores the entire tree leading to linear complexity.

1.4.5 Empirical evaluation

Besides our theoretical findings, we discuss several existing implementations of the general PLT scheme, such as XMLC-PLT [Jasinska et al., 2016], PLT-VW[1], PARABEL [Prabhu et al., 2018], BONSAI TREE [Khandagale et al., 2019], EXTREMETEXT [Wydmuch et al., 2018], ATTENTIONXML [You et al., 2019], and NAPKINXC [Jasinska-Kobus et al., 2020a]. We thoroughly analyze the implementation possibilities regarding the representation of features and models or methods of training and prediction.

In the experiments, we mostly focus on NAPKINXC and PARABEL. We thoroughly analyze different instances of PLTs and relate their results to other state-of-the-art decision-tree-based and 1-VS-ALL-based algorithms. Our experiments indicate that PLTs are very competitive, reaching the best precision@1 on the majority of benchmark data sets, being at the same time even a thousand times faster in training and prediction than the 1-VS-ALL-based approaches.

[1]https://github.com/VowpalWabbit/vowpal_wabbit

1.5 Personal contribution to development of PLTs

Below we overview the papers about PLTs which contribute to the dissertation. The initial work about PLTs [Jasinska and Dembczyński, 2015] was presented at the Extreme Classification Workshop at ICML 2015. This paper proposes the PLTs models, besides another label-tree-based model named BR-TREE. The PLTs model, with simple training and prediction procedures, was later published in [Jasinska et al., 2016]. This paper considers the use of PLTs for optimization of precision@k, using uniform-cost search based prediction, and macro-F_1-measure, using threshold-based prediction with tuned thresholds. Then, a batch uniform-cost search prediction procedure was proposed in [Jasinska, 2018]. Next, [Wydmuch et al., 2018], considered L_1 estimation error bound and precision@k regret bound. The computational complexity of PLTs was analyzed in [Busa-Fekete et al., 2019]. This dissertation includes some of the results from this paper concerning the computational cost bounds, while the paper includes many other results. Online PLTs were proposed in 2016, and finally published in [Jasinska-Kobus et al., 2020c,d]. Most of the theoretical results presented in this dissertation are available in [Jasinska-Kobus et al., 2020a], summarizing the long-term work on PLTs. This dissertation also includes results independent of PLTs. [Jasinska and Karampatziakis, 2016] describes another extreme classification algorithm named LTLS. The results related to NDCG@k are presented in [Jasinska and Dembczyński, 2018].

1.6 Outline

In Chapter 2 we formally state the extreme multi-label classification problem and present the framework used in this dissertation. In Chapter 3 we discuss the forms of the Bayes classifiers for selected predictive performance metrics and narrow down the set of metrics of interest to those, for which the optimal predictions can be determined through the conditional probabilities of labels. Those first two chapters outline the setting of our further work. The next chapters describe and thoroughly analyze the probabilistic label trees. Chapter 4 defines the PLT model and discusses training and prediction procedures. Chapter 5 contains the theoretical analysis of PLTs in terms of the regret bounds. In Chapter 6 we analyze the computational complexity of training and prediction with PLTs. In Chapter 7 we present online probabilistic label trees and prove their properties. In Chapter 8 we discuss the implementation choices of the PLT model. The experimental results are presented in Chapter 9. In Chapter 10, we briefly describe other algorithms proposed for extreme multi-label classification, discuss other results related to PLTs, briefly analyze other challenges of extreme multi-label classification, and outline open research directions related to PLTs. Chapter 11 summarizes the dissertation.

2
Theoretical background

In this chapter, we formally define binary and multi-label classification problems and characterize the extreme setting of the latter. We focus on the statistical decision theory point of view and define the risk of a classifier, the Bayes optimal classifier, and the regret of a classifier. By analyzing the $0/1$ loss in binary classification, we show the importance of estimation of conditional probabilities of labels and introduce the L_1 estimation error. Then, we recall the strongly proper composite loss functions that allow estimating the conditional probabilities and to bound the L_1 error. We define the consistency of classifiers in the context of surrogate losses and learning reductions. To this end, we use the example of 1-VS-ALL reduction. Finally, we outline the goal of the dissertation by briefly commenting on the properties and limitations of 1-VS-ALL.

2.1 Binary classification

Let $\mathcal{X}$ denote an instance space. In a *binary classification* problem, we assume that an instance $\boldsymbol{x} \in \mathcal{X}$ is associated with a binary label, $y \in \{-1, 1\}$, corresponding to a negative and positive class, respectively. We assume that the observations $(\boldsymbol{x}, y)$ are generated independently and identically according to a probability distribution $\mathbf{P}(\boldsymbol{x}, y)$ defined on $\mathcal{X} \times \{-1, 1\}$. We denote the positive class probability $\mathbf{P}(y = 1 \,|\, \boldsymbol{x})$ by $\eta(\boldsymbol{x})$, and its estimate by $\hat{\eta}(\boldsymbol{x})$.

For the sake of generality, we assume that a *binary classifier* $h(\boldsymbol{x})$ is a function from class $\mathcal{H}^1 : \mathcal{X} \to \mathbb{R}$. The predictive performance of a classifier is measured using a (*task*) *loss* ℓ. The goal of solving a classification problem is to deliver prediction $\hat{y} = h(\boldsymbol{x})$, being the best under a given task loss for an instance $\boldsymbol{x}$. The problem of binary classification can be then defined as finding classifier $h(\boldsymbol{x}) \in \mathcal{H}^1$ which minimizes the *expected loss*, or *risk*:

$$R_\ell(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbf{P}(\boldsymbol{x},y)}(\ell(y, h(\boldsymbol{x}))\,.$$

The risk can be expressed as an expectation of *conditional risks*

$$R_\ell(h|\boldsymbol{x}) = \mathbb{E}_{y\sim\mathbf{P}(y|\boldsymbol{x})}(\ell(y, h(\boldsymbol{x}))$$

as

$$R_\ell(h) = \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})} R_\ell(h|\boldsymbol{x})\,.$$

The optimal classifier, the so-called *Bayes classifier*, for a given loss function ℓ is:

$$h^*_\ell = \arg\min_h R_\ell(h)\,.$$

The *regret* of a classifier h with respect to ℓ is defined as:

$$\mathrm{reg}_\ell(h) = R_\ell(h) - R_\ell(h^*_\ell) = R_\ell(h) - R^*_\ell\,.$$

The regret quantifies the suboptimality of h compared to the optimal classifier h^*_ℓ. The goal could be then defined as finding h with a small regret, ideally equal to zero.

The canonical loss function for binary classification is the *zero-one* $(0/1)$ loss:

$$\ell_{0/1}(y, h(\boldsymbol{x})) = [\![h(\boldsymbol{x}) \neq y]\!]$$

We will analyze this loss following [Tewari and Bartlett, 2013, Bartlett, 2014]. The $0/1$ risk is:

$$\begin{aligned}
R_{0/1}(h) &= \mathbb{E}_{(\boldsymbol{x},y)\sim\mathbf{P}(\boldsymbol{x},y)}(\ell_{0/1}(y, h(\boldsymbol{x}))\\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\mathbb{E}_{y\sim\mathbf{P}(y\,|\,\boldsymbol{x})}(\ell_{0/1}(y, h(\boldsymbol{x}))\\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[\ell_{0/1}(1, h(\boldsymbol{x}))\eta(\boldsymbol{x}) + \ell_{0/1}(-1, h(\boldsymbol{x}))(1-\eta(\boldsymbol{x}))]\\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![1 \neq h(\boldsymbol{x})]\!]\eta(\boldsymbol{x}) + [\![-1 \neq h(\boldsymbol{x})]\!](1-\eta(\boldsymbol{x}))]\\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![1 \neq h(\boldsymbol{x})]\!](2\eta(\boldsymbol{x}) - 1) + 1 - \eta(\boldsymbol{x})]
\end{aligned}$$

Therefore, the Bayes optimal classifier with respect to the zero-one loss $h^*_{0/1}$ is given by:

$$h^*_{0/1}(\boldsymbol{x}) = \begin{cases} -1 & \text{if } \eta(\boldsymbol{x}) < 0.5 \\ 1 & \text{if } \eta(\boldsymbol{x}) > 0.5 \end{cases} \tag{2.1}$$

where ties $\mathbf{P}(y = 1\,|\,\boldsymbol{x}) = \mathbf{P}(y = -1\,|\,\boldsymbol{x})$ are broken randomly. The $\ell_{0/1}$-regret of a classifier h is

$$\mathrm{reg}_{0/1}(h) = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{X}}[[\![h(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]|2\eta(\boldsymbol{x}) - 1|]$$

This result can be derived from the $0/1$ risk as follows:

$$\begin{aligned}
\mathrm{reg}_{0/1}(h) &= R_{0/1}(h) - R_{0/1}(h^*_{0/1}) \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![1 \neq h(\boldsymbol{x})]\!](2\eta(\boldsymbol{x}) - 1) + 1 - \eta(\boldsymbol{x})] \\
&\quad - \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![1 \neq h^*_{0/1}(\boldsymbol{x})]\!](2\eta(\boldsymbol{x}) - 1) + 1 - \eta(\boldsymbol{x})] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![1 \neq h(\boldsymbol{x})]\!](2\eta(\boldsymbol{x}) - 1) - [\![1 \neq h^*_{0/1}(\boldsymbol{x})]\!](2\eta(\boldsymbol{x}) - 1)] \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[[\![h(\boldsymbol{x}) \neq h^*_{0/1}]\!]|2\eta(\boldsymbol{x}) - 1|]
\end{aligned}$$

The above analysis suggests that the estimation of $\eta(\boldsymbol{x})$ is one of the right approaches to solving binary classification under the $0/1$ loss. The estimate $\hat{\eta}(\boldsymbol{x})$ can be plugged-in to the rule defined by the Bayes optimal classifier (2.1) to obtain the $0/1$ predictions. Let us denote such a classifier as $h_{\hat{\eta}}$. Below, we show formally that such an approach upper bounds the $0/1$ regret.

When estimating $\eta(\boldsymbol{x})$ one usually considers the L_1 *estimation error*:

$$|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})| \,. \tag{2.2}$$

To show the relation to the $0/1$ regret, let us write:

$$\begin{aligned}
\mathrm{reg}_{0/1}(h_{\hat{\eta}}) &= R_{0/1}(h) - R_{0/1}(h^*_{\ell}) \\
&= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{X}}[[\![h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]|2\eta(\boldsymbol{x}) - 1|] \\
&= 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{X}}\left[[\![h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]\left|\eta(\boldsymbol{x}) - \frac{1}{2}\right|\right].
\end{aligned}$$

Notice that if $h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})$, then $\eta(\boldsymbol{x})$ and $\hat{\eta}(\boldsymbol{x})$ must lie on the opposite sides of $\frac{1}{2}$, which means that $\left|\eta(\boldsymbol{x}) - \frac{1}{2}\right| + \left|\hat{\eta}(\boldsymbol{x}) - \frac{1}{2}\right|$ equals $|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|$ in such case. Thus, we have:

$$\begin{aligned}
[\![h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]\left|\eta(\boldsymbol{x}) - \frac{1}{2}\right| &\leq [\![h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]\left(\left|\eta(\boldsymbol{x}) - \frac{1}{2}\right| + \left|\hat{\eta}(\boldsymbol{x}) - \frac{1}{2}\right|\right) \\
&= [\![h_{\hat{\eta}}(\boldsymbol{x}) \neq h^*_{0/1}(\boldsymbol{x})]\!]\,|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})| \\
&\leq |\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})| \,.
\end{aligned}$$

We finally get:

$$\mathrm{reg}_{0/1}(h_{\hat{\eta}}) \leq 2\mathbb{E}_{\boldsymbol{x}\sim\mathcal{X}}[|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|] \,.$$

2.2 Strongly proper composite losses

We recall the concept of *strongly proper composite losses* [Agarwal, 2014, Kotłowski and Dembczyński, 2017] for binary classification. These losses are of special interest in the case of the problem of estimation of $\eta(\boldsymbol{x})$ under the L_1 estimation error. Let us first define a class probability estimation (CPE) loss as a function

$\ell : \{-1, 1\} \times [0, 1] \mapsto \mathbb{R}_+$. Its conditional risk is given by

$$R_\ell(\hat{\eta} \,|\, \boldsymbol{x}) = \eta(\boldsymbol{x})\ell(1, \hat{\eta}(\boldsymbol{x})) + (1 - \eta(\boldsymbol{x}))\ell(-1, \hat{\eta}(\boldsymbol{x}))\,.$$

A CPE loss is proper if for any $\eta(\boldsymbol{x}) \in [0, 1]$, $\eta(\boldsymbol{x}) \in \arg\min_{\hat{\eta}} R_\ell(\hat{\eta} \,|\, \boldsymbol{x})$. Since it is often more convenient for prediction algorithms to work with a real-valued scoring function, $f : \mathcal{X} \mapsto \mathbb{R}$, than with an estimate bounded to interval $[0, 1]$, we transform $\hat{\eta}(\boldsymbol{x})$ using a strictly increasing (and therefore invertible) link function $\psi : [0, 1] \to \mathbb{R}$, that is, $f(\boldsymbol{x}) = \psi(\hat{\eta}(\boldsymbol{x}))$.

We then consider a composite loss function $\ell_c : \{-1, 1\} \times \mathbb{R} \mapsto \mathbb{R}_+$ defined via a CPE loss as

$$\ell_c(y, f(\boldsymbol{x})) = \ell(y, \psi^{-1}(f(\boldsymbol{x})))\,.$$

The regret of f in terms of a loss function ℓ_c at point $\boldsymbol{x}$ is defined as:

$$\mathrm{reg}_{\ell_c}(f \,|\, \boldsymbol{x}) = R_\ell(\psi^{-1}(f) \,|\, \boldsymbol{x}) - R^*_\ell(\boldsymbol{x})\,,$$

where $R^*_\ell(\boldsymbol{x})$ is the minimum expected loss at point $\boldsymbol{x}$, achievable by $f^*(\boldsymbol{x}) = \psi(\eta(\boldsymbol{x}))$. The above statement shows that the ℓ_c-regret of f is the ℓ-regret of $\psi^{-1}(f)$.

We say a loss function ℓ_c is λ-strongly proper composite loss, if for any $\eta(\boldsymbol{x}), \psi^{-1}(f(\boldsymbol{x})) \in [0, 1]$:

$$\left|\eta(\boldsymbol{x}) - \psi^{-1}(f(\boldsymbol{x}))\right| \leq \sqrt{\frac{2}{\lambda}}\sqrt{\mathrm{reg}_{\ell_c}(f \,|\, \boldsymbol{x})}\,. \tag{2.3}$$

It can be shown under mild regularity assumptions that ℓ_c is λ-strongly proper composite if and only if its corresponding CPE loss is proper and function $H_\ell(\eta) = R_\ell(\eta \,|\, \boldsymbol{x})$ is λ-strongly concave, that is, $\left|\frac{d^2 H_\ell(\eta)}{d^2\eta}\right| \geq \lambda$. As ψ is invertible, then we can treat $\psi^{-1}(f(\boldsymbol{x}))$ as the estimate of the class probability $\hat{\eta}(\boldsymbol{x})$. This means that the L_1 estimation error $|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|$ is bounded by the ℓ_c-regret of the scoring function f.

As an example consider the logistic loss, being a CPE loss:

$$\ell(y, \hat{\eta}(\boldsymbol{x}))) = -[\![y = 1]\!] \log \hat{\eta}(\boldsymbol{x}) - [\![y = -1]\!] \log(1 = \hat{\eta}(\boldsymbol{x}))\,. \tag{2.4}$$

Its conditional risk is given by:

$$R_\ell(\hat{\eta} \,|\, \boldsymbol{x}) = \eta(\boldsymbol{x}) \log(\hat{\eta}(\boldsymbol{x})) - (1 - \eta(\boldsymbol{x})) \log(1 - \hat{\eta}(\boldsymbol{x}))\,,$$

being the cross-entropy between $\eta(\boldsymbol{x})$ and $\hat{\eta}(\boldsymbol{x})$. The conditional ℓ-regret is the binary Kullback-Leibler divergence between $\eta(\boldsymbol{x})$ and $\hat{\eta}(\boldsymbol{x})$:

$$\mathrm{reg}_\ell(\eta(\boldsymbol{x}), \hat{\eta}(\boldsymbol{x})) = \eta(\boldsymbol{x}) \log \frac{\eta(\boldsymbol{x})}{\hat{\eta}(\boldsymbol{x})} + (1 - \eta(\boldsymbol{x})) \log \frac{1 - \eta(\boldsymbol{x})}{1 - \hat{\eta}(\boldsymbol{x})}\,.$$

Since $H_\ell(\eta(\boldsymbol{x})) = R_\ell(\eta \,|\, \boldsymbol{x})$ is the binary entropy function, and $\left|\frac{d^2 H_\ell(\eta)}{d^2\eta}\right| = \frac{1}{\eta(\boldsymbol{x})(1-\eta(\boldsymbol{x}))} \geq 4$, c is 4-strongly proper loss. Using the logit link function

$\psi(\hat{\eta}(\boldsymbol{x})) = \log \frac{\hat{\eta}(\boldsymbol{x})}{1-\hat{\eta}(\boldsymbol{x})}$, we end up with the logistic loss function defined for scoring function f:

$$\ell_c(y, f) = \log(1 + e^{-yf}).$$

Other examples of commonly used strongly proper composite losses are squared loss, squared hinge loss, and exponential loss. Notice that the standard hinge loss does not belong to this class of losses. Table 2.1 summarizes the formulas and the link functions of these losses.

loss function	squared	logistic	exponential
$l_c(y, f)$	$(y-f)^2$	$\log(1+e^{-fy})$	e^{-fy}
$l(1, \hat{\eta})$	$4(1-\hat{\eta})^2$	$-\log \hat{\eta}$	$\sqrt{\frac{1-\hat{\eta}}{\hat{\eta}}}$
$l(-1, \hat{\eta})$	$4\hat{\eta}^2$	$-\log(1-\hat{\eta})$	$\sqrt{\frac{\hat{\eta}}{1-\hat{\eta}}}$
$\psi(\hat{\eta})$	$2\hat{\eta}-1$	$\log \frac{\hat{\eta}}{1-\hat{\eta}}$	$\frac{1}{2} \log \frac{\hat{\eta}}{1-\hat{\eta}}$

Table 2.1: Loss formula, CPE loss formula, and link function for three popular strongly proper composite losses: squared, logistic and exponential losses.

2.3 Multi-label classification

Let us define the multi-label classification problem, which is the main focus of this dissertation. Let $\mathcal{X}$ denote an instance space, and let $\mathcal{L} = [m]$ be a finite set of m class labels. [1] We assume that an instance $\boldsymbol{x} \in \mathcal{X}$ is associated with a subset of labels $\mathcal{L}_{\boldsymbol{x}} \subseteq \mathcal{L}$ (the subset can be empty); this subset is often called the set of *relevant* or *positive* labels, while the complement $\mathcal{L} \backslash \mathcal{L}_{\boldsymbol{x}}$ is considered as *irrelevant* or *negative* for $\boldsymbol{x}$. We identify the set $\mathcal{L}_{\boldsymbol{x}}$ of relevant labels with the binary vector $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)$, in which $y_j = 1 \Leftrightarrow j \in \mathcal{L}_{\boldsymbol{x}}$.[2] By $\mathcal{Y} = \{0,1\}^m$ we denote the set of all possible label vectors. We assume that observations $(\boldsymbol{x}, \boldsymbol{y})$ are generated independently and identically according to a probability distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$ defined on $\mathcal{X} \times \mathcal{Y}$. The conditional (given $\boldsymbol{x}$) joint probability of a label vector is $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$, and the *conditional probability of a label j* is the following marginalized value:

$$\mathbf{P}(y_j = 1 \,|\, \boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}} y_j \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \,.$$

We use $\eta_j(\boldsymbol{x})$ to denote $\mathbf{P}(y_j = 1 \,|\, \boldsymbol{x})$.

The above definition concerns not only multi-label classification but also multi-class (when $\|\boldsymbol{y}\|_1 = 1$) and k-sparse multi-label (when $\|\boldsymbol{y}\|_1 \leq k$) problems

[1]We use $[n]$ to denote the set of integers from 1 to n.

[2] Notice that in the definition of multi-label classification to denote the negative label in $\boldsymbol{y}$ we use 0, instead of -1 which was used in the sections about binary classification. Such notation is more convenient for multi-label problems. One can be transformed to another: let $y_{0/1} \in \{0,1\}$ and $y_{-1/1} \in \{-1,1\}$ be the two different notations, then $y_{-1/1} = 2y_{0/1} - 1$.

as special cases. [3] In the case of *extreme multi-label classification*, we assume m to be a large number (for example $\geq 10^5$), and k to be much smaller than m, $k \ll m$.

The problem of extreme multi-label classification can be defined as finding a classifier $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_m(\boldsymbol{x}))$, from a function class $\mathcal{H}^m : \mathcal{X} \to \mathbb{R}^m$, that minimizes the risk:

$$R_\ell(\boldsymbol{h}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathbf{P}(\boldsymbol{x},\boldsymbol{y})}(\ell(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) \,,$$

where $\ell(\boldsymbol{y}, \hat{\boldsymbol{y}})$ is a multi-label (*task*) *loss*. The optimal classifier for a given loss function ℓ is:

$$\boldsymbol{h}^*_\ell = \arg\min_{\boldsymbol{h}} R_\ell(\boldsymbol{h}) \,,$$

and the regret of a classifier $\boldsymbol{h}$ with respect to ℓ is:

$$\mathrm{reg}_\ell(\boldsymbol{h}) = R_\ell(\boldsymbol{h}) - R_\ell(\boldsymbol{h}^*_\ell) = R_\ell(\boldsymbol{h}) - R^*_\ell \,.$$

Consider a generalization of the binary 0/1-loss to multi-class classification:

$$\ell_{0/1}(\boldsymbol{y}, h(\boldsymbol{x})) = [\![h(\boldsymbol{x}) \neq j]\!] \,,$$

where, for simplicity of notation, j denotes the only label being positive in $\boldsymbol{y}$, and the classifier predicts only one label, $h(\boldsymbol{x}) \in \mathcal{L}$. The optimal strategy, which is a generalization of the result for binary classification, for this loss is the label with the highest conditional probability:

$$h^*_{0/1}(\boldsymbol{x}) = \arg\max_{j\in\mathcal{L}} \eta_j(\boldsymbol{x}) \,. \tag{2.5}$$

This simple multi-class loss is not applicable in the general multi-label case.

A basic loss function used in multi-label classification is Hamming loss:

$$\ell_H(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \frac{1}{m}\sum_{j=1}^{m} [\![y_j \neq h_j(\boldsymbol{x})]\!] \,. \tag{2.6}$$

It is easy to notice that it is the average label-wise (binary) 0/1-loss:

$$\begin{aligned} \ell_H(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) &= \frac{1}{m}\sum_{j=1}^{m} [\![y_j \neq h_j(\boldsymbol{x})]\!] \\ &= \frac{1}{m}\sum_{j=1}^{m} \ell_{0/1}(y_j, h_j(\boldsymbol{x})) \,. \end{aligned}$$

One can easily observe that the optimal strategy for this loss is [Dembczyński et al., 2012]:

$$h^*_j(\boldsymbol{x}) = [\![\eta_j(\boldsymbol{x}) > 0.5]\!] \,. \tag{2.7}$$

This optimal strategy results then from the aforementioned result from binary classification.

[3]We use $\|\boldsymbol{x}\|_1$ to denote the L_1 norm of x.

Another standard loss used in multi-label classification is the subset-0/1-loss

$$\ell_{0/1}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = [\![\boldsymbol{y} \neq \boldsymbol{h}(\boldsymbol{x})]\!] \,.$$

This loss is analogous to the multi-class 0/1-loss, however is defined on a new multi-class problem with 2^m labels that correspond to all possible label vectors. Notice that for multi-class data this loss is the same as the multi-class 0/1 loss, as there are only m possible label vectors, each with only one positive label. The Bayes optimal classifies for the subset-0/1-loss is [Dembczyński et al., 2010]:

$$\boldsymbol{h}^*(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{y} \in \{0,1\}^m} \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \,.$$

Notice that two different losses: Hamming loss and subset-0/1-loss, both relevant to multi-label classification, have different optimal strategies.

2.4 Statistical consistency

Multi-label classification problem can be approached in different ways. We describe two approaches: first we follow the reasoning related to *surrogate losses*, and then describe an alternative one, based on *learning reductions*. Consider multi-label classification under Hamming loss. Let us start with finding a surrogate loss for this task loss. Similarly to the binary classification case, we are interested in multi-label classifiers that estimate conditional probabilities of labels, $\eta_j = \mathbf{P}(y_j = 1|\boldsymbol{x})$, $j \in \mathcal{L}$, as accurately as possible, that is, with possibly small L_1 estimation error,

$$|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \,, \tag{2.8}$$

where $\hat{\eta}_j(\boldsymbol{x})$ is an estimate of $\eta_j(\boldsymbol{x})$. This allows us to plug-in the optimal strategy rule (2.7) and minimize the task loss, i.e. Hamming loss, by minimizing the label-wise L_1 estimation errors. To obtain such estimates $\hat{\eta}_j(\boldsymbol{x})$ with $\boldsymbol{h}(\boldsymbol{x})$ one can use the label-wise logistic loss, sometimes referred to as binary cross-entropy:

$$\ell_{\log}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \sum_{j=1}^{m} \ell_{\log}(y_j, h_j(\boldsymbol{x})) = \sum_{j=1}^{m} \left(y_j \log(h_j(\boldsymbol{x})) + (1 - y_j)\log(1 - h_j(\boldsymbol{x}))\right) \,.$$

As we have discussed in Section 2.2, logistic loss (2.4) is a strongly proper CPE loss. Therefore, by minimizing the binary cross-entropy loss, we minimize the L_1 estimation error of conditional probabilities of labels, and ultimately the Hamming loss. This way, by minimizing a surrogate loss, we minimize the task loss.

Another way to solve a multi-label classification problem is reducing it to a number of less complex problems, solving the less complex problems, and then translating the solutions to these problems to a solution to the original problem. Such a strategy for solving problems is called learning reductions [Beygelzimer et al., 2016]. Under Hamming loss such a strategy may be adopted as follows: instead of solving a multi-label problem with m labels, solve m binary classification problems. From (2.6) you can easily notice, that for this reduction of the multi-label problem under Hamming loss, it is sufficient to treat each binary problem as an independent binary problem under 0/1-loss. Then, each binary problem shall be solved according to the rule (2.1), and the solution to the multi-label problem is simply aggregating all the predicted labels. Such an approach corresponds to the vanilla 1-VS-ALL method. This reduction translates an optimal solution to the base problems, into an optimal solution of the original problem. However, not every reduction has such a desirable property. Another simple reduction is a multi-class variant of 1-VS-ALL, suited for the multi-class 0/1-loss. It also solves m binary problems, but instead of predicting all the labels for which the estimated probability is higher than 0.5, it predicts the label with the highest probability, according to the rule (2.5). Notice that this variant of 1-VS-ALL would not produce an optimal solution to a multi-label problem under the Hamming loss. In this simple multi-label example, both approaches: minimizing the binary cross-entropy loss and reducing the problem via the 1-VS-ALL reduction, may seem to be the same, however, it is not always the case, as there exist many other learning reductions.

We described methods of solving a task problem by solving a surrogate problem, or a *proxy* problem, instead. The proxy problems we discussed, the use of binary cross-entropy loss or 1-VS-ALL learning reduction, allowed us to optimally solve the task problems. Let us state this formally following [Gao and Zhi-Hua, 2013]. Let the original problem be defined by the task loss ℓ. As in the example, instead of the original problem, we solve a proxy problem. The proxy problem can be defined by either a surrogate loss or a reduction. Let $\widetilde{\ell}$ denote the *proxy loss*, used instead of ℓ to select classifier $\boldsymbol{h}$.

Definition 2.1. *We say that a proxy loss $\widetilde{\ell}$ is consistent with the task loss ℓ when the following holds:*

$$\mathrm{reg}_{\widetilde{\ell}}(\boldsymbol{h}) \to 0 \Rightarrow \mathrm{reg}_{\ell}(\boldsymbol{h}) \to 0$$

In other words, for a consistent proxy loss $\widetilde{\ell}$, the set of optimal solutions with respect to $\widetilde{\ell}$ is a subset of the Bayes optimal classifiers for ℓ.

The reduction of a multi-label problem with m labels to m binary problems corresponds to the vanilla 1-VS-ALL approach. Such reduction is consistent with respect to a problem of estimation of conditional probabilities of labels under L_1 estimation error, as each conditional probability of a label can be estimated independently. However, 1-VS-ALL applied naively is costly in the extreme setting, as training and prediction complexity is linear in the number of labels. In this work, we discuss an alternative consistent reduction based on label trees, which estimates the conditional probabilities of labels with a competitive accuracy, but in a much more efficient way.

3 Multi-label classification metrics

In this chapter, we overview the most popular extreme multi-label classification metrics. We focus on the forms of the optimal classifier with respect to each of these metrics. We are interested in understanding for which metrics the optimal decisions can be determined through the conditional probabilities of labels. The results presented in this chapter include both the contributions of this dissertation and the results from the literature. Precision@k and the pick-one-label heuristic were analyzed in [Wydmuch et al., 2018, Jasinska-Kobus et al., 2020b], while NDCG@k in [Jasinska and Dembczyński, 2018]. The results related to the generalized classification performance metrics follow [Kotłowski and Dembczyński, 2017], and the form of the Bayes classifier for recall@k is based on [Menon et al., 2019]. Interestingly, the work of Menon et al. [2019] regarding recall@k was inspired by observations about precision@k from [Wydmuch et al., 2018].

3.1 Generalized classification performance metrics

We start with a wide family of generalized classification performance metrics, which includes metrics often used to report performance of multi-label classifiers, such as (weighted) Hamming loss, AM metric, as well as macro- and micro-averaged F_β-measure. As already proven [Koyejo et al., 2015, Kotłowski and Dembczyński, 2017] the optimal strategy for these metrics is to find a threshold on the conditional probability $\eta_j(\boldsymbol{x})$ for each label $j \in \mathcal{L}$. Such thresholds should be either set to predefined values, if they are known from theory (for example, this is 0.5 for Hamming loss), or tuned on a validation set, if their optimal value depends on the optimum of the metric. Here we discuss those results in more detail.

The discussed family of metrics can be defined as linear-fractional functions of label-wise false positives

$$\mathrm{FP}_j(h_j) = \mathbf{P}(h_j(\boldsymbol{x}) = 1 \wedge y_j = 0)\,,$$

and false negatives

$$\mathrm{FN}_j(h_j) = \mathbf{P}(h_j(\boldsymbol{x}) = 0 \wedge y_j = 1)\,.$$

To define generalized classification performance metrics, consider a linear-fractional function Ψ of the following generic form:

$$\Psi(\mathrm{FP}, \mathrm{FN}) = \frac{a_0 + a_1\mathrm{FP} + a_2\mathrm{FN}}{b_0 + b_1\mathrm{FP} + b_2\mathrm{FN}}\,, \tag{3.1}$$

being non-increasing in its arguments. We assume that there exists $\gamma > 0$, such that

$$b_0 + b_1\mathrm{FP} + b_2\mathrm{FN} \geq \gamma, \tag{3.2}$$

that is, the denominator of Ψ is positive and bounded away from 0. Let us notice that other parameterizations of these functions are possible, and, for example, Koyejo et al. [2015] parametrize Ψ also with true positives TP and true negatives TN. However, the parametrization using FP and FN is sufficient, as TP and TN are known for a given problem given FP and FN as they sum up to the fraction of positives and fraction of negatives that are fixed for a given problem.

A *macro-averaged generalized classification performance metric* $\Psi_{\mathrm{macro}(\boldsymbol{x})}$ is defined as:

$$\Psi_{\mathrm{macro}}(\boldsymbol{h}) = \frac{1}{m}\sum_{j=1}^{m}\Psi(h_j) = \frac{1}{m}\sum_{j=1}^{m}\Psi(\mathrm{FP}_j(h_j), \mathrm{FN}_j(h_j)). \tag{3.3}$$

It computes an average performance over single labels. Micro-averaged performance metrics compute first the average false positives and false negatives:

$$\bar{\mathrm{FP}}(\boldsymbol{h}) = \frac{1}{m}\sum_{i=1}^{m}\mathrm{FP}_j(h_j)\,, \quad \bar{\mathrm{FN}}(\boldsymbol{h}) = \frac{1}{m}\sum_{j=1}^{m}\mathrm{FN}_j(h_j)\,.$$

Then, a *micro-averaged generalized classification performance metric* $\Psi_{\mathrm{micro}(\boldsymbol{x})}$ is defined as:

$$\Psi_{\mathrm{micro}}(\boldsymbol{h}) = \Psi(\bar{\mathrm{FP}}(\boldsymbol{h}), \bar{\mathrm{FN}}(\boldsymbol{h}))\,. \tag{3.4}$$

The optimal classifier from class $\mathcal{H}^m_{\mathrm{bin}} = \mathcal{X} \to \{0,1\}^m$ for the generalized performance metrics has the generic form:

$$\boldsymbol{h}^*_\Psi(\boldsymbol{x}) = \boldsymbol{h}^*_{\boldsymbol{\alpha}^*_\Psi}(\boldsymbol{x}) = \left(h^*_{1,\alpha^*_{\Psi,1}}(\boldsymbol{x}), h^*_{2,\alpha^*_{\Psi,2}}(\boldsymbol{x}), \ldots, h^*_{m,\alpha^*_{\Psi,m}}(\boldsymbol{x})\right)\,, \tag{3.5}$$

where

$$h^*_{j,\alpha^*_{\Psi,j}}(\boldsymbol{x}) = [\![\eta_j(\boldsymbol{x}) > \alpha^*_{\Psi,j}]\!]\,,$$

with Ψ-optimal thresholds $\boldsymbol{\alpha}^*_\Psi$:

$$\boldsymbol{\alpha}^*_\Psi = (\alpha^*_{\Psi,1}, \alpha^*_{\Psi,2}, \ldots, \alpha^*_{\Psi,m}) \in [0,1]^m\,.$$

In other words, for each metric there is an optimal vector $\boldsymbol{\alpha}^*_\Psi$ of thresholds defined over the conditional probabilities of labels, $\eta_j(\boldsymbol{x})$, for all $j \in \mathcal{L}$. The values of its elements are given by the following expression [Koyejo et al., 2015, Kotłowski and Dembczyński, 2017]:

$$\alpha^*_\Psi = \frac{\Psi(\mathrm{FP}^*, \mathrm{FN}^*) b_1 - a_1}{\Psi(\mathrm{FP}^*, \mathrm{FN}^*)(b_1 + b_2) - (a_1 + a_2)} \,,$$

where FP^*, FN^* are arguments maximizing either $\Psi(\mathrm{FP}_j(h_j), \mathrm{FN}_j(h_j))$, for each label $j \in \mathcal{L}$ separately, in case of a macro-averaged metric, or $\Psi(\bar{\mathrm{FP}}(\boldsymbol{h}), \bar{\mathrm{FN}}(\boldsymbol{h}))$ in case of a micro-averaged metric. This shows that for macro-averaged metrics the threshold may be different for each label, while for micro-averaged metrics there is one common threshold shared by all labels. Therefore, we denote the optimal classifier for a macro-average metric by $\boldsymbol{h}^*_{\boldsymbol{\alpha}^*_\Psi}$, while for a micro-average metric by $\boldsymbol{h}^*_{\alpha^*_\Psi}$.

The thresholds, in general, depend on the optimal value of Ψ, which makes their value to be unknown beforehand. Only for metrics for which $b_1 = b_2 = 0$, the thresholds can be computed a priori. This is the case of Hamming loss, its cost-sensitive variant (in which there are different costs of false positive and false negative predictions), or the AM metric. In other cases, thresholds have to be found on a validation set. For some metrics, such as the micro- and macro-F measure, this can be performed efficiently even in the XMLC setting, as only positive and positively predicted labels are needed to tune thresholds [Jasinska et al., 2016]. This can be even obtained using an online procedure [Busa-Fekete et al., 2015, Jasinska et al., 2016].

We present the form of $\Psi(\mathrm{FP}, \mathrm{FN})$ and α^*_Ψ for some popular generalized performance metrics in Table 3.1. We use there P to denote $\mathbf{P}(y_j = 1)$, for macro-averaging, and $\frac{1}{m}\sum_{j=1}^m \mathbf{P}(y_j = 1)$, for micro-averaging. Remark that this is a constant not depending on $\boldsymbol{h}$. All these metrics can be used with macro- and micro-averaging. Remark, however, that for Hamming loss both variants lead to the same form. A similar table can be found in [Kotłowski and Dembczyński, 2017].

Metric	$\Psi(\mathrm{FP}, \mathrm{FN})$	α^*_Ψ
Hamming loss	$1 - \mathrm{FP} - \mathrm{FN}$	0.5
F-measure	$\frac{(1+\beta^2)(P-\mathrm{FN})}{(1+\beta^2)P-\mathrm{FN}+\mathrm{FP}}$	$\Psi(\mathrm{FP}^*, \mathrm{FN}^*)/2$
Jaccard similarity	$\frac{P-\mathrm{FN}}{P+\mathrm{FP}}$	$\frac{\Psi(\mathrm{FP}^*,\mathrm{FN}^*)}{\Psi(\mathrm{FP}^*,\mathrm{FN}^*)+1}$
AM	$\frac{2P(1-P)-P\mathrm{FP}-(1-P)\mathrm{FN}}{2P(1-P)}$	P

Table 3.1: Examples of popular generalized performance metrics, with their form of $\Psi(\mathrm{FP}, \mathrm{FN})$ and α^*_Ψ. P denotes $\mathbf{P}(y_j = 1)$, for macro-averaging, or $\frac{1}{m}\sum_{j=1}^m \mathbf{P}(y_j = 1)$, for micro-averaging.

3.2 Precision@k and recall@k

Let us now analyze precision@k and recall@k, which are of a different nature than the metrics discussed above, as they are defined for exactly k predicted labels. Therefore, consider a class of functions $\mathcal{H}^m_{@k} = \{\boldsymbol{h} \in \mathcal{H}^m_{\text{bin}} : \sum_{j=1}^m h_j(\boldsymbol{x}) = k \,, \forall \boldsymbol{x} \in \mathcal{X}\}$, that is, functions that predict exactly k labels, $k \leq m$. Then precision@k for $\boldsymbol{h}_{@k} \in \mathcal{H}^m_{@k}$ is defined as:

$$p@k(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x})) = \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} [\![y_j = 1]\!] \,,$$

while recall@k is defined as:

$$r@k(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x})) = \frac{1}{|\mathcal{L}_{\boldsymbol{x}}|} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} [\![y_j = 1]\!] \,,$$

where $\hat{\mathcal{L}}_{\boldsymbol{x}} = \{j \in \mathcal{L} : h_j(\boldsymbol{x}) = 1\}$ is a set of k labels predicted by classifier $\boldsymbol{h}_{@k}$ for $\boldsymbol{x}$.

In order to define the relevant conditional risks it is more convenient to consider the precision@k loss, $\ell_{p@k} = 1 - p@k(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x}))$, and recall@$k$ loss, $\ell_{r@k} = 1 - r@k(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x}))$. The conditional risk for $\ell_{p@k}$ is then:

$$\begin{aligned}
R_{p@k}(\boldsymbol{h}_{@k} \,|\, \boldsymbol{x}) &= \mathbb{E}_{\boldsymbol{y}} \ell_{p@k}(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x})) \\
&= 1 - \sum_{\boldsymbol{y} \in \mathcal{Y}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} [\![y_j = 1]\!] \\
&= 1 - \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \sum_{\boldsymbol{y} \in \mathcal{Y}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) [\![y_j = 1]\!] \\
&= 1 - \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \eta_j(\boldsymbol{x}) \,.
\end{aligned}$$

From the above, it is easy to notice that the optimal strategy for precision@k,

$$\boldsymbol{h}^*_{p@k}(\boldsymbol{x}) = \left(h^*_{1,p@k}, h^*_{2,p@k}, \ldots, h^*_{m,p@k} \right) \,,$$

is to predict k labels $\mathcal{L}^*_{\boldsymbol{x}}$ with the highest conditional probabilities $\eta_j(\boldsymbol{x})$,

$$h^*_{j,p@k} = \begin{cases} 1 \,, \; j \in \mathcal{L}^*_{\boldsymbol{x},p@k} \\ 0 \,, \; \text{otherwise} \end{cases} \,, \tag{3.6}$$

where $\mathcal{L}^*_{\boldsymbol{x},p@k}$ is the set of k labels with highest conditional probabilities $\eta_j(\boldsymbol{x})$ with ties solved in any way.

The optimal strategy for recall@k is in general different. To show it, we consider the conditional risk for recall@k:

$$\begin{aligned}
R_{r@k}(\boldsymbol{h}_{@k} \,|\, \boldsymbol{x}) &= \mathbb{E}_{\boldsymbol{y}} \ell_{r@k}(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x})) \\
&= 1 - \sum_{\boldsymbol{y} \in \mathcal{Y}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) \frac{1}{|\boldsymbol{y}|} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} [\![y_j = 1]\!] \\
&= 1 - \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \sum_{\boldsymbol{y} \in \mathcal{Y}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) [\![y_j = 1]\!] \frac{1}{|\boldsymbol{y}|} \\
&= 1 - \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \eta'_j(\boldsymbol{x}) \,,
\end{aligned}$$

where

$$\eta'_j(\boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} \frac{1}{|\boldsymbol{y}|} \mathbf{P}(\boldsymbol{y} | \boldsymbol{x}).$$

This risk is minimized by

$$\boldsymbol{h}^*_{r@k}(\boldsymbol{x}) = \left(h^*_{1,r@k}, h^*_{2,r@k}, \ldots, h^*_{m,r@k} \right) \,,$$

predicting k labels $\mathcal{L}^*_{\boldsymbol{x},r@k}$ with the highest values of $\eta'_j(\boldsymbol{x})$ [Menon et al., 2019],

$$h^*_{j,r@k} = \begin{cases} 1 \,, j \in \mathcal{L}^*_{\boldsymbol{x},r@k} \\ 0 \,, \text{otherwise} \end{cases} \,, \tag{3.7}$$

where $\mathcal{L}^*_{\boldsymbol{x},r@k}$ is the set of k labels with highest values of $\eta'_j(\boldsymbol{x})$ with ties solved in any way.

3.2.1 Pick-one-label heuristic

Interestingly, the $\eta'(\boldsymbol{x})$ values are estimated using the *pick-one-label heuristic*, which is sometimes used to transform multi-label classification problems to multi-class classification problems [Joulin et al., 2017, Jernite et al., 2017]. This heuristic randomly picks one of the positive labels from a given training observation. The resulting observation is then treated as a multi-class observation. Since the probability of picking a label j from $\boldsymbol{y}$ is equal to $y_j / \sum_{j'=1}^m y_{j'}$, the pick-one-label heuristic maps the multi-label distribution to a multi-class distribution in the following way:

$$\eta'_j(\boldsymbol{x}) = \mathbf{P}'(y_j = 1 \,|\, \boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}} \frac{y_j}{\sum_{j'=1}^m y_{j'}} \mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x}) \,, j \in \mathcal{L} \,. \tag{3.8}$$

The resulting $\eta'_j(\boldsymbol{x})$ form a multi-class distribution as the probabilities sum up to 1, and differ from conditional probabilities $\eta_j(\boldsymbol{x})$.

3.2.2 Comparison of the optimal classifiers

Optimal classifiers $\boldsymbol{h}^*_{p@k}$ and $\boldsymbol{h}^*_{r@k}$ are in general different solutions, selecting labels based on different marginalized values. As we show below, the pick-one-label heuristic is also not consistent for precision@k in general. To show the general inconsistency of the pick-one-label heuristic for precision@k, one needs to show that $\boldsymbol{h}^*_{r@k}$ has a non-zero regret in terms of precision@k.

Proposition 3.1. *A classifier $\boldsymbol{h}_{@k} \in \mathcal{H}^m_{@k}$ predicting k labels with highest $\eta'_j(\boldsymbol{x})$, $j \in \mathcal{L}$, has in general a non-zero regret in terms of precision@k.*

Proof. We prove the proposition by giving a simple counterexample. Consider the following conditional distribution for some $\boldsymbol{x}$:

$$\mathbf{P}(\boldsymbol{y} = (1,0,0)\,|\,\boldsymbol{x}) = 0.1\,, \quad \mathbf{P}(\boldsymbol{y} = (1,1,0)\,|\,\boldsymbol{x}) = 0.5\,, \quad \mathbf{P}(\boldsymbol{y} = (0,0,1)\,|\,\boldsymbol{x}) = 0.4\,.$$

The optimal top 1 prediction for this instance is label 1, since the conditional probabilities are $\eta_1(\boldsymbol{x}) = 0.6, \eta_2(\boldsymbol{x}) = 0.5, \eta_3(\boldsymbol{x}) = 0.4$. However, the pick-one-label heuristic will transform the original distribution to the following one: $\eta'_1(\boldsymbol{x}) = 0.35, \eta'_2(\boldsymbol{x}) = 0.25, \eta'_3(\boldsymbol{x}) = 0.4$. The predicted top label will be then label 3, giving the regret of 0.2 for precision@1. □

Interestingly, the situation changes when the labels are conditionally independent, that is, if for each $\boldsymbol{y} \in \mathcal{Y}$

$$\mathbf{P}(\boldsymbol{y}\,|\,\boldsymbol{x}) = \prod_{j=1}^{m} \mathbf{P}(y_i\,|\,\boldsymbol{x})\,. \tag{3.9}$$

Theorem 3.2. *Given conditionally independent labels, $\eta_j(\boldsymbol{x})$ and $\eta'_j(\boldsymbol{x})$, $j \in \mathcal{L}$ induce the same order of labels.*

We show here only a sketch of the proof, and give the full proof in Appendix A.1. It is enough to show that in the case of conditionally independent labels the pick-one-label heuristic does not change the order of conditional probabilities. Let labels i and j be so that $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x})$. Then in the summation over all $\boldsymbol{y}$s in (3.8), we are interested in four different subsets of $\mathcal{Y}$, $S^{u,w}_{i,j} = \{\boldsymbol{y} \in \mathcal{Y} : y_i = u \wedge y_j = w\}$, where $u, w \in \{0,1\}$. Remark that during mapping none of $\boldsymbol{y} \in S^{0,0}_{i,j}$ plays any role, and for each $\boldsymbol{y} \in S^{1,1}_{i,j}$, the value of

$$y_t / (\sum_{t'=1}^{m} y_{t'}) \cdot \mathbf{P}(\boldsymbol{y}\,|\,\boldsymbol{x})\,,$$

for $t \in \{i,j\}$, is the same for both y_i and y_j. Now, let $\boldsymbol{y}' \in S^{1,0}_{i,j}$ and $\boldsymbol{y}'' \in S^{0,1}_{i,j}$ be the same on all elements except the i-th and the j-th one. Then, because of the label independence and the assumption that $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x})$, we have $\mathbf{P}(\boldsymbol{y}'\,|\,\boldsymbol{x}) \geq \mathbf{P}(\boldsymbol{y}''\,|\,\boldsymbol{x})$. Therefore, after mapping (3.8) we obtain $\eta'_i(\boldsymbol{x}) \geq \eta'_j(\boldsymbol{x})$.

Corollary 3.3. *Given conditionally independent labels, $\boldsymbol{h}_{r@k} \in \mathcal{H}^m_{@k}$ predicting k labels with highest $\eta'_j(\boldsymbol{x})$, $j \in \mathcal{L}$ has zero regret in terms of the precision@k loss.*

Corollary 3.4. *Given conditionally independent labels, $\boldsymbol{h}_{p@k} \in \mathcal{H}^m_{@k}$ predicting k labels with highest $\eta_j(\boldsymbol{x})$, $j \in \mathcal{L}$ has zero regret in terms of the recall@k loss.*

3.3 DCG@k and NDCG@k

Let us now move on to discounted @k metrics. However they may seem similar to precision@k and recall@k, the used discounting of gains has significant implications. Unlike in the analysis of precision@k and recall@k, here we need to consider rankings of k labels, instead of just sets of $\hat{\mathcal{L}}_{\boldsymbol{x}}$ of k labels. Let s be a vector of m label-wise scores. A *ranking of labels* according to scores s, $\boldsymbol{\pi}(\mathrm{s})$, is a permutation of these labels, such that $\pi_r(\mathrm{s})$ is the r-th ranked label according to decreasing scores s_j with ties solved any way. We denote the vector of conditional probabilities of labels with $\boldsymbol{\eta}(\boldsymbol{x})$, and a ranking of labels according to these probabilities with $\boldsymbol{\pi}(\boldsymbol{\eta}(\boldsymbol{x}))$.

Consider the class of functions $\mathcal{H}^m$, and a classifier $\boldsymbol{h} \in \mathcal{H}^m$, and a ranking of labels $\boldsymbol{\pi}(\boldsymbol{h}(\boldsymbol{x}))$ according to the scores of classifier $\boldsymbol{h}$. By modifying the definition of precision@k by summing over the ranking of labels $\boldsymbol{\pi}(\boldsymbol{h}(\boldsymbol{x}))$ up to rank k, adding a rank-dependent discounted gain factor

$$g(r) = \frac{1}{\log_2(r+1)}\,,$$

and removing the normalizing factor $\frac{1}{k}$ we get the discounted cumulative gain:

$$DCG@k(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = \sum_{r=1}^{k} [\![y_{\pi_r(\boldsymbol{h})}]\!] g(r)$$

Normalized discounted cumulative gain, NDCG@k, differs from DCG@k by a normalization term. It is normalized by the highest possible DCG@k for a given label vector $\boldsymbol{y}$. This best possible, or the ideal, DCG@k is

$$IDCG@k(\boldsymbol{y}) = \sum_{r=1}^{\min(k, ||\boldsymbol{y}||_1)} g(r).$$

By normalizing DCG@k by this factor we get the normalized discounted cumulative gain

$$NDCG@k(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) = N_k(\boldsymbol{y}) DCG@k(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})), \tag{3.10}$$

where, for simplicity of notation, we define

$$N_k(\boldsymbol{y}) = IDCG@k^{-1}(\boldsymbol{y})\,.$$

Notice that Ravikumar et al. [2011] give a general formula for NDCG by parametrizing it with a monotonically increasing function of the relevance judgments defining gains, and a rank-dependent monotonically increasing discount function. However, we use the common variant of this metric, being used in extreme multi-label classification.

In order to define conditional risks we consider DCG@k loss $\ell_{D@k} = -DCG@k(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x}))$ and NDCG@k loss $\ell_{D@k} = -NDCG@k(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x}))$. The conditional risk for DCG@k is:

$$\begin{aligned} R_{D@k}(\boldsymbol{h}\,|\,\boldsymbol{x}) &= \mathbb{E}\ell_{D@k}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) \\ &= -\sum_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \sum_{r=1}^{k} [\![y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))}]\!] g(r) \\ &= -\sum_{r=1}^{k} g(r) \sum_{\boldsymbol{y}\in\mathcal{Y}} [\![y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))}]\!] \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \,. \end{aligned}$$

Notice that $\sum_{\boldsymbol{y}\in\mathcal{Y}} [\![y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))}]\!] \mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$ is the probability that the r-ranked label is positive. Moreover, the discounted gains $g(r)$ diminish with increasing r. Therefore, the optimal strategy for DCG@k is

$$\boldsymbol{h}^*_{D@k}(\boldsymbol{x}) = \left(h^*_{1,D@k}, h^*_{2,D@k}, \ldots, h^*_{m,D@k}\right) \,,$$

such that $\boldsymbol{\pi}(\boldsymbol{h}^*_{D@k}(\boldsymbol{x}))$ ranks labels (up to rank k) in the order of conditional probabilities of labels, as the ranking $\boldsymbol{\pi}(\boldsymbol{\eta}(\boldsymbol{x}))$, with ties solved in any way. This also could be accomplished by selecting k labels and sorting them accordingly. The optimal strategy for DCG@k is similar to the one for precision@k, as it considers the conditional probabilities of labels, however, it requires additionally sorting by conditional probabilities to obtain the optimal order. Notice that $\boldsymbol{h}^*_{D@k}(\boldsymbol{x})$ allows for an optimal solution with respect to precision@k. Interestingly, for $k = m$, it also coincides with the optimal solution for the unnormalized rank loss [Dembczyński et al., 2010].

The conditional risk for NDCG@k is:

$$\begin{aligned} R_{N@k}(\boldsymbol{h}\,|\,\boldsymbol{x}) &= \mathbb{E}\ell_{N@k}(\boldsymbol{y}, \boldsymbol{h}(\boldsymbol{x})) \\ &= -\sum_{\boldsymbol{y}\in\mathcal{Y}} \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) N_k(\boldsymbol{y}) \sum_{r=1}^{k} g(r) y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))} \\ &= -\sum_{r=1}^{k} g(r) \sum_{\boldsymbol{y}\in\mathcal{Y}} y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))} \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) N_k(\boldsymbol{y}) \\ &= -\sum_{r=1}^{k} g(r) \sum_{\boldsymbol{y}: y_{\pi_r(\boldsymbol{h}(\boldsymbol{x}))}=1} \Delta_j(k, \boldsymbol{x}) \,, \end{aligned}$$

where $\Delta_j(k, \boldsymbol{x})$ is the following marginalized value:

$$\Delta_j(k, \boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} N_k(\boldsymbol{y}) \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \,. \tag{3.11}$$

The optimal strategy for NDCG@k is of form

$$\boldsymbol{h}^*_{N@k}(\boldsymbol{x}) = \left(h^*_{1,N@k}, h^*_{2,N@k}, \ldots, h^*_{m,N@k}\right) \,,$$

such that $\boldsymbol{\pi}(\boldsymbol{h}^*_{N@k}(\boldsymbol{x}))$ ranks labels (up to rank k) in the order of $\boldsymbol{\pi}(\boldsymbol{\Delta}(k, \boldsymbol{x}))$, $\boldsymbol{\Delta}(k, \boldsymbol{x}) = (\Delta_1(k, \boldsymbol{x}), \Delta_2(k, \boldsymbol{x}), \ldots, \Delta_m(k, \boldsymbol{x}))$. Interestingly, the optimal strategy for NDCG@1 is the same as the optimal strategy for DCG@k, as

$$\Delta_j(1, \boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} N_1(\boldsymbol{y}) \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) = \eta_j(\boldsymbol{x}) \, .$$

However, in general $\boldsymbol{\pi}(\boldsymbol{h}^*_{N@k}(\boldsymbol{x}))$ and $\boldsymbol{\pi}(\boldsymbol{h}^*_{D@k}(\boldsymbol{x}))$ are different rankings.

3.3.1 NDCG at different ranks

Unlike the optimal decisions for DCG@k, the optimal decisions for NDCG@k depend on k. More precisely, for a fixed conditional distribution, NDCG@k and NDCG@l, $k < l$, may be optimized by rankings with different labels on top k positions.

Theorem 3.5. *Let $k < l \leq m$. Optimal rankings for NDCG@k and NDCG@l in general may differ on first k ranks.*

Proof. This can be proved by a counterexample for $k = 1$ and $l = 2$. Consider the following conditional distribution for some $\boldsymbol{x}$:

$$\mathbf{P}(\boldsymbol{y} = (1, 0, 0) \,|\, \boldsymbol{x}) = 0.4 \, , \quad \mathbf{P}(\boldsymbol{y} = (0, 1, 1) \,|\, \boldsymbol{x}) = 0.6 \, .$$

The optimal decision for NDCG@1 is a ranking consisting of one label, being either y_2 or y_3, as their marginal probabilities are the same, and $N_1(1) = 1$. To state the optimal decision for NDCG@2, compute the Δ values:

$$\Delta_1(2, \boldsymbol{x}) = N_2((1, 0, 0)) \cdot 0.4 = 0.4 \, ,$$

$$\Delta_2(2, \boldsymbol{x}) = \Delta_3(2, \boldsymbol{x}) = N_2((0, 1, 1)) \cdot 0.6 \approx 0.3679 < 0.4 \, .$$

The first element of the ranking optimal for NDCG@2 is y_1, not y_2 or y_3. Therefore, it is different from the optimal decision for NDCG@1. □

3.3.2 Comparison of the optimal classifiers

However $\boldsymbol{h}^*_{D@k}$ and $\boldsymbol{h}^*_{N@k}$ in general indicate different rankings, if labels are conditionally independent, see (3.9), $\boldsymbol{h}^*_{D@k}$ and $\boldsymbol{h}^*_{N@k}$ indicate the same rankings. This result is analogous to the one for $\boldsymbol{h}^*_{p@k}$ and $\boldsymbol{h}^*_{r@k}$. The proof is included in Appendix A.1.

Theorem 3.6. *Given conditionally independent labels, $\eta_j(\boldsymbol{x})$ and $\Delta_j(k, \boldsymbol{x})$, $j \in \mathcal{L}$ induce the same order of labels.*

Corollary 3.7. *Given conditionally independent labels, a classifier $\boldsymbol{h}^*_{D@k}$ delivers optimal solution for NDCG@k.*

Corollary 3.8. *Given conditionally independent labels, a classifier $\boldsymbol{h}^*_{N@k}$ delivers optimal solution for DCG@k.*

3.4 Conclusions

As we have shown, the optimal strategy for precision@k, the most popular measure used in extreme multi-label classification, is determined through the conditional probabilities of labels. Also metrics such as the DCG@k, Hamming loss, and the micro- and macro F-measure are optimized through the conditional probabilities of labels. This suggests that estimating the conditional probability of labels, and then predicting the labels according to the form of the Bayes optimal classifier, is a consistent strategy for solving extreme multi-label classification under those metrics. In this dissertation, we propose multi-label classifiers that estimate conditional probabilities of labels, $\eta_j(\boldsymbol{x}) = \mathbf{P}(y_j = 1|\boldsymbol{x})$, $j \in \mathcal{L}$ with possibly small L_1-estimation error (2.2), and bound the regrets with respect to these metrics of the proposed approach.

4 Probabilistic label trees (PLTs)

In this chapter, we introduce the probabilistic label trees, which efficiently estimate the conditional probabilities of labels. We discuss the factorization of these probabilities used by PLTs, give the general training procedure, and discuss several prediction procedures. PLTs have been proposed in [Jasinska and Dembczyński, 2015], then analyzed and developed in [Wydmuch et al., 2018, Jasinska-Kobus et al., 2020a].

4.1 Probabilistic label trees

Probabilistic label trees (PLTs) follow a label-tree approach to efficiently estimate the conditional probabilities of labels. They reduce the original problem to a set of binary estimation problems organized in the form of a rooted, leaf-labeled tree with m leaves. We denote a single tree by T, a root node by r_T, and the set of leaves by L_T. The leaf $l_j \in L_T$ corresponds to the label $j \in \mathcal{L}$. The set of leaves of a (sub)tree rooted in an inner node v is denoted by L_v. The set of labels corresponding to leaf nodes in L_v is denoted by $\mathcal{L}_v$. The parent node of v is denoted by $\mathrm{pa}(v)$, and the set of child nodes by $\mathrm{Ch}(v)$. A pre-leaf node is a parent node whose all children are leaves. The path from node v to the root is denoted by $\mathrm{Path}(v)$. The length of the path, that is, the number of nodes on the path, is denoted by len_v. The set of all nodes is denoted by V_T. The set of nodes of a (sub)tree rooted in an inner node v is denoted by V_v. The degree of a node $v \in V_T$, being the number of its children, is denoted by $\deg_v = |\mathrm{Ch}(v)|$. An example of a label tree is given in Figure 4.1.

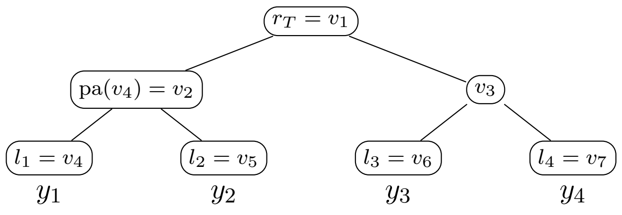


Figure 4.1: An example of a label tree T with labels $\mathcal{L} = \{y_1, y_2, y_3, y_4\}$ assigned to the leaf nodes.

The assignment of labels to tree leaves corresponds to encoding them by a prefix code, as any such code can be given in the form of a tree. Under the coding, each label y_j is uniquely represented by a code word $\boldsymbol{c}_j = (1, c_{j1}, \ldots, c_{jd})$ corresponding to a path from the root to leaf l_j. The length of the code equals the length of the path, that is, $|\boldsymbol{c}_j| = d + 1 = \mathrm{len}_{l_j}$. The zero position of the code allows one to indicate a situation in which there is no label assigned to an instance. Therefore, each label code starts with 1. For $c_{ji} \in \{0, 1\}$, the code and the label tree are binary. In general, the code alphabet can contain more than two symbols. Furthermore, c_{ji}s can take values from different sets of symbols depending on the prefix of the code word. In other words, the code can result in nodes of a different arity, like in [Grave et al., 2017] and [Prabhu et al., 2018]. Notice that any node v in the tree can be uniquely identified by the partial code word $\boldsymbol{c}_v = (1, c_{v1}, \ldots, c_{vd_v})$. An example of the coding is visualized in Figure 4.2. This coding perspective has been used in the original paper introducing the HSM model [Morin and Bengio, 2005], as well as in some later articles [Dembczyński et al., 2016, Wydmuch et al., 2018]. In the following, however, we use the tree notation introduced in the paragraph before.

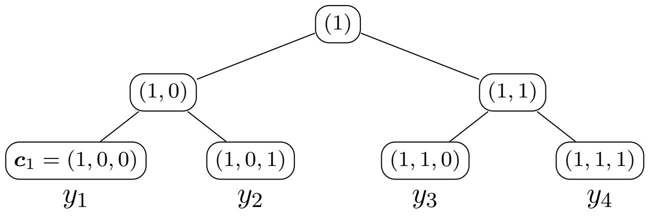


Figure 4.2: Example of assignment of codes to nodes and labels $\mathcal{L} = \{y_1, y_2, y_3, y_4\}$.

A PLT uses tree T to factorize the conditional probabilities of labels, $\eta_j(\boldsymbol{x}) = \mathbf{P}(y_j = 1|\boldsymbol{x})$, for all $j \in \mathcal{L}$. To this end let us define for every $\boldsymbol{y}$ a corresponding vector $\boldsymbol{z}$ of length $|V_T|$,[1] whose coordinates, indexed by $v \in V_T$, are given by:

$$z_v = [\![\textstyle\sum_{j \in \mathcal{L}_v} y_j \geq 1]\!] \,, \quad \text{or equivalently by } z_v = \textstyle\bigvee_{j \in \mathcal{L}_v} y_j \,. \tag{4.1}$$

In other words, the element z_v of $\boldsymbol{z}$, corresponding to the node $v \in V_T$, is set to one iff $\boldsymbol{y}$ contains at least one label in $\mathcal{L}_v$. With the above definition, it holds

[1]Note that $\boldsymbol{z}$ depends on T, but T will always be clear from the context.

based on the chain rule that for any node $v \in V_T$:

$$\eta_v(\boldsymbol{x}) = \mathbf{P}(z_v = 1 \,|\, \boldsymbol{x}) = \prod_{v' \in \mathrm{Path}(v)} \eta(\boldsymbol{x}, v') \,, \tag{4.2}$$

where $\eta(\boldsymbol{x}, v) = \mathbf{P}(z_v = 1 | z_{\mathrm{pa}(v)} = 1, \boldsymbol{x})$ for non-root nodes, and $\eta(\boldsymbol{x}, v) = \mathbf{P}(z_v = 1 \,|\, \boldsymbol{x})$ for the root. Notice that for leaf nodes we get the conditional probabilities of labels, that is,

$$\eta_{l_j}(\boldsymbol{x}) = \eta_j(\boldsymbol{x}) = \prod_{v' \in \mathrm{Path}(l_j)} \eta(\boldsymbol{x}, v') \,, \quad \text{for } l_j \in L_T \,. \tag{4.3}$$

Remark that (4.2) can also be stated as recursion:

$$\eta_v(\boldsymbol{x}) = \eta(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \,, \tag{4.4}$$

with the base case $\eta_{r_T}(\boldsymbol{x}) = \eta(\boldsymbol{x}, r_T) = \mathbf{P}(z_{r_T} = 1 \,|\, \boldsymbol{x})$.

As we deal here with multi-label distributions, the relation between probabilities of a parent node and its children is not trivial. The following result characterizes this relation precisely.

Proposition 4.1. *For any T, $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$, and internal node $v \in V_T \setminus L_T$ we have that:*

$$\sum_{v' \in \mathrm{Ch}(v)} \eta(\boldsymbol{x}, v') \geq 1 \,. \tag{4.5}$$

Moreover, the probability $\eta_v(\boldsymbol{x})$ satisfies:

$$\max\left\{\eta_{v'}(\boldsymbol{x}) : v' \in \mathrm{Ch}(v)\right\} \leq \eta_v(\boldsymbol{x}) \leq \min\left\{1, \textstyle\sum_{v' \in \mathrm{Ch}(v)} \eta_{v'}(\boldsymbol{x})\right\} \,. \tag{4.6}$$

Proof. We first prove the first inequality. From the definition of tree T and z_v we have that if $z_v = 1$, then there exists at least one $v' \in \mathrm{Ch}(v)$ for which $z_{v'} = 1$. This gives that $\sum_{v' \in \mathrm{Ch}(v)} z_{v'} \geq 1$, if $z_v = 1$. By taking expectation and recalling that $\eta(\boldsymbol{x}, v') = \mathbf{P}(z_{v'} = 1 | z_v = 1, \boldsymbol{x})$, for $v' \in \mathrm{Ch}(v)$, we get:

$$\sum_{v' \in \mathrm{Ch}(v)} \eta(\boldsymbol{x}, v') \geq 1 \,.$$

To prove (4.6) we use the above result and (4.4). As $\eta(\boldsymbol{x}, v) \in [0, 1]$, for any $v \in V_T$, therefore $\eta_{v'}(\boldsymbol{x}) \leq \eta_v(\boldsymbol{x})$ for every $v' \in \mathrm{Ch}(v)$. Moreover, from (4.4) we have that

$$\eta(\boldsymbol{x}, v') = \eta_v'(\boldsymbol{x}) / \eta_v(\boldsymbol{x}) \,,$$

for every $v' \in \mathrm{Ch}(v)$. Substituting this to (4.5) gives $\eta_v(\boldsymbol{x}) \leq \sum_{v' \in \mathrm{Ch}(v)} \eta_{v'}(\boldsymbol{x})$. Since we clearly have $\eta_v(\boldsymbol{x}) \leq 1$, we get the final result. $\square$

4.2 Training

Let $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$ be a training set consisting of n tuples consisting of a feature vector $\boldsymbol{x}_i \in \mathcal{X}$ and a label vector $\boldsymbol{y}_i \in \mathcal{Y}$. Depending on the context we also use the set notation for label vectors, that is, $\boldsymbol{y}_i \equiv \mathcal{L}_{\boldsymbol{x}_i}$. From factorization (4.2) we see that we need to train classifiers estimating $\eta(\boldsymbol{x}, v)$, for $v \in V_T$. We use a function class $\mathcal{H}^1_{\text{prob}} : \mathcal{X} \mapsto [0, 1]$ which contains probabilistic classifiers of choice, for example, logistic regression. We assign a classifier from $\mathcal{H}^1_{\text{prob}}$ to each node of the tree T. We index this set of classifiers by elements of V_T as $H = \{\hat{\eta}(v) \in \mathcal{H}^1_{\text{prob}} : v \in V_T\}$. We denote by $\hat{\eta}(\boldsymbol{x}, v)$ the prediction made by $\hat{\eta}(v)$ for some $\boldsymbol{x}$, which is the estimate of $\eta(\boldsymbol{x}, v)$. The training algorithm for a PLT is given in Algorithm 1. For simplicity, we discuss here a batch procedure, but an online counterpart can be easily obtained based on it (see Chapter 7).

Algorithm 1 PLT.TRAIN($T, A, \mathcal{D}$)

```
1:  $H = \emptyset$                                    ▷ Initialize a set of node probabilistic classifiers
2:  for each node $v \in V_T$ do                       ▷ For each node in the tree
3:      $\mathcal{D}(v) = \emptyset$                    ▷ Initialize its set of training observation in $\mathcal{D}$
4:  for $i = 1 \to n$ do                               ▷ For each training observation
5:      $(P, N) = $ ASSIGNTONODES$(T, \boldsymbol{x}_i, \mathcal{L}_{\boldsymbol{x}_i})$   ▷ Compute positive and negative nodes
6:      for $v \in P$ do                               ▷ For all positive nodes
7:          $\mathcal{D}(v) = \mathcal{D}(v) \cup \{(\boldsymbol{x}_i, z_v = 1)\}$   ▷ Add $(\boldsymbol{x}_i, z_v = 1)$ to the training set of node $t$
8:      for $v \in N$ do                               ▷ For each negative node
9:          $\mathcal{D}(v) = \mathcal{D}(v) \cup \{(\boldsymbol{x}_i, z_v = 0)\}$   ▷ Add $(\boldsymbol{x}_i, z_v = 0)$ to the training set of node $t$
10: for each node $v \in T$ do                          ▷ For all nodes in the tree
11:     $\hat{\eta}(v) = A(\mathcal{D}(v))$, $H = H \cup \{\hat{\eta}(v)\}$   ▷ Train a node classifier with algorithm $A$
12: return $H$                                         ▷ Return the set of node probabilistic classifiers
```

To train probabilistic classifiers $\hat{\eta}(v)$, $v \in V_T$, we need to properly filter training observations as given in (4.2). The TRAIN procedure first initializes the sets of training observations in all nodes of T. Then, for each training observation, it identifies the set of *positive* and *negative nodes*, that is, the nodes for which the training observation is treated respectively as positive or negative. The ASSIGNTONODES method, given in Algorithm 2, initializes the positive nodes to the empty set and the negative nodes to the root node (to deal with $\boldsymbol{y}$ of all zeros). Next, it traverses the tree from the leaves, corresponding to the labels of the training observation, to the root adding the visited nodes to the set of positive nodes. It also removes each visited node from the set of negative nodes, if it has been added to this set before. All children of the visited node, which are not in the set of positive nodes, are then added to the set of negative nodes. If the parent node of the visited node has already been added to positive nodes, the traversal of this path stops. Using the result of the ASSIGNTONODES method the algorithm distributes the training observation to the corresponding nodes. Finally, a probabilistic classifier $\hat{\eta}(v)$ is trained in each node v using algorithm A of choice.

Notice that training of each node classifier can be performed simultaneously as an independent task. The output of the algorithm is a set of probabilistic classifiers H.

Algorithm 2 PLT.AssignToNodes($T, \boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}$)

```
 1: $P = \emptyset$, $N = \{r_T\}$                       ▷ Initialize sets of positive and negative nodes
 2: for $j \in \mathcal{L}_{\boldsymbol{x}}$ do                          ▷ For all labels of the training observation
 3:     $v = \ell_j$                                   ▷ Set $v$ to a leaf corresponding to label $j$
 4:     while $v$ not null and $v \notin P$ do ▷ On path to the root or till the first positive node
 5:         $P = P \cup \{v\}$                            ▷ Assign a node to positive nodes
 6:         $N = N \setminus \{v\}$  ▷ Remove the node from negative nodes if added there before
 7:         for $v' \in \mathrm{Ch}(v)$ do                                  ▷ For all its children
 8:             if $v' \notin P$ then                        ▷ If a child is not a positive node
 9:                 $N = N \cup \{v'\}$                        ▷ Assign it to negative nodes
10:         $v = \mathrm{pa}(v)$                                       ▷ Move up along the path
11: return $(P, N)$        ▷ Return a set of positive and negative nodes for the observation
```

4.3 Prediction

For test instance $\boldsymbol{x}$, the estimate of the conditional probability of label j is a product of probability estimates on the path from the root to leaf $l_j \in L_T$:

$$\hat{\eta}_j(\boldsymbol{x}) = \prod_{v \in \mathrm{Path}(l_j)} \hat{\eta}(\boldsymbol{x}, v) \,, \tag{4.7}$$

where we assume $\hat{\eta}(\boldsymbol{x}, v) \in [0, 1]$. The recursive dependency (4.4) also holds for the estimates. We have, for any $v \in V_T$, that:

$$\hat{\eta}_v(\boldsymbol{x}) = \hat{\eta}(\boldsymbol{x}, v)\hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \,, \tag{4.8}$$

with the base case $\hat{\eta}_{r_T}(\boldsymbol{x}) = \hat{\eta}(\boldsymbol{x}, r_T)$. However, the estimates may not satisfy property (4.5) given in Proposition 4.1. Namely, it may not hold, for $v \in V_T$, that:

$$\sum_{v' \in \mathrm{Ch}(v)} \hat{\eta}(\boldsymbol{x}, v') \geq 1 \,,$$

since the node classifiers are trained independently from each other. The remedy is an additional normalization step during prediction, which may take the following form, for each child node $v' \in \mathrm{Ch}(v)$:

$$\hat{\eta}(\boldsymbol{x}, v') \leftarrow \frac{\hat{\eta}(\boldsymbol{x}, v')}{\sum_{v'' \in \mathrm{Ch}(v)} \hat{\eta}(\boldsymbol{x}, v'')} \,, \quad \text{if} \sum_{v'' \in \mathrm{Ch}(v)} \hat{\eta}(\boldsymbol{x}, v'') < 1 \,. \tag{4.9}$$

Nevertheless, this normalization is not always necessary. The theoretical results presented in Chapter 5 hold without it. Also, empirically an algorithm without normalization performs similarly, being often slightly better. However, the complexity analysis of the prediction algorithms, presented in Chapter 6, requires this normalization.

The estimation of the label probabilities is only a part of the solution as we usually need a prediction algorithm that delivers a set of labels being as similar as possible to the actual one with respect to some application-specific loss function. Below we introduce two such algorithms based on tree search. Let us first consider a prediction algorithm which finds, for a test instance $\boldsymbol{x}$, all labels such that:

$$\hat{\eta}_j(\boldsymbol{x}) \geq \tau_j \, , \quad j \in \mathcal{L} \, ,$$

where $\tau_j \in [0,1]$ are label-specific thresholds. The threshold-based predictions are in line with the theoretical analysis given in the next section, as for many performance metrics they lead to optimal decisions. Here, we present the algorithmic solution assuming that the particular values of τ_j, for all $j \in \mathcal{L}$, have been provided.

Consider the tree search procedure presented in Algorithm 3. It starts with the root node and traverses the tree by visiting the nodes $v \in V_T$ for which $\hat{\eta}_{pa(v)}(\boldsymbol{x}) \geq \tau_v$, where $\tau_v = \min\{\tau_j : l_j \in L_v\}$. It uses a simple stack $\mathcal{Q}$ to guide the search. The final prediction consists of labels corresponding to the visited leaves for which $\hat{\eta}_{\ell_j}(\boldsymbol{x}) \geq \tau_j$.

Algorithm 3 PLT.PREDICTWITHTHRESHOLDS$(T, H, \boldsymbol{\tau}, \boldsymbol{x})$

```
1:  $\hat{\boldsymbol{y}} = \mathbf{0}$, $\mathcal{Q} = \emptyset$          ▷ Initialize prediction vector to all zeros and a stack
2:  $\mathcal{Q}$.add$((r_T, \hat{\eta}(\boldsymbol{x}, r_T)))$          ▷ Add the tree root with the corresponding estimate
3:  while $\mathcal{Q} \neq \emptyset$ do          ▷ In the loop
4:      $(v, \hat{\eta}_v(\boldsymbol{x})) = \mathcal{Q}$.pop()          ▷ Pop an element from the stack
5:      if $\hat{\eta}_v(\boldsymbol{x}) \geq \tau_v$ then          ▷ If the probability estimate is greater or equal $\tau_v$
6:          if $v$ is a leaf then          ▷ If the node is a leaf
7:              $\hat{y}_v = 1$          ▷ Set the corresponding label in the prediction vector
8:          else          ▷ If the node is an internal node
9:              for $v' \in \mathrm{Ch}(v)$ do          ▷ For all child nodes
10:                 $\hat{\eta}_{v'}(\boldsymbol{x}) = \hat{\eta}_v(\boldsymbol{x}) \times \hat{\eta}(\boldsymbol{x}, v')$          ▷ Compute $\hat{\eta}_{v'}(\boldsymbol{x})$ using $\hat{\eta}(v') \in H$
11:                 $\mathcal{Q}$.add$((v', \hat{\eta}_{v'}(\boldsymbol{x})))$          ▷ Add the node and the computed estimate
12: return $\hat{\boldsymbol{y}}$          ▷ Return the prediction vector
```

The next algorithm finds the top k labels with the highest $\hat{\eta}_j(\boldsymbol{x})$, $j \in \mathcal{L}$. Consider a variant of uniform-cost search [Russell and Norvig, 2009] presented in Algorithm 4. It uses, in turn, a priority queue $\mathcal{Q}$ to guide the search. In each iteration, a node with the highest $\hat{\eta}_v(\boldsymbol{x})$ is popped from the queue. If the node is not a leaf, then each its child node v' is added to the priority queue with its $\hat{\eta}_{v'}(\boldsymbol{x})$. Otherwise, a label corresponding to the visited leaf is added to the prediction. If the number of predicted labels is equal k, then the procedure returns prediction $\hat{\boldsymbol{y}}$. The priority queue guarantees that k labels with the highest $\hat{\eta}_j(\boldsymbol{x})$, $j \in \mathcal{L}$, are predicted.

In many practical applications, the above algorithms work in time logarithmic (or close to logarithmic) in the number of labels m, as discussed formally in Chapter 6. However, in the worst case, both can visit all nodes of the tree. To

Algorithm 4 PLT.PREDICTTOPLABELS$(T, H, k, \boldsymbol{x})$

```
 1: $\hat{\boldsymbol{y}} = \mathbf{0}$, $\mathcal{Q} = \emptyset$,      ▷ Initialize prediction vector to all zeros and a priority queue
 2: $k' = 0$                                   ▷ Initialize counter of predicted labels
 3: $\mathcal{Q}$.add$((r_T, \hat{\eta}(\boldsymbol{x}, r_T)))$      ▷ Add the tree root with the corresponding estimate
 4: while $k' < k$ do                ▷ While the number of predicted labels is less than $k$
 5:     $(v, \hat{\eta}_v(\boldsymbol{x})) = \mathcal{Q}$.pop()                ▷ Pop the top element from the queue
 6:     if $v$ is a leaf then                              ▷ If the node is a leaf
 7:         $\hat{y}_v = 1$             ▷ Set the corresponding label in the prediction vector
 8:         $k' = k' + 1$                                  ▷ Increment the counter
 9:     else                                     ▷ If the node is an internal node
10:         for $v' \in \mathrm{Ch}(v)$ do                          ▷ For all its child nodes
11:             $\hat{\eta}_{v'}(\boldsymbol{x}) = \hat{\eta}_v(\boldsymbol{x}) \times \hat{\eta}(\boldsymbol{x}, v')$        ▷ Compute $\hat{\eta}_{v'}(\boldsymbol{x})$ using $\hat{\eta}(v') \in H$
12:             $\mathcal{Q}$.add$((v', \hat{\eta}_{v'}(\boldsymbol{x})))$ ▷ Add the node and the computed probability estimate
13: return $\hat{\boldsymbol{y}}$                                  ▷ Return the prediction vector
```

control the complexity directly, an algorithm based on beam search can be used as an alternative to uniform-cost search. Its disadvantage is that it may not find the exact top-k scoring labels. Moreover, as shown by experiments, the average number of visited nodes is not necessarily smaller than that of uniform-cost search. Algorithm 5 shows the pseudocode of this algorithm.

Algorithm 5 PLT.PREDICTBEAMSEARCH$(T, H, k, B, \boldsymbol{x})$

```
 1: $\hat{\boldsymbol{y}} = \mathbf{0}$, $\mathcal{Q} = \emptyset$,          ▷ Initialize prediction vector to all zeros and a list
 2: $\mathbf{p} = \mathbf{0}$                  ▷ Initialize prediction vector to all zero probabilities
 3: $\mathcal{Q}$.add$((r_T, \hat{\eta}(\boldsymbol{x}, r_T)))$      ▷ Add the tree root with the corresponding estimate
 4: while $\mathcal{Q}$ is not empty do              ▷ While there is a tree level to be processed
 5:     $\mathcal{Q}_{\text{next}} = \emptyset$
 6:     while $\mathcal{Q}$ is not empty do
 7:         $(v, \hat{\eta}_v(\boldsymbol{x})) = \mathcal{Q}$.pop()
 8:         for $v' \in \mathrm{Ch}(v)$ do                          ▷ For all its child nodes
 9:             $\hat{\eta}_{v'}(\boldsymbol{x}) = \hat{\eta}_v(\boldsymbol{x}) \times \hat{\eta}(\boldsymbol{x}, v')$        ▷ Compute $\hat{\eta}_{v'}(\boldsymbol{x})$ using $\hat{\eta}(v') \in H$
10:             if $v'$ is a leaf then                              ▷ If the node is a leaf
11:                 $\mathbf{p}_{v'} = \hat{\eta}_{v'}(\boldsymbol{x})$      ▷ Add label and its estimate to the retrieved leaves
12:             else
13:                 $\mathcal{Q}_{\text{next}}$.add$((v', \hat{\eta}_{v'}(\boldsymbol{x})))$         ▷ Add the node and the estimate to $\mathcal{Q}_{\text{next}}$
14:     $\mathcal{Q}$ = top $B$ elements of $\mathcal{Q}_{\text{next}}$     ▷ Leave only the top-$B$ scoring nodes from $\mathcal{Q}_{\text{next}}$
15: $\hat{\mathcal{L}}_k$ = top $k$ scoring labels from $\mathbf{p}_{v'}$     ▷ Select the $k$ highest scoring retrieved leaves
16: for $j \in \hat{\mathcal{L}}_k$ do $\hat{\boldsymbol{y}}_j = 1$
17: return $\hat{\boldsymbol{y}}$                                  ▷ Return the prediction vector
```

This algorithm works in a breadth-first manner searching the tree level by level. We denote the width of the beam by B. To guide the search it uses a list $\mathcal{Q}$ of B internal nodes from the previous level with the highest intermediate estimates of conditional probabilities. At each level, it visits all the children of nodes in $\mathcal{Q}$, and constructs the $\mathcal{Q}_{\text{next}}$ for the next level by selecting B child nodes v' with the highest $\hat{\eta}_{v'}(\boldsymbol{x})$. If a leaf is encountered, the $\hat{\eta}_{v'}(\boldsymbol{x})$ of this node is saved. Once the search is finished, the algorithm selects k highest scored leaves from the set of visited leaves.

5 Statistical analysis of PLTs

In this chapter, we analyze the PLT model in terms of its statistical properties. We first upper-bound the L_1 estimation error of conditional probabilities of labels, $|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|$, by the L_1 error of the node classifiers, $|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|$. We then generalize this result to a wide class of strongly proper composite losses to connect the L_1 error of conditional probability estimates with a learning algorithm used in the tree nodes. Having the link between the quality of solving the binary problems, and the L_1 estimation error, we move on to regrets of metrics discussed in Chapter 2. Most of these results are included in [Jasinska-Kobus et al., 2020a]. Finally, we discuss the relation of PLTs to hierarchical softmax [Wydmuch et al., 2018].

5.1 L_1 estimation error

We start with the L_1 estimation error, being a building block for the rest of the results. First, we give a bound that expresses the quality of probability estimates $\hat{\eta}_v(\boldsymbol{x})$ in nodes v for any node $v \in V_T$. The lemma and corollary below generalize a similar result obtained for multi-class classification in [Beygelzimer et al., 2009b].

Lemma 5.1. *For any tree T and distribution $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$ the following holds for each $v \in V_T$:*

$$|\eta_v(\boldsymbol{x}) - \hat{\eta}_v(\boldsymbol{x})| \leq \sum_{v' \in \mathrm{Path}(v)} \eta_{\mathrm{pa}(v')}(\boldsymbol{x}) \left|\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')\right| , \qquad (5.1)$$

where we assume $\hat{\eta}(\boldsymbol{x}, v) \in [0, 1]$, for each $v \in V_T$, and $\eta_{\mathrm{pa}(r_T)}(\boldsymbol{x}) = 1$, for the root node r_T.

From this lemma, we immediately get guarantees for the estimate of the conditional probability for label $j \in \mathcal{L}$, which corresponds to a bound for the leaf node l_j.

Corollary 5.2. *For any tree T and distribution $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$, the following holds for each label $j \in \mathcal{L}$:*

$$|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \le \sum_{v \in \mathrm{Path}(l_j)} \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \left| \eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v) \right| , \tag{5.2}$$

where we assume $\eta_{\mathrm{pa}(r_T)}(\boldsymbol{x}) = 1$ for the root node r_T.

It is worth to notice that the above bounds are tighter than the one in [Beygelzimer et al., 2009b], since the L_1 estimation error of the node classifiers is additionally multiplied by the probability of the parent node $\eta_{\mathrm{pa}(v')}(\boldsymbol{x})$. Our proof, given below, is also obtained using different arguments.

Proof. Recall the recursive factorization of probability $\eta_v(\boldsymbol{x})$ given in (4.4):

$$\eta_v(\boldsymbol{x}) = \eta(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) .$$

As the same recursive relation holds for $\hat{\eta}_v(\boldsymbol{x})$, see (4.8), we have that

$$|\eta_v(\boldsymbol{x}) - \hat{\eta}_v(\boldsymbol{x})| = \left| \eta(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x}, v)\hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \right| .$$

By adding and subtracting $\hat{\eta}(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x})$, using the triangle inequality $|a + b| \le |a| + |b|$ and the assumption that $\hat{\eta}(\boldsymbol{x}, v) \in [0, 1]$, we obtain:

$$\begin{aligned}
&|\eta_v(\boldsymbol{x}) - \hat{\eta}_v(\boldsymbol{x})| = \\
&\quad = \left| \eta(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) + \hat{\eta}(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x}, v)\hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \right| \\
&\quad \le \left| \eta(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \right| + \left| \hat{\eta}(\boldsymbol{x}, v)\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x}, v)\hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \right| \\
&\quad \le \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \left| \eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v) \right| + \hat{\eta}(\boldsymbol{x}, v) \left| \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \right| \\
&\quad \le \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \left| \eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v) \right| + \left| \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) - \hat{\eta}_{\mathrm{pa}(v)}(\boldsymbol{x}) \right|
\end{aligned}$$

Since the rightmost term corresponds to the L_1 error of the parent of v, we use recursion to get the result of Lemma 5.1:

$$|\eta_v(\boldsymbol{x}) - \hat{\eta}_v(\boldsymbol{x})| \le \sum_{v' \in \mathrm{Path}(v)} \eta_{\mathrm{pa}(v')}(\boldsymbol{x}) \left| \eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v') \right| ,$$

where for the root node $\eta_{\mathrm{pa}(r_T)}(\boldsymbol{x}) = 1$. As the above holds for any $v \in V$, the result also applies to conditional probabilities of labels as stated in Corollary 5.2. □

The above results are conditioned on $\boldsymbol{x}$ and concern a single node $v \in V$. The expected L_1 error for label j and $\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})$ is given by the next lemma.

Lemma 5.3. *For any tree T and distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$ the following holds for $j \in \mathcal{L}$:*

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})} \left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \right] \\
&\qquad \le \sum_{v \in \mathrm{Path}(l_j)} \mathbf{P}(z_{\mathrm{pa}(v)} = 1) \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x} | z_{\mathrm{pa}(v)} = 1)} \left[|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)| \right] ,
\end{aligned}$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$.

This lemma bounds the expected L_1 error for label j and $\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})$ by a weighted sum of L_1 errors of node v classifiers in $\mathrm{Path}(l_j)$. The weights correspond to the probabilities that an observation reaches each node v, $\mathbf{P}(z_{\mathrm{pa}(v)} = 1)$. The expected L_1 errors are given for conditional distribution of $\boldsymbol{x}$ defined also on $\boldsymbol{x}$ reaching node v.

Finally, the next theorem gives the understanding of the average performance over all labels and the entire distribution $\mathbf{P}(\boldsymbol{x})$. We present the result in a general form of a weighted average as this form will be required later.

Theorem 5.4. *For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, and weights $W_j \in R$, $j \in \{1, \ldots, m\}$, the following holds:*

$$\frac{1}{m}\sum_{j=1}^{m} W_j \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \leq \\ \frac{1}{m}\sum_{v\in V} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')|\right] \sum_{j\in L_v} W_j\,, \quad (5.3)$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$. *For* $W_j = 1$, $j \in \{1, \ldots, m\}$, *we have:*

$$\frac{1}{m}\sum_{j=1}^{m} \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \leq \\ \frac{1}{m}\sum_{v\in V} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')|\right] |L_v|\,.$$

The result states that the weighted expected L_1 estimation error averaged over all labels can be bounded by a weighted sum of expected L_1 errors of node classifiers divided by the number of labels. A weight associated with node v is a product of the probability mass of a parent node and the number of leave nodes in a subtree rooted in v. This means that a node closer to the root has a higher impact on the overall performance. This agrees with the intuition as such nodes impact estimates of more labels. We omit the proof of this theorem here as it is quite technical. It is presented with other proofs in Appendix A.2.

5.2 Strongly proper composite losses

Recall that for λ-strongly proper composite loss ℓ_c for any $\eta(\boldsymbol{x}), \psi^{-1}(f(\boldsymbol{x})) \in [0, 1]$:

$$\left|\eta(\boldsymbol{x}) - \psi^{-1}(f(\boldsymbol{x}))\right| \leq \sqrt{\frac{2}{\lambda}}\sqrt{\mathrm{reg}_{\ell_c}(f \,|\, \boldsymbol{x})}\,.$$

We apply the above to node classifiers in a PLT tree. In each node $v \in V_T$ we consider a binary problem with $y = 2z_v - 1$ and pairs $(\boldsymbol{x}, z_v)$ generated i.i.d. according to $\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)} = 1)$. Moreover, let f_v be a scoring function in node

$v \in V_T$ minimized by a strongly proper composite loss function ℓ_c. The estimates $\hat{\eta}(\boldsymbol{x}, v)$, for all $v \in V$, are then computed as:

$$\hat{\eta}(\boldsymbol{x}, v) = \psi^{-1}(f_v(\boldsymbol{x})) .$$

We start with Corollary 5.2 and apply the bound from (2.3), recalled above.

Corollary 5.5. *For any tree T and distribution $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$, a strongly proper composite loss function ℓ_c, the following holds for each label $j \in \mathcal{L}$:*

$$|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \leq \sqrt{\frac{2}{\lambda}} \sum_{v \in \mathrm{Path}(l_j)} \eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \sqrt{\mathrm{reg}_{\ell_c}(f_v \,|\, \boldsymbol{x})} \,, \tag{5.4}$$

where we assume $\eta_{\mathrm{pa}(r_T)}(\boldsymbol{x}) = 1$ for the root node r_T.

This result shows how the ℓ_c regrets accumulate among classifiers from $\mathrm{Path}(l_j)$ and bound the L_1 error of estimation of $\eta_j(\boldsymbol{x})$ for a single label j. To bound the expected L_1 error for all labels using the ℓ_c regrets of node classifiers, we give a result similar to the one in Theorem 5.4.

Theorem 5.6. *For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, weights $W_j \in R$, $j \in \{1, \ldots, m\}$, a strongly proper composite loss function ℓ_c, and a set of scoring functions f_v, $v \in V_T$, the following holds:*

$$\frac{1}{m} \sum_{j=1}^{m} W_j \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})} \left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \right] \leq \frac{\sqrt{2}}{m\sqrt{\lambda}} \sum_{v \in V} \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1) \mathrm{reg}_{\ell_c}(f_v)} \sum_{j \in L_v} W_j \,, \tag{5.5}$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$, and $\mathrm{reg}_{\ell_c}(f_v)$ is the expected ℓ_c-regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)} = 1)$. For $W_j = 1$, $j \in \{1, \ldots, m\}$, we have:

$$\frac{1}{m} \sum_{j=1}^{m} \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})} \left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \right] \leq \frac{\sqrt{2}}{m\sqrt{\lambda}} \sum_{v \in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1) \mathrm{reg}_{\ell_c}(f_v)} \,. \tag{5.6}$$

The theorem justifies the use of strongly proper composite losses during the training of node classifiers. The technical details of the proof are presented in Appendix A.3. Here, we only notice that the weights of node errors follow from Theorem 5.4, while the squared root dependency from (2.3).

5.3 Generalized classification performance metrics

In this section, we use the previously obtained bounds to demonstrate the regret bounds for generalized classification performance metrics. The regret of the Ψ_{macro} metric decomposes into a weighted sum:

$$\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}) = \Psi_{\mathrm{macro}}(\boldsymbol{h}^*_{\boldsymbol{\alpha}^*_{\Psi}}) - \Psi_{\mathrm{macro}}(\boldsymbol{h}) = \frac{1}{m} \sum_{j=1}^{m} (\Psi(h^*_{j,\alpha^*_{\Psi,j}}) - \Psi(h_j)) \tag{5.7}$$

In turn, the regret of the Ψ_{micro} metric is given by:

$$\mathrm{reg}_{\Psi_{\mathrm{micro}}}(\boldsymbol{h}) = \Psi_{\mathrm{micro}}(\boldsymbol{h}^*_{\alpha^*_\Psi}) - \Psi_{\mathrm{micro}}(\boldsymbol{h}) \,. \tag{5.8}$$

We assume, similarly as in the previous subsection, that a score function $f_v(\boldsymbol{x})$ in a node $v \in V$ is trained via minimization of a strongly proper composite loss function ℓ_c. We are interested in bounding these regrets with the performance of node classifiers of a PLT. The estimates $\hat{\eta}(\boldsymbol{x}, v)$, for all $v \in V$, are then computed as:

$$\hat{\eta}(\boldsymbol{x}, v) = \psi^{-1}(f_v(\boldsymbol{x}))\,.$$

The prediction of a PLT is computed by Algorithm 3 and has a form similar to the optimal classifier (3.5):

$$\boldsymbol{h}_{\boldsymbol{\tau}}(\boldsymbol{x}) = (h_{1,\tau_1}(\boldsymbol{x}), h_{2,\tau_2}(\boldsymbol{x}), \ldots, h_{m,\tau_m}(\boldsymbol{x}))\,, \text{ where } h_{j,\tau_j}(\boldsymbol{x}) = [\![\hat{\eta}_j(\boldsymbol{x}) > \tau_j]\!]\,,$$

for some vector $\boldsymbol{\tau} = (\tau_1, \tau_2, \ldots, \tau_m) \in [0,1]^m$ of thresholds. Estimates $\hat{\eta}_j(\boldsymbol{x})$ are computed as in (4.7), that is, $\hat{\eta}_j(\boldsymbol{x}) = \prod_{v \in \mathrm{Path}(l_j)} \hat{\eta}(\boldsymbol{x}, v)$, where $l_j \in L_T$ is a node corresponding to label j.

Theorem 5.7. *Let $\tau^*_j = \arg\max_\tau \Psi(h_{j,\tau})$, for each $j \in \mathcal{L}$, and $\boldsymbol{\tau}^* = (\tau^*_1, \tau^*_2, \ldots, \tau^*_m)$. For any tree T and distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, the classifier $\boldsymbol{h}_{\boldsymbol{\tau}^*}$ achieves the following upper bound on its Ψ_{macro}-regret:*

$$\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\tau}^*}) \le \frac{\sqrt{2}}{m\sqrt{\lambda}} \sum_{v \in V} \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)} \sum_{j \in L_v} C_j \,,$$

where $C_j = \frac{1}{\gamma}(\Psi(h^*_\Psi, j)(b_1 + b_2) - (a_1 + a_2))$, for each $j \in \mathcal{L}$, with γ defined in (3.2), *$\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ for the root node, and $\mathrm{reg}_{\ell_c}(f_v)$ is the expected ℓ_c-regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v \mid z_{\mathrm{pa}(v)} = 1)$.*

Theorem 5.8. *Let $\boldsymbol{h}_\tau = (h_{1,\tau}, h_{2,\tau}, \ldots, h_{m,\tau})$ be a classifier which shares the same threshold τ over all labels $j \in \mathcal{L}$. For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, and $\tau^* = \arg\max_\tau \Psi_{\mathrm{micro}}(\boldsymbol{h}_\tau)$, classifier $\boldsymbol{h}_{\tau^*}$ achieves the following upper bound on its Ψ_{micro}-regret:*

$$\mathrm{reg}_{\Psi_{micro}}(\boldsymbol{h}_{\tau^*}) \le \frac{C}{m}\sqrt{\frac{2}{\lambda}} \sum_{v \in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,,$$

where $C = \frac{1}{\gamma}(\Psi_{\mathrm{micro}}(\boldsymbol{h}^*_\Psi)(b_1 + b_2) - (a_1 + a_2))$ with γ defined in (3.2), *$\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ for the root node, and $\mathrm{reg}_{\ell_c}(f_v)$ is the expected ℓ_c-regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v \mid z_{\mathrm{pa}(v)} = 1)$.*

The above theorems can be interpreted in the following way. For conditional probability estimates $\hat{\eta}_j(\boldsymbol{x})$, $j \in \mathcal{L}$, obtained as described just before the theorem, there exists a vector $\boldsymbol{\tau}$ of thresholds, for which the regret of a generalized performance metric is upper-bounded solely by regrets of node classifiers, expressed in terms of a strongly proper composite loss function. Therefore, from the perspective of learning one needs to focus on the node classifiers to get as accurate as possible estimates of conditional probabilities of labels, by minimizing a strongly proper composite loss function in each node. The next step, being independent of the previous one, is to obtain the right values of thresholds $\boldsymbol{\tau}^*$, following one of the approaches mentioned above.

Let us analyze the regret bounds more carefully. In the case of macro-averaged metrics, the regret of each node classifier is weighted by the sum of C_j-values of all labels in the corresponding subtree. In the case of micro-averaged metrics, there is only one global C-value, and each node classifier is weighted by the number of labels in the corresponding subtree. The values of C and γ for different metrics are given in Table 5.1 (a similar table can be found in [Kotłowski and Dembczyński, 2017]). It is easy to verify with these values that for the Hamming loss the regret bounds for macro- and micro-averaging are the same. This agrees with the fact that both averaging schemes boil down to the same metric in the case of the Hamming loss. In general, the macro- and micro-averaging bounds coincide for all metrics with constant C. Interestingly, the bounds are different for the F_1-measure and the Jaccard similarity, while they both share the same optimal solution (since the Jaccard similarity is a strictly monotone transformation of the F_1-measure). As γ is the same for both metrics, this observation suggests that C could be defined more tightly. One can also observe that C grows with decreasing P. Therefore, for sparse problems and labels from the long-tail the value of C can be large, potentially leading to poor guarantees.

Metric	γ	C
Hamming loss	1	2
F_β-measure	$\beta^2 P$	$\frac{1+\beta}{\beta^2 P}$
Jaccard similarity	P	$\frac{\Psi(\mathrm{FP}^*,\mathrm{FN}^*)+1}{P}$
AM	$2P(1-P)$	$\frac{1}{2P(1-P)}$

Table 5.1: The values of γ and C values for some generalized classification performance metrics. As before, P denotes $\mathbf{P}(y_j = 1)$, for macro-averaging, or $\frac{1}{m}\sum_{j=1}^{m}\mathbf{P}(y_j = 1)$, for micro-averaging.

The proofs of both theorems are given in Appendix A.4. They are based on results previously obtained for the 1-VS-ALL approach in [Kotłowski and Dembczyński, 2017], combined with Theorem 5.6. The result for the 1-VS-ALL approach is based on two observations. The first one states that the regret for a cost-sensitive binary classification can be upper-bounded by the L_1 estimation error of the conditional probabilities, if a classification procedure uses a threshold which directly corresponds to the misclassification cost. The second shows that the regret of the generic function $\Psi(\mathrm{FP}, \mathrm{FN})$ can be upper-bounded by the regret of the cost-sensitive binary classification with costs related to α^*_Ψ. The actual value of the optimal thresholds is a direct consequence of the proof. Putting these two observations together along with Theorem 5.6 gives the final results.

5.4 Precision@k

We analyze also the @k metrics, and again use the previously obtained bounds for strongly proper composite loss functions. We start with giving a regret bound for precision@k. The optimal strategy for precision@k is predicting labels with the highest conditional probabilities, $\hat{\mathcal{L}}^*_{\boldsymbol{x}}$. However, a classifier $\boldsymbol{h}_{@k}(\boldsymbol{x})$ predicts a set $\hat{\mathcal{L}}^*_{\boldsymbol{x}}$, which not necessarily is equal to $\hat{\mathcal{L}}^*_{\boldsymbol{x}}$. Therefore $\boldsymbol{h}_{@k}(\boldsymbol{x})$ may suffer a regret with respect to precision@k. This conditional regret for precision@k is:

$$\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k} \,|\, \boldsymbol{x}) = \frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \eta_i(\boldsymbol{x}) - \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \eta_j(\boldsymbol{x}) \,.$$

The conditional regret with respect to precision@k can be upper-bounded by the L_1-estimation errors as stated by the following theorem.

Theorem 5.9. *For any distribution $\mathbf{P}(\boldsymbol{y} \,|\, \boldsymbol{x})$ and classifier $\boldsymbol{h}_{@k} \in \mathcal{H}^m_{@k}$ the following holds:*

$$\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k} \,|\, \boldsymbol{x}) = \frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \eta_i(\boldsymbol{x}) - \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \eta_j(\boldsymbol{x}) \le 2\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \,.$$

Proof. Let us add and subtract the following two terms, $\frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \hat{\eta}_i(\boldsymbol{x})$ and $\frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \hat{\eta}_j(\boldsymbol{x})$, to the regret and reorganize the expression in the following way:

$$\begin{aligned}
\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k} \,|\, \boldsymbol{x}) = & \underbrace{\frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \eta_i(\boldsymbol{x}) - \frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \hat{\eta}_i(\boldsymbol{x})}_{\le \frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} |\eta_i(\boldsymbol{x}) - \hat{\eta}_i(\boldsymbol{x})|} \\
& + \underbrace{\frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \hat{\eta}_j(\boldsymbol{x}) - \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \eta_j(\boldsymbol{x})}_{\le \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} |\hat{\eta}_j(\boldsymbol{x}) - \eta_j(\boldsymbol{x})|} \\
& + \underbrace{\frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} \hat{\eta}_i(\boldsymbol{x}) - \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \hat{\eta}_j(\boldsymbol{x})}_{\le 0} \\
& \le \frac{1}{k}\sum_{i \in \hat{\mathcal{L}}^*_{\boldsymbol{x}}} |\eta_i(\boldsymbol{x}) - \hat{\eta}_i(\boldsymbol{x})| + \frac{1}{k}\sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|
\end{aligned}$$

Next we bound each L_1 error, $|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|$ by $\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|$. There are at most $|\mathcal{Y}_k| + |\hat{\mathcal{Y}}_k| = 2k$ such terms. Therefore

$$\mathrm{reg}_{p@k}(\boldsymbol{h} \,|\, \boldsymbol{x}) \le 2\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \,.$$

□

Interestingly, the bound does not depend neither on k nor m. However, if

$k = m$ then $\mathrm{reg}_{p@m} = 0$ for any distribution, since $\hat{\mathcal{L}}^*_{\boldsymbol{x}} = \hat{\mathcal{L}}_{\boldsymbol{x}}$ in this case. In general, if $m < 2k$, then $\hat{\mathcal{L}}^*_{\boldsymbol{x}} \cap \hat{\mathcal{L}}_{\boldsymbol{x}} \neq \emptyset$. In other words, some of the labels from $\hat{\mathcal{L}}_{\boldsymbol{x}}$ are also in $\hat{\mathcal{L}}^*_{\boldsymbol{x}}$, so the bound can be tighter. For example, one can multiply the bound by $\frac{\min(k, m-k)}{k}$, assuming that $k \leq m$. However, in extreme classification usually $k \ll m$, so we do not use the more complex bound.

Again, we assume that a score function $f_v(\boldsymbol{x})$ in a node $v \in V$ is trained via minimization of a strongly proper composite loss function ℓ_c, and the estimates $\hat{\eta}(\boldsymbol{x}, v)$, for all $v \in V$, are computed as: $\hat{\eta}(\boldsymbol{x}, v) = \psi^{-1}(f_v(\boldsymbol{x}))$. We assume that the k predicted labels are the k labels with the highest $\hat{\eta}_j(\boldsymbol{x})$. Such labels are found by Algorithm 4. The form of the final classifier $\boldsymbol{h}_{@k}(\boldsymbol{x}) \in \mathcal{H}^m_{@k}$ is then similar to (3.6), but with permutation $\pi(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))$ defined over $\hat{\eta}_j(\boldsymbol{x})$, $j \in \mathcal{L}$. The next theorem provides an upper bound of the unconditional regret for a PLT. Unfortunately, the max operator from Theorem 5.9 needs to be replaced by the sum in the derivations, therefore the theorem has the following form.

Theorem 5.10. *For any tree T and distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, classifier $\boldsymbol{h}_{@k}(\boldsymbol{x})$ achieves the following upper bound on its precision@k regret:*

$$\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k}) \leq \frac{2\sqrt{2}}{\sqrt{\lambda}} \sum_{v \in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,,$$

where $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ for the root node, and $\mathrm{reg}_{\ell_c}(f_v)$ is the expected ℓ_c-regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)} = 1)$.

Proof. By taking an expectation over $\mathbf{P}(\boldsymbol{x})$ of the bound from Theorem 5.9 and replacing the max operator by sum, that is, $\max(a, b) \leq a + b$, for $a, b \geq 0$, we obtain:

$$\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k}) \leq 2 \sum_{j=1}^{m} \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})} \left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \right]$$

Next, by applying (5.6) from Theorem 5.6, we get the statement:

$$\mathrm{reg}_{p@k}(\boldsymbol{h}_{@k}) \leq \frac{2\sqrt{2}}{\sqrt{\lambda}} \sum_{v \in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,.$$

□

It is worth comparing the above bound with the one for the Hamming loss, taken either from Theorem 5.7 or Theorem 5.8, with $C = 2$ and $\gamma = 1$ (see Table 5.1). It turns out that the bound for precision@k is m times larger. The reason is that if there were k labels with the L_1-estimation error approaching 1, but with the actual probability close to 0, then the precision@k regret would get its maximum. On the other hand, the Hamming loss regret is an average of label-wise regrets. Therefore, it does not suffer much, as there were only k labels out of m with the highest regret.

5.5 DCG@k

Let us now move on to DCG@k, for which we give similar results to the ones given for precision@k. In the following, we consider the rankings of labels. Recall the definition of label ranking $\boldsymbol{\pi}(\mathbf{s})$ according to label-wise scores s, in which $\pi_r(\mathbf{s})$ is the r-th ranked label. We denote the rank of label j in the label ranking $\pi(\mathbf{s})$ with $\pi_j^{-1}(\mathbf{s})$. The optimal classifier for DCG@k ranks the labels in the order of decreasing conditional probabilities $\eta_j(\boldsymbol{x})$, and produces a ranking $\pi(\boldsymbol{h}^*_{D@k})$. However, a classifier $\boldsymbol{h}(\boldsymbol{x})$ gives a ranking $\boldsymbol{\pi}(\boldsymbol{h})$ that is not necessarily is equal on first k ranks to $\boldsymbol{\pi}(\boldsymbol{h}^*_{D@k})$. The conditional regret for DCG@k is then:

$$\begin{aligned}\mathrm{reg}_{D@k}(\boldsymbol{h}\,|\,\boldsymbol{x}) &= \sum_{r=1}^{k}\big(\eta_{\pi_r(\boldsymbol{h}^*_{D@k})}(\boldsymbol{x}) - \eta_{\pi_r(\boldsymbol{h})}(\boldsymbol{x})\big)g(r)\\ &= \sum_{i\in\mathcal{L}^*_k}\eta_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}^*_{D@k})) - \sum_{j\in\hat{\mathcal{L}}_k}\eta_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}))\end{aligned}$$

where $\mathcal{L}^*_k$ is the set of labels on first k ranks in ranking $\boldsymbol{\pi}(\boldsymbol{h}^*_{D@k})$, and analogously, $\hat{\mathcal{L}}_k$ is the set of labels on first k ranks in ranking $\boldsymbol{\pi}(\boldsymbol{h})$. The conditional regret with respect to DCG@k can be upper-bounded by the L_1-estimation error analogously to the bound for precision@k. Since the proof is also analogous, we give it in the Appendix A.5.

Theorem 5.11. *For any distribution $\mathbf{P}(\boldsymbol{y}\,|\,\boldsymbol{x})$ and classifier $\boldsymbol{h}\in\mathcal{H}^m$ the following holds:*

$$\begin{aligned}\mathrm{reg}_{D@k}(\boldsymbol{h}_{@k}\,|\,\boldsymbol{x}) &= \sum_{i\in\mathcal{L}^*_k}\eta_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}^*_{D@k})) - \sum_{j\in\hat{\mathcal{L}}_k}\eta_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}_{@k}))\\ &\le 2\max_j|\eta_j(\boldsymbol{x})-\hat{\eta}_j(\boldsymbol{x})|\,IDCG@k(k)\,,\end{aligned}$$

where $IDCG@k(k)=\sum_{r=1}^{k}g(r)$, $\mathcal{L}^*_k$ is the set of labels on first k ranks in ranking $\pi(\boldsymbol{h}^*_{D@k})$, and $\hat{\mathcal{L}}_k$ is the set of labels on first k ranks in ranking $\boldsymbol{\pi}(\boldsymbol{h})$.

Compare the regret bounds for precision@k, from Theorem 5.9, and DCG@k, above. They differ only by a factor IDCG@$k(k)$. The bound for precision@k is a special case of the bound for DCG@k, with constant, rank-independent, gains g equal to $\frac{1}{k}$. Moreover, by maximizing DCG@k one also optimizes precision@k. Finally, we give an upper bound of the unconditional DCG@k regret in terms of strongly proper composite loss functions, and prove it in the Appendix A.5.

Theorem 5.12. *For any tree T and distribution $\mathbf{P}(\boldsymbol{x},\boldsymbol{y})$, classifier $\boldsymbol{h}(\boldsymbol{x})$ achieves the following upper bound on its DCG@k regret:*

$$\mathrm{reg}_{D@k}(\boldsymbol{h}) \le IDCG@k(k)\frac{2\sqrt{2}}{\sqrt{\lambda}}\sum_{v\in V}|L_v|\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v)}\,.$$

where IDCG@k(k) $= \sum_{r=1}^{k} g(r)$, $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ *for the root node, and* $\mathrm{reg}_{\ell_c}(f_v)$ *is the expected* ℓ_c*-regret of* f_v *taken over* $\mathbf{P}(\boldsymbol{x}, z_v \mid z_{\mathrm{pa}(v)} = 1)$*.*

5.6 Relation to hierarchical softmax

Finally, we show that PLTs are strictly related to hierarchical softmax [Morin and Bengio, 2005]. Hierarchical softmax is designed for multi-class classification. Recall that using our notation, we have $\sum_{i=1}^{m} y_i = 1$ for multi-class problems, that is, there is one and only one label assigned to an instance $(\boldsymbol{x}, \boldsymbol{y})$. The conditional probabilities $\eta_j(\boldsymbol{x})$ in this case sum up to 1. Since in multi-class classification always exactly one label is assigned to an instance, there is no need to learn a root classifier that verifies whether there exists a positive label for an instance. Nevertheless, the factorization of the conditional probability of label j is given by the same equation (4.2) as for multi-label case:

$$\eta_j(\boldsymbol{x}) = \prod_{v' \in \mathrm{Path}(l_j)} \eta(\boldsymbol{x}, v')\,.$$

However, in this case, $\eta(\boldsymbol{x}, v') = 1$, for v' being the root, and the $\eta(\boldsymbol{x}, v')$ among v' children of a single node v sum to one

$$\sum_{v' \in \mathrm{Ch}(v)} \eta(\boldsymbol{x}, v') = 1\,,$$

since $\sum_{i=1}^{m} y_i = 1$. The model above is the same as the one presented in [Morin and Bengio, 2005], where the parent nodes are identified by a code indicating a path from the root to this node. When used with a sigmoid function to model the conditional probabilities, we obtain the popular formulation of hierarchical softmax.

To deal with multi-label problems, some popular tools, such as FASTTEXT [Joulin et al., 2017] and its extension LEARNED TREE [Jernite et al., 2017], apply hierarchical softmax with the pick-one-label heuristic (3.8). As we have shown in Section 3.2, this heuristic is not consistent with respect to precision@k and L_1 estimation error of conditional probabilities of labels. In this chapter, we have proven that PLTs are consistent for a wide spectrum of metrics. Since on multi-class data PLTs boil down to hierarchical softmax, we can say that PLTs are a non-regret generalization of hierarchical softmax to multi-label classification with respect to those metrics.

6 Computational complexity of PLTs

In extreme classification the computational performance of the training and prediction algorithms is crucial. In this chapter, we briefly analyze the training and prediction complexity of PLTs, mostly following [Busa-Fekete et al., 2019], and adding the results related to uniform-cost search and beam search. We first define training and prediction costs as the number of updated or evaluated node classifiers. Then, we bound those costs, relate the prediction cost with the training cost and the L_1 error of the classifiers, and explain the relation of cost to the computational complexity of training and prediction algorithms.

6.1 Training complexity

We define the training complexity of PLTs in terms of the number of nodes in which a training observation $(\boldsymbol{x}, \boldsymbol{y})$ is used. From the definition of the tree and the PLT model (4.2), we have that each training observation is used in the root, to estimate $\mathbf{P}(z_{r_T} = 1|\boldsymbol{x})$, and in each node v for which $z_{\mathrm{pa}(v)} = 1$, to estimate $\mathbf{P}(z_v = 1|z_{\mathrm{pa}(v)} = 1, \boldsymbol{x})$. Therefore, the *training cost for a single observation* $(\boldsymbol{x}, \boldsymbol{y})$ can be given as:

$$c(T, \boldsymbol{y}) = 1 + \sum_{v \in \mathcal{I}(T)} \deg_v \boldsymbol{z}_v = 1 + \sum_{v \in V_T \setminus r_T} z_{\mathrm{pa}(v)} \,. \tag{6.1}$$

where $\mathcal{I}(T)$ is the set of all non-leaf nodes of the tree T. This definition agrees with the time complexity of the ASSIGNTONODES procedure, which is $O(c(T, \boldsymbol{y}))$ if the set operations are performed in $O(1)$ time. The *training cost* for the training set $\mathcal{D}$ is then $c(T, \mathbf{Y}) = \sum_{i=1}^{n} c(T, \boldsymbol{y})$. From the perspective of the training complexity of node classifiers, the above definition of cost is only justified for learning algorithms that scale linearly in the number of training observations. There exists, however, plenty of such algorithms with a prominent example of stochastic gradient descent. The following proposition determines the upper bound for the training cost for a single observation $c(T, \boldsymbol{y})$.

Proposition 6.1. *For any tree T and vector $\boldsymbol{y}$ it holds that:*

$$c(T, \boldsymbol{y}) \leq 1 + \|\boldsymbol{y}\|_1 \cdot \mathrm{depth}_T \cdot \deg_T \,,$$

where $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$ *is the depth of the tree, and* $\deg_T = \max_{v \in V_T} \deg_v$ *is the highest degree of a node in T.*

We present the proof in the Appendix A.6. The immediate consequence of this result is the following remark which states that the training complexity of PLTs can scale logarithmically in the number of labels.

Remark 6.2. *Consider k-sparse multi-label classification (for which, $\|\boldsymbol{y}\|_1 \leq k$). For a balanced tree of constant* $\deg_T = \lambda \ (\geq 2)$ *and* $\mathrm{depth}_T = \log_\lambda m$*, the training cost is* $c(T, \boldsymbol{y}) = O(k \log m)$*.*

Interestingly, the problem of finding the optimal tree structure in terms of the training cost is NP-hard, as proven in [Busa-Fekete et al., 2019]. However, the balanced trees achieve a logarithmic approximation of the optimal tree in the number of labels.

Let us also define the expected training cost, which we will use later to bound the expected prediction cost. The *expected training cost* is

$$C_{\mathbf{P}}(T) = \mathbb{E}_{\boldsymbol{y}}\left[c(T, \boldsymbol{y})\right] = \sum_{\boldsymbol{y} \in \mathcal{Y}} c(T, \boldsymbol{y}) \mathbf{P}(\boldsymbol{y}) \,.$$

This cost can be formulated in terms of probabilities $\mathbf{P}(z_v = 1)$, $v \in V_T$, as follows:

Remark 6.3. *For any tree T and distribution $\mathbf{P}(\boldsymbol{y})$ it holds that:*

$$C_{\mathbf{P}}(T) = 1 + \sum_{v \in V_T \setminus r_T} \mathbf{P}(z_{\mathrm{pa}(v)} = 1) = 1 + \sum_{v \in V_T} \mathbf{P}(z_v = 1) \cdot \deg_v \,.$$

6.2 Prediction complexity

For the prediction cost, we use a similar definition. We define it as the number of calls to node classifiers for a single instance $\boldsymbol{x}$. Let us first consider Algorithm 3 with a threshold vector $\boldsymbol{\tau}$. Its *prediction cost $c_{\tau}(T, \boldsymbol{x})$ for a single instance* is given by:

$$c_\tau(T, \boldsymbol{x}) = 1 + \sum_{v \in V_T \setminus r_T} [\![\hat{\eta}_{pa(v)}(\boldsymbol{x}) \geq \tau_v]\!] = 1 + \sum_{v \in V_T} [\![\hat{\eta}_v(\boldsymbol{x}) \geq \tau_v]\!] \cdot \deg_v \,.$$

The *expected prediction cost* of the threshold-based algorithm is defined as

$$C_{\mathbf{P}(\boldsymbol{x}),\tau}(T) = \mathbb{E}_{\boldsymbol{x}}[c_\tau(T, \boldsymbol{x})].$$

Analogously to Proposition 6.1, we determine the upper bound for the prediction cost $c_\tau(T, \boldsymbol{x})$. To this end, let us upper bound $\sum_{j=1}^m \hat{\eta}_j(\boldsymbol{x})$ by a constant $\hat{P}$. Moreover, we assume that $\hat{\eta}_v(\boldsymbol{x})$ are properly normalized to satisfy the same requirements as true probabilities expressed in Proposition 4.1, like in (4.9). For simplicity, we set all τ_v, $v \in V_T$, to τ. Then, we can prove the following result.

Theorem 6.4. *For Algorithm 3 with all thresholds τ_v, $v \in V_T$, set to τ and any $\boldsymbol{x} \in \mathcal{X}$, we have that:*

$$c_\tau(T, \boldsymbol{x}) \leq 1 + \lfloor \hat{P}/\tau \rfloor \cdot \mathrm{depth}_T \cdot \mathrm{deg}_T \,, \tag{6.2}$$

where $\hat{P}$ is a constant upper-bounding $\sum_{j=1}^m \hat{\eta}_j(\boldsymbol{x})$, $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, and $\mathrm{deg}_T = \max_{v \in V_T} \mathrm{deg}_v$.

We present the proof of this result, and the following ones, in Appendix A.6. Similarly, as in the case of the training cost, we can conclude the logarithmic cost in the number of labels.

Remark 6.5. *For a tree of constant $\mathrm{deg}_T = \lambda (\geq 2)$ and $\mathrm{depth}_T = \log_\lambda m$, the cost of Algorithm 3 is $O(\log m)$.*

The above result can also be related to the sum of true conditional label probabilities, $\sum_{j=1}^m \eta_j(\boldsymbol{x})$. To this end, one needs to relate the training to prediction and take into account the L_1-estimation error of $\eta_j(\boldsymbol{x})$. The next theorem shows this relation in terms of expected costs.

Theorem 6.6. *Using the notation above, it holds that*

$$C_{\mathbf{P}(\boldsymbol{x}),\tau}(T) \leq \frac{1}{\tau}\left(C_{\mathbf{P}}(T) + \sum_{v \in V_T} \mathbb{E}_{\boldsymbol{x}} \left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \cdot |\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)| \right] \cdot |V_v \backslash L_v| \cdot \mathrm{deg}_v \right) - \frac{1-\tau}{\tau}.$$

This shows that the prediction cost is related to the training cost, in terms of expectations, and to the L_1 errors of node classifiers, minimized in the training under strongly proper composite loss functions. Notice that the term corresponding to the training cost represents the true $\mathbf{P}(z_v)$, while to bound the expected prediction cost we additionally take into account the relevant errors.

The analysis of Algorithm 4 which predicts the top k labels is more complex. We define the prediction cost of Algorithm 4, denoted as $c_k(T, \boldsymbol{x})$, as the number of nodes visited by the uniform-cost search procedure, i.e., the number of calls to the node classifiers. Let $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ denote the k-th highest estimated conditional probability of label The $c_k(T, \boldsymbol{x})$ cost is upper-bounded by the following expression,

$$\begin{aligned} c_k(T, \boldsymbol{x}) &\leq 1 + \sum_{v \in V_T \backslash r_T} [\![\hat{\eta}_{pa(v)}(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})]\!] \\ &= 1 + \sum_{v \in V_T} [\![\hat{\eta}_v(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})]\!] \cdot \mathrm{deg}_v \,, \end{aligned}$$

i.e., $c_k(T, \boldsymbol{x})$ is upper-bounded by $c_\tau(T, \boldsymbol{x})$, with τ equal to $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$. The actual number of visited nodes depends however on the order of nodes with the same $\hat{\eta}_v(\boldsymbol{x})$ in the priority queue guiding the search.

Additionally, we give a condition under which the cost $c_k(T, \boldsymbol{x})$ is $O(\log m)$. This result follows from similar arguments as Theorem 6.4. However, as it is suited for Algorithm 4, it gives a better understanding of the cost bound for the uniform-cost search based prediction.

Theorem 6.7. *For Algorithm 4 retrieving k top-scoring labels and any $\boldsymbol{x} \in \mathcal{X}$ we have that:*

$$c_k(T, \boldsymbol{x}) \leq 1 + (k + c - 1) \cdot \mathrm{depth}_T \cdot \mathrm{deg}_T,$$

where c, $c \geq 1$, is an integer for which $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x}) > \frac{1}{c}\sum_{i=k+1}^{m} \hat{\eta}_{\pi_i(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$, $\hat{\boldsymbol{\eta}}(\boldsymbol{x})$ denotes the vector of estimates of conditional label probabilities using the normalization (4.9), $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, *and* $\mathrm{deg}_T = \max_{v \in V_T} \mathrm{deg}_v$.

Remark 6.8. *If $c = 1$, then $c_k(T, \boldsymbol{x}) \leq 1 + k \cdot \mathrm{depth}_T \cdot \mathrm{deg}_T$.*

Remark 6.9. *For a tree of constant* $\mathrm{deg}_T = \lambda (\geq 2)$ *and* $\mathrm{depth}_T = \log_\lambda m$, *the cost of Algorithm 4 is $O(\log m)$ under the assumptions of Theorem 6.7.*

The prediction cost of Algorithm 5, implementing the beam-search-based prediction, is determined by the structure of the tree and the width B of the beam and does not depend on the values of estimated conditional probabilities of labels.

Theorem 6.10. *For Algorithm 5 with beam B and any $\boldsymbol{x} \in \mathcal{X}$, we have that:*

$$c_B(T, \boldsymbol{x}) \leq 1 + B \cdot \mathrm{deg}_T \cdot \mathrm{depth}_T$$

where $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, *and* $\mathrm{deg}_T = \max_{v \in V_T} \mathrm{deg}_v$.

Remark 6.11. *For a tree of constant* $\mathrm{deg}_T = \lambda (\geq 2)$ *and* $\mathrm{depth}_T = \log_\lambda m$, *the cost of Algorithm 5 is $O(\log m)$.*

The above bounds consider the prediction cost, not the computational complexity of prediction algorithms. The computational complexity of Algorithm 3 is linear with respect to the prediction cost, $O(c_{\tau(T,\boldsymbol{x})})$, as each visit of a node classifier is associated with a constant cost. Therefore it is logarithmic in m under certain assumptions. However, Algorithm 4 uses a priority queue in which operations require $O(\log m)$ time. Therefore, each visit of a node is not associated with $O(1)$ cost, but with $O(\log m)$ cost, and the computational complexity with respect to m of Algorithm 4 is $O(c_{\tau(T,\boldsymbol{x})} \log m) = O(\log^2 m)$, i.e., sublinear in m under certain assumptions. Moreover, in practical scenarios, the maintenance of the priority queue is almost negligible as only in the worst-case scenario its size approaches m. To assess the computational complexity of Algorithm 5 we need to consider the cost of selecting the B highest scores nodes from at most $B \cdot \mathrm{deg}_T$ estimates $\mathrm{depth}_T - 1$ times and selecting the highest scoring leaves from at most $B \cdot \mathrm{deg}_T$ estimates. We assume that this is done by sorting the $B \cdot \mathrm{deg}_T$ in $O(B \cdot \mathrm{deg}_T \log(B \cdot \mathrm{deg}_T))$. Then under assumptions regarding the tree structure, given the logarithmic depth, we get the complexity $O(\log(m) B \,\mathrm{deg}_T \log(B \,\mathrm{deg}_T) + B \,\mathrm{deg}_T \log(m))$. This is sublinear in m. Moreover, in practical scenarios, the cost of sorting $B \cdot \deg_T$ is negligible compared to the cost of evaluation of $c_B(T, \boldsymbol{x})$ classifiers, and therefore the beam width B plays an important role in controlling the computational cost of this algorithm given a fixed tree.

6.3 Memory complexity

Finally, let us shortly discuss the space complexity. The space needed for storing the final model can also be expressed in terms of the number of nodes. As the number of nodes of a label tree is upper-bounded by $2m-1$, that is, the maximum number of nodes of the tree with m leaves, the space complexity is $O(m)$. During training or prediction, there are no other structures with a higher space demand. Nevertheless, different design choices impact the space requirements of PLTs. We discuss some of them in Chapter 8.

7
Online PLT

In this chapter, we describe the online probabilistic label trees algorithm, which trains a PLT classifier online, without prior knowledge of the set of labels. This algorithm and results have been published in [Jasinska-Kobus et al., 2020c,a,d].

7.1 Online and incremental training of PLTs

A PLT model can be trained either in batch mode or incrementally. The batch algorithm has been presented in Algorithm 1 in Chapter 4. It can be easily transformed into an incremental algorithm operating sequentially on observations from $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$. To this end, we need to use an incremental learning algorithm A_{online} in the tree nodes. Such *incremental PLT* (IPLT) is given in Algorithm 6.

Algorithm 6 IPLT.TRAIN($T, A_{\text{online}}, \mathcal{D}$)

```
1: $H_T = \emptyset$                                   ▷ Initialize a set of node probabilistic classifiers
2: for each node $v \in V_T$ do                                               ▷ For each node in the tree
3:     $\hat{\eta}(v) = \text{NewClassifier}()$, $H_T = H_T \cup \{\hat{\eta}(v)\}$        ▷ Initialize its binary classifier.
4: for $i = 1 \to n$ do                              ▷ For each observation in the training sequence
5:     $(P, N) = \text{AssignToNodes}(T, \boldsymbol{x}_i, \mathcal{L}_{\boldsymbol{x}_i})$  ▷ Compute positive and negative nodes
6:     for $v \in P$ do                                                        ▷ For all positive nodes
7:         $A_{\text{online}}.\text{Update}(\hat{\eta}(v), (\boldsymbol{x}_i, 1))$                    ▷ Do a positive update with $\boldsymbol{x}_i$.
8:     for $v \in N$ do                                                      ▷ For each negative node
9:         $A_{\text{online}}.\text{Update}(\hat{\eta}(v), (\boldsymbol{x}_i, 0))$                   ▷ Do a negative update with $\boldsymbol{x}_i$.
10:    return $H_T$                                ▷ Return the set of node probabilistic classifiers
```

The above algorithm, similarly to its batch counterpart, works on a finite training set and requires a tree structure T to be given in advance. To construct T at least the number m of labels needs to be known. More advanced tree construction procedures, as discussed in Section 8.6, exploit additional information like feature values or label co-occurrence [Prabhu et al., 2018]. In all such algorithms, the tree is built in a batch mode prior to the learning of node classifiers. Here, we analyze a different scenario in which an algorithm operates on a possibly infinite sequence of training instances and the tree is constructed online, simultaneously with incremental training of node classifiers, without any prior knowledge of the set of labels or training data. We refer to such an approach as online probabilistic label trees.

Let us denote a sequence of observations by $\mathcal{S} = \{(\boldsymbol{x}_i, \mathcal{L}_{\boldsymbol{x}_i})\}_{i=1}^{\infty}$ and a subsequence consisting of the first t instances by $\mathcal{S}_t$. We refer here to labels of $\boldsymbol{x}_i$ only by $\mathcal{L}_{\boldsymbol{x}_i}$, not using the vector notation $\boldsymbol{y}_i$. This is because the number of labels m increases over time, which would also change the length of $\boldsymbol{y}_i$.[1] Furthermore, let the set of labels observed in $\mathcal{S}_t$ be denoted by $\mathcal{L}_t$, with $\mathcal{L}_0 = \emptyset$. An online algorithm returns at step t a tree structure T_t constructed over labels in $\mathcal{L}_t$ and a set of node classifiers H_t. Notice that the tree structure and the set of classifiers change in each iteration in which one or more new labels are observed. Below we discuss two properties that are desired for such an online algorithm, defined in relation to the IPLT algorithm given above.

Definition 7.1 (A proper online PLT algorithm)**.** *Let T_t and H_t be respectively a tree structure and a set of node classifiers trained on a sequence $\mathcal{S}_t$ using an online algorithm A. We say that A is a* proper online PLT algorithm, *when for any $\mathcal{S}$ and t we have that*

- *$l_j \in L_{T_t}$ iff $j \in \mathcal{L}_t$, that is, leaves of T_t correspond to all labels observed in S_t,*
- *and H_t is exactly the same as $H =$ IPLT.TRAIN$(T_t, A_{online}, \mathcal{S}_t)$, that is, node classifiers from H_t are the same as the ones trained incrementally by Algorithm 6 on $\mathcal{D} = \mathcal{S}_t$ and tree T_t given as input parameter.*

In other words, we require that whatever tree an online algorithm produces, the node classifiers should be trained the same way as the tree would be know from the very beginning of training. Thanks to that we can control the quality of each node classifier, as we are not missing any update. Moreover, since the result of a proper online PLT is the same as of IPLT, the same statistical guarantees apply to both of them.

The above definition can be satisfied by a naive algorithm that stores all observations seen so far, use them in each iteration to build a tree, and train node classifiers with the IPLT algorithm. This approach is costly in terms of both memory, used for storing S_t, and time, as all computations are run from scratch in each iteration. Therefore, we also demand an online algorithm to be space and time-efficient in the following sense.

Definition 7.2 (An efficient online PLT algorithm)**.** *Let T_t and H_t be respectively a tree structure and a set of node classifiers trained on a sequence $\mathcal{S}_t$ using an online algorithm A. Let C_s and C_t be the space and time training cost of* IPLT *trained on*

[1]The same applies to $\boldsymbol{x}_t$ as the number of features also increases. We keep however the vector notation in this case, as it does not impact the description of the algorithm.

sequence $\mathcal{S}_t$ and tree T_t. An online algorithm is an efficient online PLT algorithm *when for any S and t we have its space and time complexity to be in a constant factor of C_s and C_t, respectively.*

In this definition, we abstract from the actual implementation of IPLT. In other words, the complexity of an efficient online PLT algorithm depends directly on design choices for an IPLT. Let us recall that the training cost for a single training observation can be expressed by (6.1), as discussed in Chapter 6. By summing it over all observations in $\mathcal{S}_t$, we obtain the cost C_t of an IPLT. The space complexity is upper-bounded by $2m - 1$ (the maximum number of node models), but it also depends on the chosen type of node models and the way of storing them (see Chapter 8 for a detailed discussion on implementation choices). Let us also notice that the definition implies that the update of a tree structure has to be in a constant factor of the training cost of a single instance, given by (6.1).

7.2 Online tree building and training of node classifiers

Below we describe an online algorithm that, as we show in the next subsection, satisfies both properties defined above. It is similar to the conditional probability tree (CPT) [Beygelzimer et al., 2009a], introduced for multi-class problems and binary trees, but extends it to multi-label problems and trees of any arity. We refer to this algorithm as OPLT.

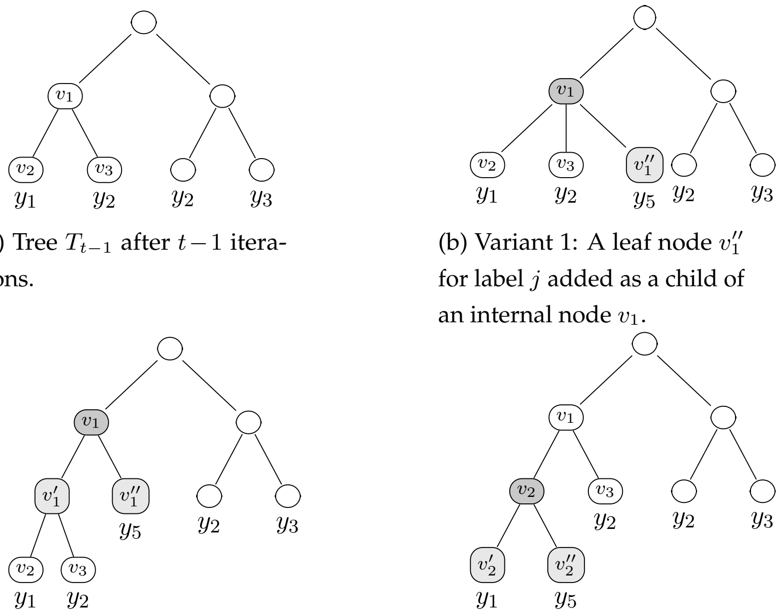
The pseudocode is presented in Algorithms 7-12. In a nutshell, OPLT processes observations from $\mathcal{S}$ sequentially, updating node classifiers. For new incoming labels it creates new nodes according to a chosen tree building policy which is responsible for the main logic of the algorithm. Each new node v is associated with two classifiers, a regular one $\hat{\eta}(v) \in H_T$, and an *auxiliary* one $\hat{\theta}(v) \in \Theta_T$, where H_T and Θ_T denote the corresponding sets of node classifiers. The task of the auxiliary classifiers is to accumulate positives updates. The algorithm uses them later to initialize classifiers in new nodes added to a tree. They can be removed if a given node will not be used anymore to extend the tree. A particular criterion for removing an auxiliary classifier depends, however, on a tree building policy.

OPLT.TRAIN, outlined in Algorithm 7, administrates the entire process. It first initializes a tree with a root node r_T only and creates two corresponding classifiers, $\hat{\eta}(v_{r_T})$ and $\hat{\theta}(v_{r_T})$. Notice that the root has both classifiers initialized from the very beginning without a label assigned to it. Thanks to this, the algorithm can properly estimate the probability of $\mathbf{P}(\mathcal{L}_{\boldsymbol{x}} = \emptyset \,|\, \boldsymbol{x})$. Observations from $\mathcal{S}$ are processed sequentially in the main loop of OPLT.TRAIN. If a new observation contains one or more new labels then the tree structure is appropriately extended by calling UPDATETREE. The node classifiers are updated in UPDATECLASSIFIERS. After each iteration t, the algorithm sends H_T along with the tree structure T, respectively as H_t and T_t, to be used outside the algorithm for prediction tasks.

We assume that tree T along with sets of its all nodes V_T and leaves L_T, as well as sets of classifiers H_T and Θ_T, are accessible to all subroutines discussed below.

Algorithm 7 OPLT.TRAIN($\mathcal{S}, A_{\text{online}}, A_{\text{policy}}$)

```
1: $r_T$ = NEWNODE(), $V_T = \{r_T\}$                                ▷ Create the root of the tree
2: $\hat{\eta}(r_T)$ = NEWCLASSIFIER(), $H_T = \{\hat{\eta}_T(r_T)\}$   ▷ Initialize a new classifier in the root
3: $\hat{\theta}(r_T)$ = NEWCLASSIFIER(), $\Theta_T = \{\theta(r_T)\}$   ▷ Initialize a root auxiliary classifier
4: for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}) \in \mathcal{S}$ do                         ▷ For each observation in $\mathcal{S}$
5:     if $\mathcal{L}_{\boldsymbol{x}_t} \setminus \mathcal{L}_{t-1} \neq \emptyset$ then               ▷ If the observation contains new labels
6:         UPDATETREE($\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}, A_{\text{policy}}$)                  ▷ Add them to the tree
7:     UPDATECLASSIFIERS($\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t}, A_{\text{online}}$)               ▷ Update the classifiers
8:     send $H_t, T_t = H_T, V_T$          ▷ Send the node classifiers and the tree structure.
```



(a) Tree T_{t-1} after $t-1$ iterations.

(b) Variant 1: A leaf node v_1'' for label j added as a child of an internal node v_1.

(c) Variant 2: A leaf node v_1'' for label j and an internal node v_1' (with all children of v_1 reassigned to it) added as children of v_1.

(d) Variant 3: A leaf node v_2'' for label j and a leaf node v_2' (with a reassigned label of v_2) added as children of v_2.

Figure 7.1: Three variants of tree extension for a new label j.

Algorithm 8, UPDATETREE, builds the tree structure. It iterates over all new labels from $\mathcal{L}_{\boldsymbol{x}}$. If there were no labels in the sequence $\mathcal{S}$ before, the first new label taken from $\mathcal{L}_{\boldsymbol{x}}$ is assigned to the root note. Otherwise, the tree needs to be extended by one or two nodes according to a selected tree building policy. One of these nodes is a leaf to which the new label will be assigned. There are in general three variants of performing this step illustrated in Figure 7.1. The first one is selecting an internal node v whose number of children is lower than the accepted maximum, and adding to it a child node v'' with the new label assigned to it. In the second one, two new child nodes, v' and v'', are added to a selected internal node v. Node v' becomes a new parent of child nodes of the selected node v, that is, the subtree of v is moved down by one level. Node v'' is a leaf with the new label assigned to it. The third variant is a modification of the second one. The difference is that the selected node v is a leaf node. Therefore there are no children nodes to be moved to v', but label of v is reassigned to v'. The A_{policy} method encodes the tree building policy, that is, it decides which of the three variants to follow and selects the node v. The additional node v' is inserted by the INSERTNODE method. Finally, a leaf node is added by the ADDLEAF method. We discuss the three methods in more detail below.

Algorithm 8 OPLT.UPDATETREE($\boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}, A_{\text{policy}}$)

```
1: for $j \in \mathcal{L}_{\boldsymbol{x}} \setminus \mathcal{L}_{t-1}$ do          ▷ For each new label in the observation
2:     if $\mathcal{L}_T$ is $\emptyset$ then                          ▷ If no labels have been seen so far
3:         LABEL($r_T$) $= j$                                 ▷ Assign label $j$ to the root node
4:     else                                                   ▷ If there are already labels in the tree.
5:         $v$, $insert = A_{\text{policy}}(\boldsymbol{x}, j, \mathcal{L}_{\boldsymbol{x}})$      ▷ Select a variant of extending the tree
6:         if $insert$ then INSERTNODE($v$)                  ▷ Insert an additional node if needed.
7:         ADDLEAF($j, v$)                                    ▷ Add a new leaf for label $j$.
```

A_{policy} returns the selected node v and a Boolean variable $insert$ which indicates whether an additional node v' has to be added to the tree. For the first variant, v is an internal node and $insert$ is set to false. For the second variant, v is an internal node and $insert$ is set to true. For the third variant, v is a leaf node and $insert$ is set to true. In general, the policy can be guided by $\boldsymbol{x}$, current label j, and set $\mathcal{L}_{\boldsymbol{x}}$ of all labels of $\boldsymbol{x}$. As an instance of the tree building policy, we consider, however, a much simpler method presented in Algorithm 9. It creates a b-ary complete tree. In this case, the selected node is either the leftmost internal node with the number of children less than b or the leftmost leaf of the lowest depth. The $insert$ variable is then $false$ or $true$, respectively. So, only the first and third variants occur here. Notice, however, that this policy can be efficiently performed in amortized constant time per label if the complete tree is implemented using a dynamic array with doubling. Nevertheless, more advanced and computationally complex policies can be applied. As mentioned before, the complexity of this step should be at most proportional to the complexity of updating the node classifiers for one label, that is, it should be proportional to the depth of the tree.

Algorithm 9 BUILDCOMPLETETREE(b)

```
1: array $= T$.array ▷ Let nodes of complete tree $T$ be stored in a dynamic array $T$.array
2: $s =$ array.length                                  ▷ Read the number of nodes in $T$
3: $pa = \lceil \frac{s}{b} \rceil - 1$   ▷ Index of a parent of a next added node; the array is indexed from 0
4: $v =$ array($pa$)                                       ▷ Get the parent node
5: return $v$, ISLEAF($v$)                       ▷ Return the node and whether it is a leaf.
```

The INSERTNODE and ADDLEAF procedures involve specific operations concerning the initialization of classifiers in the new nodes. INSERTNODE is given in Algorithm 10. It inserts a new node v' as a child of the selected node v. If v is a leaf then its label is reassigned to the new node. Otherwise, all children of v become the children of v'. In both cases, v' becomes the only child of v.

Figure 7.1 illustrates inserting v' as either a child of an internal node (c) or a leaf node (d). Since, the node classifier of v' aims at estimating $\eta(\boldsymbol{x}, v')$, defined as $\mathbf{P}(z_{v'} = 1 \mid z_{\mathrm{pa}(v')} = 1, \boldsymbol{x})$, its both classifiers, $\hat{\eta}(v')$ and $\hat{\theta}(v')$, are initialized as copies (by calling the COPY function) of the auxiliary classifier $\hat{\theta}(v)$ of the parent node v. Recall that the task of auxiliary classifiers is to accumulate all positive updates in nodes, so the conditioning $z_{\mathrm{pa}(v')} = 1$ is satisfied in that way.

Algorithm 10 OPLT.INSERTNODE(v)

```
1: $v' = $ NEWNODE(), $V_T = V_T \cup \{v'\}$   ▷ Create a new node and add it to the tree nodes
2: if ISLEAF($v$) then   ▷ If node $v$ is a leaf
3:     LABEL($v'$) = LABEL($v$), LABEL($v$) = NULL   ▷ Reassign label of $v$ to $v'$
4: else   ▷ Otherwise
5:     $\mathrm{Ch}(v') = \mathrm{Ch}(v)$   ▷ All children of $v$ become children of $v'$
6:     for $v_{\mathrm{ch}} \in \mathrm{Ch}(v')$ do $\mathrm{pa}(v_{\mathrm{ch}}) = v'$   ▷ And $v'$ becomes their parent
7: $\mathrm{Ch}(v) = \{v'\}$, $\mathrm{pa}(v') = v$   ▷ The new node $v'$ becomes the only child of $v$
8: $\hat{\eta}(v') = $ COPY($\hat{\theta}(v)$), $H_T = H_T \cup \{\hat{\eta}(v')\}$   ▷ Create a classifier.
9: $\hat{\theta}(v') = $ COPY($\hat{\theta}(v)$), $\Theta_T = \Theta_T \cup \{\hat{\theta}(v')\}$   ▷ And an auxiliary classifier.
```

Algorithm 11 outlines the ADDLEAF procedure. It adds a new leaf node v'' for label j as a child of node v . The classifier $\hat{\eta}(v'')$ is created as an "inverse" of the auxiliary classifier $\hat{\theta}(v)$ from node v. More precisely, the INVERSECLASSIFIER procedure creates a wrapper inverting the behavior of the base classifier. It predicts $1 - \hat{\eta}$, where $\hat{\eta}$ is the prediction of the base classifier, and flips the updates, that is, positive updates become negative and negative updates become positive. Finally, the auxiliary classifier $\hat{\theta}(v'')$ of the new leaf node is initialized.

Algorithm 11 OPLT.ADDLEAF(j, v)

```
1: $v'' = $ NEWNODE(), $V_T = V_T \cup \{v''\}$   ▷ Create a new node and add it to the tree nodes
2: $\mathrm{Ch}(v) = \mathrm{Ch}(v) \cup \{v''\}$, $\mathrm{pa}(v'') = v$   ▷ Add this node to children of $v$.
3: LABEL($v''$) = $j$   ▷ Assign label $j$ to the node $v''$
4: $\hat{\eta}(v'') = $ INVERSECLASSIFIER($\hat{\theta}(v)$), $H_T = H_T \cup \{\hat{\eta}(v'')\}$   ▷ Initialize a classifier for $v''$
5: $\hat{\theta}(v'') = $ NEWCLASSIFIER(), $\Theta_T = \Theta_T \cup \{\hat{\theta}(v'')\}$   ▷ Initialize an auxiliary classifier for $v''$
```

The final step in the main loop of OPLT.TRAIN updates the node classifiers. The regular classifiers, $\hat{\eta}(v) \in H_T$, are updated exactly as in IPLT.TRAIN given in Algorithm 6. The auxiliary classifiers, $\theta(v) \in \Theta_T$, are updated only in positive nodes according to their definition and purpose.

Algorithm 12 OPLT.UPDATECLASSIFIERS($\boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}, A_{\mathrm{online}}$)

```
1: $(P, N) = $ ASSIGNTONODES($T, \boldsymbol{x}, \mathcal{L}_{\boldsymbol{x}}$)   ▷ Compute its positive and negative nodes
2: for $v \in P$ do   ▷ For all positive nodes
3:     $A_{\mathrm{online}}$.UPDATE($\hat{\eta}(v), (\boldsymbol{x}, 1)$)   ▷ Update classifiers with a positive update with $\boldsymbol{x}$.
4:     if $\hat{\theta}(v) \in \Theta$ then   ▷ Update auxiliary classifier if it exists.
5:         $A_{\mathrm{online}}$.UPDATE($\hat{\theta}(v), (\boldsymbol{x}, 1)$)   ▷ With a positive online update with $\boldsymbol{x}_i$.
6: for $v \in N$ do   ▷ For each negative node
7:     $A_{\mathrm{online}}$.UPDATE($\hat{\eta}(v), (\boldsymbol{x}, 0)$)   ▷ Update classifiers with a negative update with $\boldsymbol{x}$.
```

7.3 Theoretical analysis of OPLT

The OPLT algorithm has been designed to satisfy the properness and efficiency property of online probabilistic label trees. The theorem below states this fact formally.

Theorem 7.3. OPLT *is a proper and efficient* OPLT *algorithm.*

The proof is quite technical and we present it in Appendix A.7. To show the properness, it uses induction for both the outer and inner loop of the algorithm, where the outer loop iterates over observations $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$, while the inner loop over new labels in $\mathcal{L}_{\boldsymbol{x}_t}$. The key elements used to prove this property are the use of the auxiliary classifiers and the analysis of the three variants of the tree structure extension. The efficiency is proved by noticing that each node has two classifiers, and the algorithm creates and updates no more than one additional classifier per node comparing to IPLT. Moreover, any node selection policy which cost is proportional to the cost of updating IPLT classifiers for a single label meets the efficiency requirement. Particularly, the policy building a complete tree presented above satisfies this constraint.

The OPLT algorithm aims at constructing the node classifiers in such a way that its properness can be met by a wide range of tree building policies. The naive complete tree policy was introduced mainly for ease of presentation. One can, for example, easily adapt and further extend the policy originally used in CPT [Beygelzimer et al., 2009a]. In short, the CPT policy selects a node v which trade-offs balancedness of the tree and a fit of $\boldsymbol{x}$, that is, the value of $\hat{\eta}_v(\boldsymbol{x})$. Since it works with binary trees only, the policy uses solely the third variant of the tree extension. Moreover, it was designed for multi-class problems. In [Jasinska-Kobus et al., 2020c] we have considered such an extension. From this point of view, the presented framework significantly extends CPT. It solves both types of problems, multi-class and multi-label, and can be used with more advanced policies that exploit all three variants of the tree extension.

8 Implementation

In this chapter, we discuss the possibilities of implementation of PLTs. We first recall several existing packages implementing this approach, and briefly characterize their design. Then we analyze possible design choices in detail.

8.1 Popular PLT packages

There are several packages implementing the PLT model, for example, XMLC-PLT [Jasinska et al., 2016],[1] PLT-VW,[2] PARABEL [Prabhu et al., 2018],[3] EXTREMETEXT [Wydmuch et al., 2018],[4] ATTENTIONXML [You et al., 2019],[5] BONSAI [Khandagale et al., 2019],[6] or NAPKINXC [Jasinska-Kobus et al., 2020b].[7] In the following sections, we discuss the differences between them in terms of training of node classifiers, dealing with sparse and dense features, efficient prediction, tree structure learning, and ensembling. Table 8.1 summarizes the differences between the discussed implementations. At the end of this chapter we also shortly discuss a different approach to obtain $\hat{\eta}_v(\boldsymbol{x})$, which uses multi-class probability estimation instead of binary probability estimation.

[1]https://github.com/busarobi/XMLC
[2]https://github.com/VowpalWabbit/vowpal_wabbit
[3]http://manikvarma.org/code/Parabel/download.html
[4]https://github.com/mwydmuch/extremeText
[5]https://github.com/yourh/AttentionXML
[6]https://github.com/xmc-aalto/bonsai
[7]https://github.com/mwydmuch/napkinXC

Implementation	node classifiers	represen-tation	prediction	tree structure	ensem-bling
XMLC-PLT/ PLT-VW	online	sparse	online/ unif.-cost search	complete tree based on freq.	no
PARABEL	batch	sparse	batch/ beam search	balanced h. 2-means	yes
BONSAI	batch	sparse	batch/ beam search	unbalanced h. k-means	yes
EXTREMETEXT	online	dense	online/ unif.-cost search.	h. k-means	yes
ATTENTIONXML	online & leveled	dense	batch & leveled/ beam search	shallow h. k-means	no
NAPKINXC	both	both	online/ unif.-cost search, thresholds-based	any	yes

Table 8.1: Comparison of different implementations of the PLT model in terms of node classifiers (online, batch, or both), representation of features (sparse, dense, both), prediction algorithm (online, batch, leveled/beam search, uniform cost search, thresholds-based), tree structure learning (complete tree based on frequencies, hierarchical k means, their shallow variant, or any), ensembling (yes, no). All the options are described in text.

8.2 Training of node classifiers

Given the tree structure, the node classifiers of PLTs can be trained either in online or batch mode. Both training modes have their pros and cons. The batch variant, implemented for example in PARABEL, can benefit from using well-known batch solvers, such as LIBLINEAR [Fan et al., 2008]. These variants are relatively easy to train and achieve high predictive performance. Moreover, each model can be trained independently which enables a simple parallelization of training.

In turn, the online variant, such as XMLC-PLT, PLT-VW, or EXTREMETEXT, can be applied to stream data. However, all those implementations demand a tree structure to be known prior to the training of node classifiers. Nevertheless, the online node learners give the possibility of combining them with the online tree construction, as discussed in the previous section. Moreover, they can benefit from using deep networks to learn a complex representation of input instances. Parallelization can be performed similarly as in the case of batch models. If additional conflicts during updates of the node models are accepted, we can apply parallelization on the level of single observation as in EXTREMETEXT. Each thread consumes a part of the training observations and updates the model allocated in shared memory.

NAPKINXC follows a modular design, therefore it can be used with either batch or online node learners. In the latter case, it has an implemented functionality of collaborating with external learners to exchange *forward* and *backward* signals. It also supports the online tree construction.

8.3 Sparse features

For problems with sparse features, such as text classification, PLTs can be efficiently implemented using different approaches. One is training a dense model using the sparse observations efficiently transformed into dense ones when necessary, and then storing the model in a sparse representation. For example, in the popular LIBLINEAR package, the observations are kept in the original sparse format, but for efficiency, internal operations use dense vectors, and the resulting model is also dense. This model can be stored as a sparse one, either if it was trained with L_1 regularization, or after removal of weights being close to zero in the case of L_2 regularization. Such an approach is applied in DISMEC [Babbar and Schölkopf, 2017]. Since the node classifiers can be trained independently, the runtime memory of this approach can also be optimized.

An alternative is sparse learning algorithms. Such algorithms follow usually the online/incremental learning paradigm, in which training observations are processed sequentially one-by-one. Examples of such algorithms are Fobos [Duchi and Singer, 2009] or AdaGrad [Duchi et al., 2011]. To store and update weights one may use either hash maps or feature hashing [Weinberger et al., 2009]. The latter, implemented for example in the popular VOWPAL WABBIT package [Langford et al., 2007], allocates constant memory space for features weights. Since the allocated space might be too small, conflicts can exist between different weights. They are not resolved, that is, the weight is shared by all conflicting features. If used with PLTs, the allocated memory may be shared by all node models. In the case of hash maps, one needs to reallocate the memory if the map is nearly full. NAPKINXC follows the ROBIN HOOD HASHING [Celis et al., 1985] which allows doing very efficient insert and find operations, having at the same time minimal memory overhead. It uses open addressing, but compared to hash maps with linear and quadratic probing, it significantly reduces the expected average and maximum probe lengths. This is achieved by shifting the keys around in such a way that all keys stay reasonably close to the slot they hash to. When a new element is inserted, if the probe length for the existing element is less than the current probe length for the element being inserted, ROBIN HOOD swaps the two elements and continues the procedure. This results in much lower average probe lengths as well as its variance. This also allows using a straightforward lookup algorithm that ignores empty slots and keeps looking for a key until it reaches the known maximum probe length for the whole table. This property also allows using high load factors (higher than 0.9). Since it uses open addressing, the memory usage for the whole map is very close to the memory needed to store its content as a sparse vector.

Interestingly, for sparse data, the sparsity of weights increases with the depth of a tree. This implies a significant reduction of space of the final PLT model. Paradoxically, this reduction can be the largest in the case of binary trees, although the number of nodes is the highest in this case, (equal to $2m - 1$, being as much as twice the number of models in the 1-VS-ALL approach). This is because the models use only non-zero features of the sibling nodes, and there are only two such nodes in binary trees. No other features are needed to build the corresponding classifiers.

8.4 Dense features

PLTs can also work with dense features. However, the dimensionality of the feature space cannot be too high, otherwise, the memory used for models would be too large. The dense representation is usually connected with deep or shallow neural networks.

One possibility is to use pre-trained embeddings. For text classification, one can use word representations trained by WORD2VEC [Mikolov et al., 2013] or GLOVE [Pennington et al., 2014] on large text corpora. The document representation can be then created from the word representations in many ways, for example, by taking the average, maximum or minimum value of each element of the embeddings [De Boom et al., 2016]. Alternatively, the word embeddings can be trained simultaneously with the node classifiers, similarly to in FASTTEXT [Joulin et al., 2017]. This approach is adopted in EXTREMETEXT. Another option is to initialize the network with the pre-trained embeddings and then update all the parameters of both the node classifiers and word representations.

To improve the representation of the documents, instead of a simple aggregation over words, one can use word embeddings and a more advanced deep architecture, such as LSTM [Hochreiter and Schmidhuber, 1997] or the text-based convolution neural network [Liu et al., 2017]. PLTs can be used as the output layer of such architecture. However, in the case of GPU-based training, one may not observe any speedup compared to a simple linear output layer, as matrix multiplication using GPUs is efficient. Nevertheless, in the case of complex and memory-intensive approaches, PLTs can be used to decompose the problem in such a way that computation is performed level-by-level in a tree, with additional decomposition possible on a given level. All the layers except the PLT one (output of the network) are initialized using the trained values from the preceding level. This idea is followed by ATTENTIONXML [You et al., 2019].

8.5 Prediction

The top-k prediction, discussed in Section 4.3 as Algorithm 4, is a variant of the uniform-cost search. It is used in XMLC-PLT, EXTREMETEXT, and NAPKINXC. It is very efficient for online prediction, as we discussed in Chapter 6. The algorithm also allows extending k at any time, without re-running the procedure. If one does not need to change k, the algorithm can be improved by adding to the priority queue only the nodes with probability greater than the probability of the

k-th top leaf already added to the queue. Also, the threshold-based prediction Algorithm 3, or beam-search-based Algorithm 5, can be easily implemented in the online setting. However, in the case of sparse models, online prediction requires the models to be stored either in hash maps or by feature hashing to allow efficient random access to model weights.

Unpacking a sparse model to dense representation can be very costly in the case of online prediction. However, if a node classifier is unpacked only once for a number of test instances, for sufficiently large batches, the dense representation may be beneficial, as it allows efficient computation of the dot products. Such batch variants of prediction methods need to be used in PARABEL, due to its model representation. PARABEL uses batch beam search [Prabhu et al., 2018]. The batch version of the uniform-cost search [Jasinska, 2018] is given in Appendix B.1. Also, the threshold-based prediction Algorithm 3 could easily be implemented in the batch mode. Batch beam search unpacks each node classifier at most once, while uniform-cost search may need to unpack a node several times for different subsets of the batch. Therefore in certain cases, batch beam search may be more computationally efficient than batch uniform-cost search. In the case of memory-intensive deep models, such as ATTENTIONXML, the prediction is usually performed level-by-level in batches, similarly to training, and, therefore, it uses batch beam search. Nevertheless, the batch variant of uniform-cost search also could be used in such setting. If models are dense and they can be all loaded to the main memory, then both online and batch methods perform similarly in terms of computational times.

Finally, notice that uniform-cost search finds the exact top-k labels with the highest estimated conditional probabilities, visiting the least possible number of nodes (required to ensure that the predictions are exact). On the other hand, beam search is an approximate method and it may not find the actual top-k labels. Therefore it may suffer regret for precision@k [Zhuo et al., 2020], while uniform-cost search based prediction gives a non-regret prediction. Moreover, given a beam size large enough to reach predictive performance competitive to uniform-cost search, beam search may visit more nodes than necessary.

8.6 Tree structure

The tree structure of a PLT is a crucial modeling decision. The theoretical results from Chapter 5 concerning the vanishing regret of PLT hold regardless of the tree structure, however, this theory requires the regret of the node classifiers also to vanish. In practice, we can only estimate the conditional probabilities in the nodes, therefore the tree structure does indeed matter as it affects the difficulty of the node learning problems. In Chapter 7, we already discussed the problem of building a tree in the online setting. Here, we focus on batch approaches which assume that labels are known. The original PLT paper [Jasinska et al., 2016] uses simple complete trees with labels assigned to leaves according to their frequencies. Another option, routinely used in HSM [Joulin et al., 2017], is the Huffman tree built over the label frequencies. Such a tree takes into account the computational complexity by putting the most frequent labels close to the root. This approach has been further extended to optimize GPU operations in [Grave et al., 2017]. Unfortunately, for multi-label classification the Huffman tree is no longer optimal in terms of computational cost. As already mentioned in Chapter 6, the problem of optimization of the tree structure with respect to the computational complexity of PLTs has been analyzed by Busa-Fekete et al. [2019]. Furthermore, Huffman trees ignore the statistical properties of the tree structure. There exist, however, other methods that focus on building a tree with high overall accuracy.

The method of [Prabhu et al., 2018] performs a simple top-down hierarchical clustering. Each label in this approach is represented by a profile vector being an average of the training vectors tagged by this label. Then the profile vectors are clustered using balanced k-means which divides the labels into two or more clusters of approximately the same size. This procedure is then repeated recursively until the clusters are smaller than a given value (for example, 100). The nodes of the resulting tree are then of various arities. The internal nodes up to the pre-leaf nodes have k children, but the pre-leaf nodes are usually of higher arity. Thanks to this clustering, similar labels are close to each other in the tree. Moreover, the tree is balanced, so its depth is logarithmic in terms of the number of labels. Variants of this method have been used in PARABEL [Prabhu et al., 2018] (with $k = 2$), EXTREMETEXT [Wydmuch et al., 2018], NAPKINXC [Jasinska-Kobus et al., 2020a], BONSAI TREES [Khandagale et al., 2019] and ATTENTIONXML [You et al., 2019]. The two latter algorithms promote shallow trees, that is, trees of a much higher arity.

8.7 Ensemble of PLTs

Various ensemble techniques, such as bagging, are routinely applied with tree-based learners. A simple ensemble approach can also be implemented for PLTs. To produce a diverse ensemble, one can use several PLT instances with different tree structures, but sharing the feature space. Such various tree structures can be obtained using k-means-based top-down hierarchical clustering several times with different initialization. Depending on the tree structure, the accuracy of a single PLT for specific labels may vary. Thus the aggregation of predictions of this diverse pool of PLTs should lead to improvement of the overall predictive performance.

There exist various aggregation strategies. Consider top-k prediction. PARABEL for a single tree uses beam search with beam B. Then, the predictions of trees, each consisting of B (or $B \deg_T$) predictions of labels with their probability estimates, are averaged and sorted. However, such inference with multiple PLTs can be performed with another strategy, which can be efficiently implemented in

NAPKINXC. First, each tree is queried for its top-k predictions, obtained using uniform-cost search. If a label does not have a probability estimate from a given tree (it is not included in the top-k predictions), then the missing estimate is computed by traversing a single path corresponding to the label in the relevant tree. Finally, the average predictions for labels are computed. The good trade-off between the improvement of the predictive performance and the required computational resources is usually obtained by about 3 or 5 trees.

8.8 Node probabilities via multi-class classification

So far we assumed that each node $v \in V$ is associated with a binary probabilistic classifier that estimates $\eta_v(\boldsymbol{x})$. As already mentioned in Section 4.3, this may require the additional normalization step (4.9), as all models of siblings nodes are trained independently. To avoid this problem, one can train a joint multi-class classifier over the sibling nodes. Let $v \in V$ be a parent of the sibling nodes $\mathrm{Ch}(v)$. Then, the class labels of the multi-class problem correspond to binary codes of vector $\boldsymbol{c}$ whose elements correspond to $z_{v'}$, $v' \in \mathrm{Ch}(v)$. The classifier estimates $\mathbf{P}(\boldsymbol{c} \,|\, \boldsymbol{x}, z_v = 1)$, for all $\boldsymbol{c} \in \{0,1\}^{|\mathrm{Ch}(v)|}$. Probability $\eta_{v'}(\boldsymbol{x})$, for $v' \in \mathrm{Ch}(v)$, is obtained by proper marginalization of the multi-class distribution over vectors $\boldsymbol{c}$:

$$\eta_{v'}(\boldsymbol{x}) = \sum_{z_{v'}=1} \mathbf{P}(\boldsymbol{c} \,|\, \boldsymbol{x}, z_v = 1)\,.$$

This approach has been proposed in PARABEL [Prabhu et al., 2018]. However, it can be applied only to trees of small arity, as the number of class labels grows exponentially with the number of sibling nodes.

9 Empirical validation of PLTs

In this chapter, we present a wide empirical study performed to comprehensively evaluate the described algorithms and theoretical findings. In the first part of the study, we analyze different design choices for PLTs. In the next part, we evaluate PLTs in terms of Hamming loss, micro- and macro-F measure, to verify our theoretical results concerning generalized performance metrics. Later, we empirically confirm the suboptimality of hierarchical softmax with pick-one-label heuristic. The next experiment studies the performance of the fully online variant of PLTs, in which both node classifiers and tree structure are built incrementally on a sequence of training observations. Finally, we compare PLTs to relevant state-of-the-art algorithms.

9.1 Experimental setting

In the experiments evaluating the predictive performance, we mainly report the results in terms of precision@k, as this is the most used metric in XMLC area. Moreover, as shown in Chapter 5, PLTs are well-suited for this metric. To evaluate the computational performance, we also present training and test times, as well as memory consumption. Whenever it is necessary, we repeat the experiment 5 times to eliminate the impact of the randomness of algorithms. In such cases, we report the mean performance along with standard errors. All computations have been conducted on an Intel Xeon E5-2697 v3 2.60GHz (14 cores) machine with 128GB RAM.[1] In most experiments, we use TF-IDF versions of the real-word benchmark data sets from the Extreme Classification Repository [Bhatia et al., 2016],[2] for which we use the original train and test splits. Table 9.1 gives basic statistics of the data sets.

[1]Computational experiments have been performed in Poznan Supercomputing and Networking Center.

[2]`http://manikvarma.org/downloads/XC/XMLRepository.html`

Dataset	$\dim \mathcal{X}$	$\dim \mathcal{Y}$ (m)	N_{train}	N_{test}	avg. $\lvert\mathcal{L}_{\boldsymbol{x}}\rvert$
EurLex-4K	5000	3993	15539	3809	5.31
AmazonCat-13K	203882	13330	1186239	306782	5.04
Wiki10-30K	101938	30938	14146	6616	18.64
DeliciousLarge-200K	782585	205443	196606	100095	75.54
WikiLSHTC-325K	1617899	325056	1778351	587084	3.19
WikipediaLarge-500K	2381304	501070	1813391	783743	4.77
Amazon-670K	135909	670091	490449	153025	5.45
Amazon-3M	337067	2812281	1717899	742507	36.17

Table 9.1: The number of unique features, labels, observations in train and test splits, and the average number of true labels per observation in the benchmark data sets.

9.2 PLTs with different design choices

We analyze different design choices for PLTs. To this end, we use NAPKINXC, as it allows us to do experiments with different settings. However, whenever a given configuration agrees with an existing PLT implementation, we use this one in the experiment. This is the case of PARABEL and EXTREMETEXT. The former uses a dual coordinate descent method from LIBLINEAR with squared hinge loss to train node classifiers. It uses weight pruning at threshold 0.1, that is, it sets model weights lower than 0.1 to zero. The prediction algorithm is based on the beam search. EXTREMETEXT is built over FASTTEXT [Grave et al., 2017]. It uses dense representation, shared by all nodes, which is a result of a 1-layer network implementing the CBOW architecture [Mikolov et al., 2013]. This representation is trained along with the node models using stochastic gradient descent with L_2 regularization and logistic loss. Both implementations use hierarchical k-means clustering. PARABEL uses $k = 2$, while EXTREMETEXT allows using different values of k. Both use pre-leaves of high degree equal to 100. In the experiment, we do not use ATTENTIONXML, as it uses a complex deep architecture requiring GPUs and requires raw textual data. By comparing the results from the original paper [You et al., 2019], we admit that it achieves the best results among PLT-based approaches. Nevertheless, in this study, we focus on efficient CPU implementations and the TF-IDF versions of the benchmark data sets.

We start with a comparison of batch and incremental learning of node classifiers. For both, we use logistic and squared hinge loss. Next, we compare different methods of prediction: uniform-cost search based and beam search based, both in online and batch implementations. We then compare training and prediction with sparse and dense representation. In the next experiment, we analyze different tree-building strategies. Finally, we check the impact of ensembling.

Optimizer	p@1 [%]	p@3 [%]	p@5 [%]	T_{train} [h]	T/N_{test} [ms]	M_{size} [GB]
			EurLex-4K			
NXC-B,log	**80.51±0.16**	65.65±0.40	53.33±0.68	0.02±0.00	0.39±0.03	0.02±0.00
NXC-B,s.h.	80.17±0.27	65.33±0.53	53.01±0.87	**0.01±0.00**	**0.24±0.02**	**0.00±0.00**
NXC-I,log	80.43±0.09	**66.08±0.26**	**53.87±0.58**	0.01±0.00	0.25±0.02	0.05±0.00
NXC-I,s.h.	78.72±0.18	61.54±0.34	48.09±0.51	0.01±0.00	0.33±0.03	0.03±0.00
			AmazonCat-13K			
NXC-B,log	93.04±0.02	78.44±0.02	63.70±0.02	0.72±0.02	0.32±0.03	0.35±0.00
NXC-B,s.h.	92.40±0.04	78.49±0.02	63.88±0.02	0.29±0.00	**0.19±0.00**	**0.19±0.00**
NXC-I,log	**93.23±0.02**	**78.76±0.03**	**64.05±0.02**	0.17±0.00	0.32±0.02	0.72±0.00
NXC-I,s.h.	92.62±0.05	76.39±0.06	60.67±0.06	**0.16±0.00**	0.38±0.01	0.55±0.00
			Wiki10-30K			
NXC-B,log	85.36±0.09	73.90±0.07	63.84±0.07	0.21±0.00	5.35±0.32	0.58±0.00
NXC-B,s.h.	84.17±0.10	72.43±0.10	63.12±0.04	0.11±0.00	**2.87±0.08**	**0.06±0.00**
NXC-I,log	84.92±0.10	**74.52±0.09**	**65.29±0.04**	0.11±0.00	5.24±0.42	0.91±0.00
NXC-I,s.h.	**85.64±0.10**	70.37±0.09	59.43±0.13	**0.10±0.00**	7.19±0.06	0.35±0.00
			DeliciousLarge-200K			
NXC-B,log	**49.55±0.05**	**43.08±0.03**	**39.90±0.02**	**2.58±0.15**	**9.89±0.89**	**0.95±0.00**
NXC-B,s.h.	46.30±0.07	39.76±0.08	36.54±0.07	5.51±0.29	10.07±0.23	1.82±0.00
NXC-I,log	45.27±0.06	38.26±0.03	34.88±0.02	2.97±0.03	11.51±0.57	15.05±0.00
NXC-I,s.h.	45.29±0.34	38.15±0.50	34.44±0.58	3.13±0.10	27.11±1.55	9.59±0.00
			WikiLSHTC-325K			
NXC-B,log	61.96±0.03	40.77±0.02	30.19±0.02	2.95±0.15	1.77±0.11	2.73±0.00
NXC-B,s.h.	**62.78±0.03**	**41.17±0.02**	**30.25±0.02**	1.60±0.06	**0.86±0.06**	**0.97±0.00**
NXC-I,log	60.99±0.04	39.85±0.02	29.50±0.01	1.52±0.00	2.44±0.02	4.93±0.00
NXC-I,s.h.	59.55±0.05	37.33±0.04	27.02±0.03	**1.41±0.05**	1.70±0.17	3.64±0.00
			WikipediaLarge-500K			
NXC-B,log	66.20±0.05	47.14±0.02	36.83±0.01	16.10±0.44	6.67±0.23	8.89±0.00
NXC-B,s.h.	**66.77±0.08**	**47.63±0.04**	**36.94±0.02**	9.48±0.33	**2.86±0.07**	**1.78±0.00**
NXC-I,log	65.68±0.15	46.62±0.09	36.52±0.06	**8.11±0.18**	7.86±0.19	19.11±0.00
NXC-I,s.h.	65.05±0.08	44.35±0.03	33.74±0.05	8.22±0.06	6.66±0.06	24.71±0.00
			Amazon-670K			
NXC-B,log	43.54±0.01	38.71±0.02	35.15±0.03	0.56±0.00	4.13±0.28	2.26±0.00
NXC-B,s.h.	43.31±0.03	38.19±0.03	34.31±0.03	**0.40±0.01**	**1.32±0.08**	**0.63±0.00**
NXC-I,log	**43.82±0.01**	**38.88±0.03**	**35.31±0.03**	0.42±0.00	5.93±0.11	6.22±0.00
NXC-I,s.h.	41.46±0.02	36.16±0.04	32.34±0.03	0.41±0.00	2.08±0.17	5.26±0.00
			Amazon-3M			
NXC-B,log	46.09±0.02	43.11±0.01	40.98±0.01	7.07±0.56	3.26±0.08	20.84±0.00
NXC-B,s.h.	**46.23±0.01**	**43.48±0.01**	**41.41±0.01**	5.44±0.13	**1.96±0.05**	**9.86±0.00**
NXC-I,log	43.61±0.12	40.44±0.09	38.32±0.06	**4.44±0.10**	4.42±0.00	52.16±0.00
NXC-I,s.h.	43.73±0.07	40.19±0.06	37.75±0.05	4.45±0.04	11.03±0.02	24.77±0.01

Table 9.2: Comparison of NAPKINXC (NXC) with different modes of node classifiers training. We perform batch LIBLINEAR (B) or incremental ADAGRAD (I) training with logistic loss (log) or squared hinge loss (s.h.).

9.2.1 Batch and incremental learning

For batch learning, we use LIBLINEAR, the dual coordinate descent method, with either logistic loss or squared hinge loss. We use L_2 regularization for both and tune its C parameter for each data set. For incremental learning, we use ADAGRAD [Duchi et al., 2011] with 3 epochs and tune the base learning rate ϵ for each data set. As above, we use either logistic loss or squared hinge loss. In all algorithms, we prune the weights at 0.1 to obtain smaller models and use uniform-cost search to obtain top-k predictions. Let us point out that the configuration based on LIBLINEAR with squared hinge loss is similar to

PARABEL. The difference is that PARABEL uses beam search, thus we run the implementation from NAPKINXC here.

The results are given in Table 9.2. None of the configurations strictly dominates the others. It seems, however, that ADAGRAD with squared hinge loss usually performs the worst. This agrees with the fact that stochastic gradient approaches perform usually better with logistic loss. This configuration also leads to models with substantially longer testing times. In turn, significantly larger models for some data sets are built by ADAGRAD with logistic loss, while the training time can be doubled by LIBLINEAR with the same loss. It seems from this analysis that the batch training with squared hinge loss is the most reliable, without outlying results. Moreover, it performs the best on both WIKIPEDIA data sets. Nevertheless, incremental learning is a valid competitor that can be easily combined with training of dense representation or online tree structure building.

9.2.2 Prediction methods

We compare two prediction algorithms, uniform-cost search and beam search, and for each of them, we consider two implementations: online and batch. Online implementations are done in NAPKINXC. It is well-suited for online predictions and exploits efficient ROBIN HOOD hash maps to perform fast predictions. The computational cost of online methods is independent of the number of processed instances. PARABEL uses beam search implemented for batches of test instances. It benefits from using larger batches of test instances. In each node it decompresses its sparse model to a dense form before evaluation for test instances (see discussion on both algorithms in Section 8.5). For this experiment, we implement the batch uniform-cost in PARABEL, besides the original beam search based prediction method. Besides the prediction methods, NAPKINXC is set in the experiment to have the same setting as PARABEL. We use a binary tree with pre-leaf node arity equal to 100, built using hierarchical k-means clustering, and use squared hinge loss with weight pruning at threshold 0.1. This way the PLT models in NAPKINXC and PARABEL are as similar as possible. The small deviations in the results between both models are likely caused by small differences in their implementations and randomness. In each experiment, we use 5 different NAPKINXC or PARABEL models and average the obtained results.

First we focus on predictive performance and computational cost of both methods implemented online in NAPKINXC. In this experiment, we run top-$1, 3, 5$ prediction with uniform-cost search and beam search prediction with beam $B = 1, 5, 10$. We measure precision@$1, 3, 5$ and the average number of calls to node classifiers per observation. Table 9.3 gives the measured precision. We aggregate the results of all uniform-cost runs into a single column so that a precision@k value corresponds to uniform-cost search retrieving at least k top-scored leaves. Notice that for uniform-cost search, first, increasing k does not affect the predictions up to the former k, and second, for top-k prediction only precision up to k-th place makes sense. On the other hand, for beam search, running with a larger beam may affect all the predictions. Uniform-cost search in most cases obtains the highest precision among the tested methods, sometimes on par with beam search with the beam of size 10, sometimes is slightly worse, within the standard error. Beam search with beam equal 5 frequently does not meet the performance of beam search with the beam of size 10 or uniform cost search.

Prediction method	u-c search	beam search		
		B=1	B=5	B=10
	$p@1$ [%]			
EurLex-4K	**80.20±0.28**	73.72±0.27	80.20±0.27	**80.20±0.27**
AmazonCat-13K	92.39±0.05	90.29±0.20	**92.44±0.06**	**92.44±0.06**
Wiki10-30K	**84.16±0.11**	66.79±1.17	84.07±0.10	**84.16±0.11**
DeliciousLarge-200K	**46.29±0.06**	32.63±0.53	43.58±2.19	46.25±0.05
WikiLSHTC-325K	**62.79±0.03**	45.77±7.41	40.87±13.45	**62.79±0.03**
WikipediaLarge-500K	66.89±0.11	54.98±0.30	**66.94±0.12**	**66.94±0.12**
Amazon-670K	**43.32±0.02**	36.17±0.03	41.91±0.90	43.22±0.03
Amazon-3M	**46.26±0.01**	37.23±0.03	45.82±0.02	46.19±0.02
	$p@3$ [%]			
EurLex-4K	**65.32±0.53**	47.16±0.31	65.27±0.52	65.32±0.53
AmazonCat-13K	78.49±0.02	52.74±0.83	78.49±0.02	**78.50±0.02**
Wiki10-30K	**72.43±0.11**	38.78±0.41	71.79±0.17	72.41±0.11
DeliciousLarge-200K	**39.71±0.07**	25.02±0.61	35.53±3.26	39.63±0.08
WikiLSHTC-325K	**41.18±0.02**	22.13±3.44	26.58±8.76	41.16±0.02
WikipediaLarge-500K	47.69±0.06	29.22±0.16	47.33±0.07	**47.70±0.06**
Amazon-670K	**38.20±0.02**	25.68±0.03	36.52±0.82	38.03±0.02
Amazon-3M	**43.47±0.01**	30.87±0.02	42.75±0.01	43.35±0.01
	$p@5$ [%]			
EurLex-4K	**52.98±0.88**	33.39±0.26	52.84±0.86	**52.98±0.88**
AmazonCat-13K	**63.88±0.02**	35.22±0.78	63.82±0.02	**63.88±0.02**
Wiki10-30K	**63.11±0.04**	26.79±0.16	61.54±0.16	63.07±0.04
DeliciousLarge-200K	**36.45±0.09**	21.43±0.58	31.71±3.56	36.29±0.10
WikiLSHTC-325K	**30.25±0.02**	14.35±2.19	19.28±6.34	30.20±0.02
WikipediaLarge-500K	**36.98±0.04**	19.53±0.09	36.26±0.06	36.96±0.04
Amazon-670K	**34.32±0.03**	20.17±0.01	32.39±0.76	34.07±0.04
Amazon-3M	**41.41±0.01**	27.16±0.03	40.44±0.01	41.24±0.00

Table 9.3: Comparison of PLTs implementations using different online prediction methods: exact uniform-cost search and beam search with beam 1,5,10, all in NAPKINXC.

Let us now move on to the computational cost of these methods. Figure 9.1 gives the average number of calls to node classifiers per observation and the average prediction time per observation, using different prediction methods. First, let us focus on the average number of calls to node classifiers. The higher is k, or the larger is the beam B, the more node classifiers are called. Beam search with $B = 10$, suggested by Prabhu et al. [2018], calls in most cases more classifiers than the uniform-cost search. This shows that beam search usually needs to search more nodes, and call their classifiers, to achieve the predictive performance of the uniform-cost search. Uniform-cost search calls more node classifiers for DeliciousLarge-200K data set. This is caused by the large average number of positive labels in this data set. The average prediction time per instance mostly follows the number of calls to node classifiers. The slower growth of the average prediction time with increasing k or B compared to the number of visited nodes is likely caused by the increasing sparsity of models down the tree.

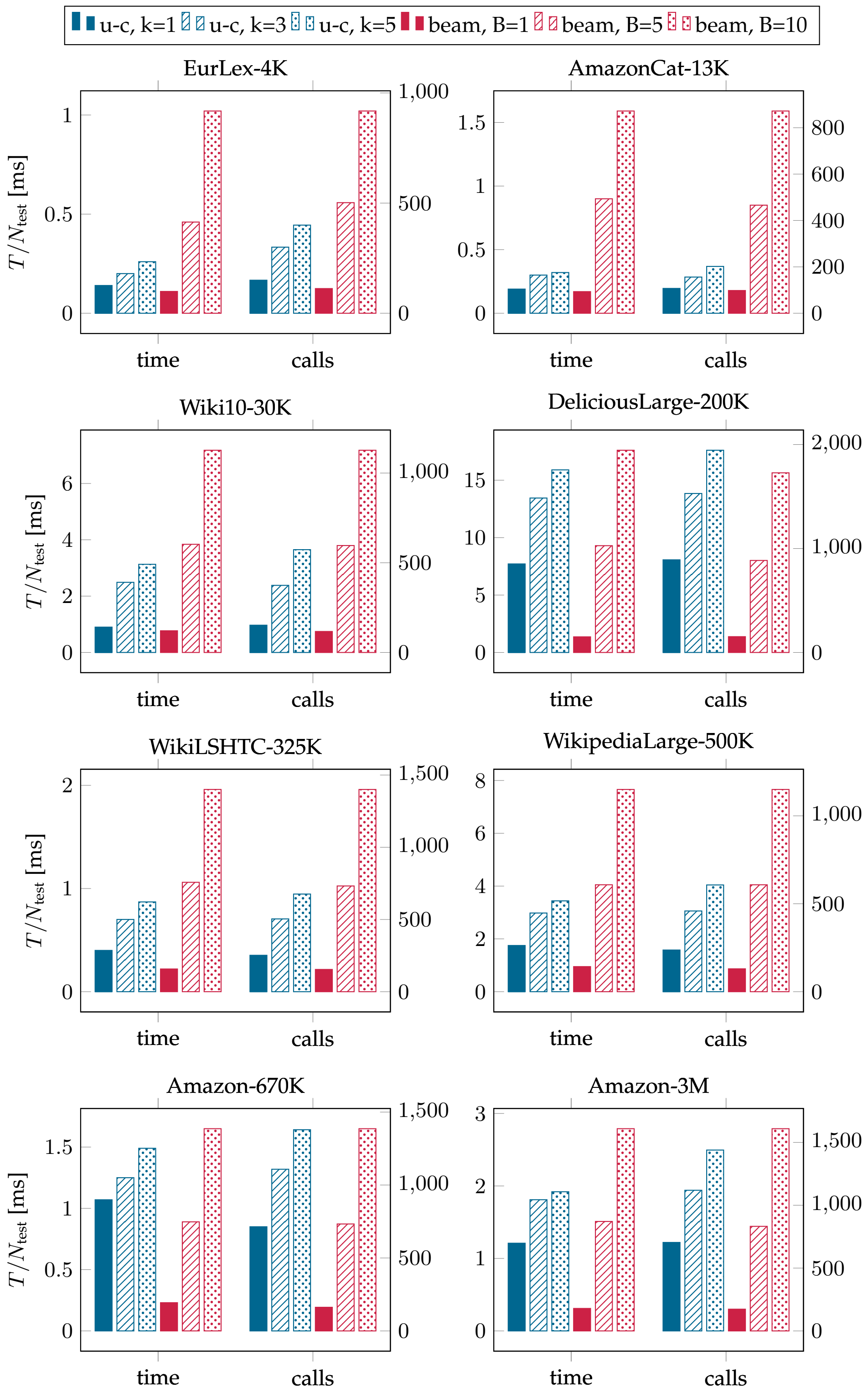


Figure 9.1: Average average prediction time T/N_{test} in [ms] and number of calls to node classifiers (calls) for online uniform-cost search (u-c) with $k = 1, 3, 5$ and online beam search (beam) with $B = 1, 5, 10$. The left y-axis corresponds to the prediction time, the right, to the number of calls to node classifies.

In the next experiment, we consider the batch implementations. We run the batch uniform-cost search with $k = 5$, and compare its performance to batch beam search with beam $B = 10$. We focus on computational costs. Figure 9.2 (lines representing batch uniform-cost search and batch beam search) shows the average prediction time per a single test instance, T/N_{test}, as a function of the batch size N_{test}. For each batch size, we create 50 samples of instances by selecting them uniformly from the test set. We measure the average prediction time over 5 different PARABEL models, which gives 250 measurements in total per each batch size. For smaller N_{test} the average prediction time per instance is shorter for uniform cost search, while for larger batches the beam search is faster. Moreover, batch uniform-cost search performs better in terms of the average prediction time compared to batch beam search on data sets with a small number of labels than on data sets with more labels.

Finally, we compare different implementations of PLTs and their default prediction methods: uniform-cost search for NAPKINXC and batch beam search with $B = 10$ for PARABEL. We consider the top-5 prediction. As shown in Table 9.4, both methods perform very similarly in terms of precision@k, even having slightly different models. To compare the computational performance of those two default methods we perform an analogous experiment to the previous one with uniform-cost search in NAPKINXC. The results are also given in Figure 9.2 (lines representing online u-c search and batch beam search). The prediction time of the uniform-cost search, processing instance by instance, is independent of the batch size and it is lower than 10ms for most of the data sets. Beam search, working on batches of instances, is more than 100 times slower than the uniform-cost search for small batches. To reach the prediction time of uniform-cost search it requires often batch sizes greater than 1000.

	p@1 [%]		p@3 [%]		p@5 [%]	
Implementation	NAPKINXC	PARABEL	NAPKINXC	PARABEL	NAPKINXC	PARABEL
Prediction method	u-c search	beam search	u-c search	beam search	u-c search	beam search
EurLex-4K	80.17±0.27	**80.66±0.16**	65.33±0.53	**67.76±0.05**	53.01±0.87	**56.57±0.07**
AmazonCat-13K	92.40±0.04	**92.58±0.02**	78.49±0.02	**78.53±0.00**	63.88±0.02	**63.90±0.01**
Wiki10-30K	**84.17±0.10**	**84.17±0.03**	**72.43±0.10**	72.12±0.04	63.12±0.04	**63.30±0.05**
DeliciousLarge-200K	46.30±0.07	**46.44±0.07**	**39.76±0.08**	39.66±0.06	**36.54±0.07**	36.19±0.05
WikiLSHTC-325K	**62.78±0.03**	**62.78±0.02**	41.17±0.02	**41.22±0.02**	30.25±0.02	**30.27±0.01**
WikipediaLarge-500K	66.77±0.08	**67.05±0.14**	47.63±0.04	**47.75±0.10**	36.94±0.02	**36.99±0.08**
Amazon-670K	**43.31±0.03**	43.13±0.02	**38.19±0.03**	37.94±0.03	**34.31±0.03**	34.00±0.00
Amazon-3M	**46.23±0.01**	46.14±0.01	**43.48±0.01**	43.32±0.01	**41.41±0.01**	41.20±0.01

Table 9.4: Precision@k of uniform-cost (u-c) search in NAPKINXC and beam search in PARABEL.

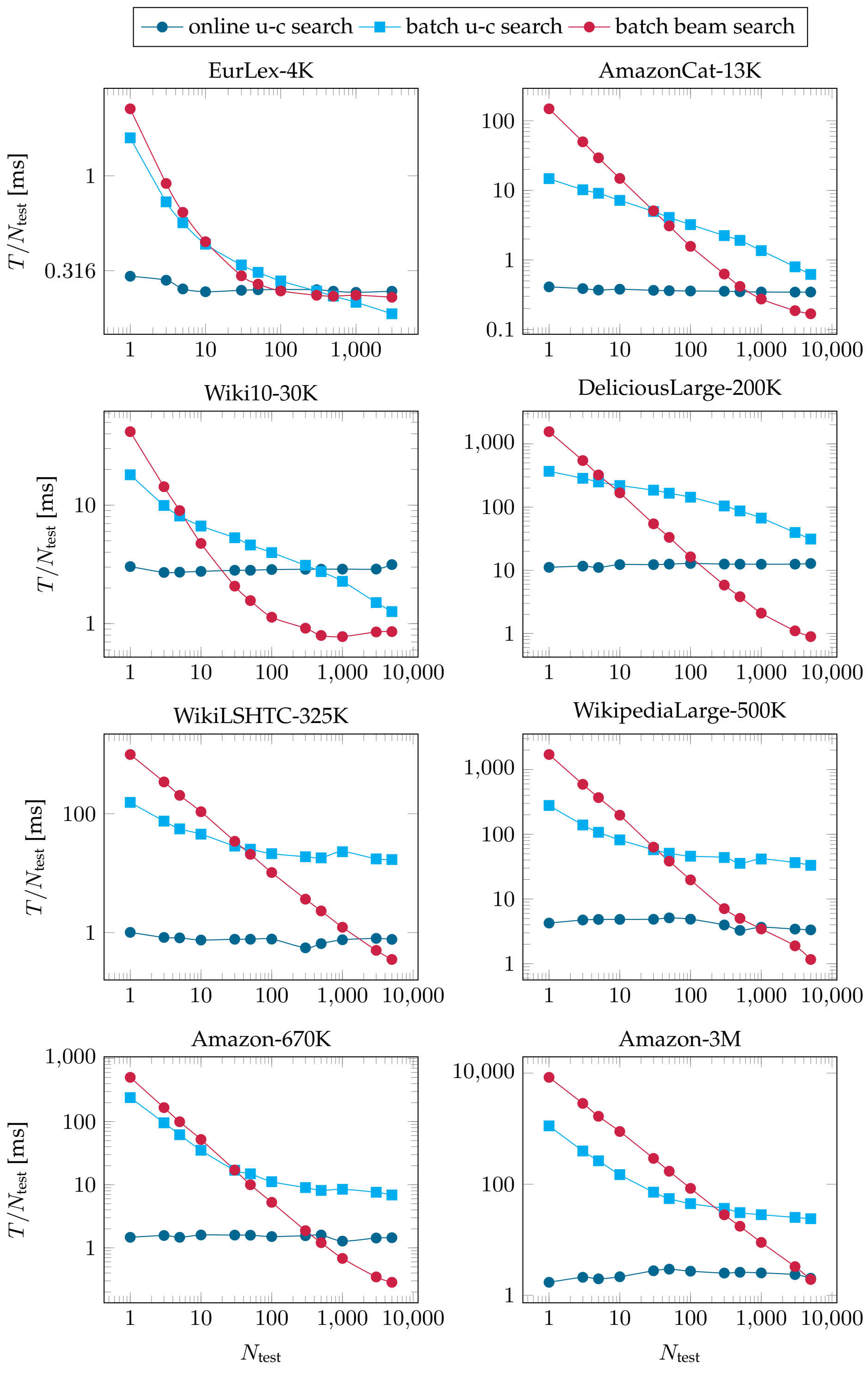


Figure 9.2: Average prediction times of online uniform-cost search in NAPKINXC, batch uniform-cost search and batch beam search in PARABEL as a function of batch size N_{test}.

9.2.3 Sparse and dense representation

In the next experiment, we test the performance of PLTs with sparse and dense representation. To this end, we use NAPKINXC and EXTREMETEXT, respectively. Besides representation, we use a similar setting for both algorithms. Trees are built with hierarchical k-means clustering. Node classifiers are trained incrementally by minimizing logistic loss, however, NAPKINXC uses ADAGRAD, while EXTREMETEXT stochastic gradient descent with L_2 regularization. Prediction in both is based on uniform-cost search.

Representation	dense	sparse	dense	sparse	dense	sparse
	$p@1$ [%]		$p@3$ [%]		$p@5$ [%]	
EurLex-4K	77.29±0.21	**80.43±0.09**	64.41±0.11	**66.08±0.26**	53.56±0.11	**53.87±0.58**
AmazonCat-13K	91.96±0.02	**93.23±0.02**	77.41±0.01	**78.76±0.03**	62.75±0.02	**64.05±0.02**
Wiki10-30K	**85.76±0.06**	84.92±0.10	74.37±0.10	**74.52±0.09**	64.44±0.05	**65.29±0.04**
DeliciousLarge-200K	**47.95±0.03**	45.27±0.06	**41.69±0.02**	38.26±0.03	**38.60±0.01**	34.88±0.02
WikiLSHTC-325K	57.58±0.06	**60.99±0.04**	38.01±0.04	**39.85±0.02**	28.33±0.02	**29.50±0.01**
WikipediaLarge-500K	64.56±0.06	**65.68±0.15**	46.04±0.06	**46.62±0.09**	36.06±0.04	**36.52±0.06**
Amazon-670K	40.24±0.03	**43.82±0.01**	35.84±0.04	**38.88±0.03**	32.61±0.05	**35.31±0.03**
Amazon-3M	39.29±0.03	**43.61±0.12**	36.53±0.02	**40.44±0.09**	34.65±0.01	**38.32±0.06**
	T_{train} [h]		T/N_{test} [ms]		M_{size} [GB]	
EurLex-4K	0.21±0.00	**0.01±0.00**	0.36±0.00	**0.25±0.02**	**0.02±0.00**	0.05±0.00
AmazonCat-13K	10.75±0.71	**0.17±0.00**	**0.17±0.01**	0.32±0.02	**0.41±0.00**	0.72±0.00
Wiki10-30K	0.72±0.06	**0.11±0.00**	**0.91±0.06**	5.24±0.42	**0.25±0.00**	0.91±0.00
DeliciousLarge-200K	38.32±1.84	**2.97±0.03**	**1.69±0.05**	11.51±0.57	**1.90±0.00**	15.05±0.00
WikiLSHTC-325K	2.34±0.17	**1.52±0.00**	**0.57±0.01**	2.44±0.02	**3.30±0.00**	4.93±0.00
WikipediaLarge-500K	28.23±1.41	**8.11±0.18**	**0.64±0.01**	7.86±0.19	**5.50±0.00**	19.11±0.00
Amazon-670K	6.27±0.49	**0.42±0.00**	**1.18±0.01**	5.93±0.11	**1.60±0.00**	6.22±0.00
Amazon-3M	36.07±2.05	**4.44±0.10**	**0.91±0.01**	4.42±0.00	**6.20±0.00**	52.16±0.00

Table 9.5: Comparison of dense (EXTREMETEXT) and sparse (NAPKINXC) representation.

Table 9.5 shows the results. NAPKINXC with sparse representation achieves higher precision@k than EXTREMETEXT on almost all data sets. This agrees with a common observation that learning a powerful dense representation for XMLC problems is difficult. Nevertheless, the results of EXTREMETEXT are approaching those of NAPKINXC, despite its very simple architecture and gradient updates. Because of the additional layer, EXTREMETEXT needs more time for training, but the dense representation allows faster predictions and smaller models. Let us comment, however, on the last observation. The size of a model in EXTREMETEXT is determined by the dimension of dense representation, the number of labels, features, and tree nodes. In NAPKINXC with sparse representation, the model size is influenced by the distribution of features among the labels, the tree structure, the chosen loss function and regularization. Moreover, aggressive weight pruning can be applied without a significant drop in predictive performance, as we show in Appendix B.5.

9.2.4 Tree structure

The choice of the tree structure is crucial as it affects all aspects of the performance: the accuracy of predictions, execution times, and model sizes. In this experiment, we investigate different tree-building strategies described in Section 8.6. We first compare two types of balanced binary trees in which labels are split either randomly or using k-means clustering in a top-down procedure. In trees of both types, pre-leaf nodes are set to have a high degree equal to 100, that is, the splitting procedure stops when a cluster contains less than 100 labels which are then transformed into leaves. Both algorithms create trees of the same depth, but with a different arrangement of labels. The results are given in Table 9.6. In all cases, the k-means tree outperforms the random one in terms of precision@k. On some data sets this difference is not substantial, but in several cases, k-means clustering leads to a huge boost of almost 20 percent. Also training time and model size benefit from the clustering. The power of k-means trees can be explained by the fact that co-occurring and similar labels are grouped. Thanks to this a training observation is used in fewer nodes on average, the training tasks in tree nodes become simpler, and fewer features are necessary to solve them.

Tree type	random	k-means	random	k-means	random	k-means
	p@1 [%]		p@3 [%]		p@5 [%]	
EurLex-4K	76.22±0.21	**80.51±0.16**	54.54±0.13	**65.65±0.40**	39.20±0.09	**53.33±0.68**
AmazonCat-13K	91.39±0.04	**93.04±0.02**	75.86±0.03	**78.44±0.02**	61.00±0.02	**63.70±0.02**
Wiki10-30K	84.26±0.07	**85.36±0.09**	71.68±0.06	**73.90±0.07**	60.16±0.12	**63.84±0.07**
DeliciousLarge-200K	49.13±0.03	**49.55±0.05**	42.68±0.01	**43.08±0.03**	39.47±0.02	**39.90±0.02**
WikiLSHTC-325K	44.04±0.02	**61.96±0.03**	24.69±0.45	**40.77±0.02**	18.20±0.18	**30.19±0.02**
WikipediaLarge-500K	48.40±0.02	**66.20±0.05**	32.05±0.01	**47.14±0.02**	24.82±0.01	**36.83±0.01**
Amazon-670K	33.76±0.06	**43.54±0.01**	28.25±0.02	**38.71±0.02**	24.88±0.01	**35.15±0.03**
Amazon-3M	37.77±0.05	**46.09±0.02**	34.55±0.01	**43.11±0.01**	32.49±0.01	**40.98±0.01**
	T_{train} [h]		T/N_{test} [ms]		M_{size} [GB]	
EurLex-4K	**0.02±0.00**	0.02±0.00	**0.27±0.02**	0.39±0.03	**0.02±0.00**	0.02±0.00
AmazonCat-13K	0.84±0.03	**0.72±0.02**	0.37±0.02	**0.32±0.03**	0.40±0.00	**0.35±0.00**
Wiki10-30K	**0.19±0.01**	0.21±0.00	**2.59±0.21**	5.35±0.32	0.62±0.00	**0.58±0.00**
DeliciousLarge-200K	5.26±0.26	**2.58±0.15**	**5.24±0.43**	9.89±0.89	1.30±0.00	**0.95±0.00**
WikiLSHTC-325K	3.01±0.16	**2.95±0.15**	**1.36±0.13**	1.77±0.11	3.25±0.00	**2.73±0.00**
WikipediaLarge-500K	22.07±0.59	**16.10±0.44**	10.65±0.52	**6.67±0.23**	12.00±0.00	**8.89±0.00**
Amazon-670K	1.02±0.02	**0.56±0.00**	4.74±0.22	**4.13±0.28**	3.00±0.00	**2.26±0.00**
Amazon-3M	48.09±1.72	**7.07±0.56**	8.05±0.17	**3.26±0.08**	29.00±0.00	**20.84±0.00**

Table 9.6: Precision@k for $k = 1, 3, 5$, training time, average test time per example, and model size for random and k-means trees.

In the next two experiments, we evaluate the impact of tree depth on the predictive and computational performance of PLTs. In the first experiment, we increase the degree of tree nodes to 16 and 64, but keep the degree of pre-leaves equal to 100. Such an approach is similar to the one used in BONSAI TREE [Khandagale et al., 2019]. The results for k-means trees and logistic loss are given in Table 9.7a. In the second experiment, we use binary trees, but change the degree of pre-leaves from 100 to 25 and 400. The results are given in Table 9.7b.[3] In both experiments, precision@k slightly increases with a decrease in tree depth. This behavior is expected as suggested by the theoretical results from Chapter 5. The shorter paths should result in tighter upper bounds. On the other hand, a shallower tree leads to longer training times, as an observation is used for training in more nodes. Notice that the 1-VS-ALL approach can be treated as an extremely shallow tree, with each training observation used in all nodes. Similarly to the training time, we should expect prediction time to increase with

[3]For completeness, we present the results for the squared hinge loss in Appendix B.4.

Arity	2	16	64	2	16	64
		$p@1$ [%]			T/N_{test} [ms]	
EurLex-4K	80.51±0.16	81.06±0.03	**81.20±0.13**	0.39±0.03	**0.23±0.02**	0.28±0.03
AmazonCat-13K	93.04±0.02	93.15±0.04	**93.19±0.01**	**0.32±0.03**	0.34±0.03	0.43±0.04
Wiki10-30K	85.36±0.09	85.85±0.06	**86.03±0.07**	**5.35±0.32**	5.63±0.33	7.34±0.62
DeliciousLarge-200K	49.55±0.05	**49.56±0.04**	49.51±0.04	**9.89±0.89**	12.74±1.69	11.77±0.61
WikiLSHTC-325K	61.96±0.03	63.16±0.03	**63.62±0.03**	1.77±0.11	**1.26±0.03**	1.91±0.19
WikipediaLarge-500K	66.20±0.05	67.36±0.10	**67.49±0.05**	6.67±0.23	**6.47±0.30**	9.02±0.39
Amazon-670K	**43.54±0.01**	43.33±0.04	43.53±0.05	4.13±0.28	**2.91±0.17**	5.12±0.24
Amazon-3M	46.09±0.02	46.74±0.01	**46.97±0.01**	3.26±0.08	**3.06±0.04**	4.19±0.16
		T_{train} [h]			M_{size} [GB]	
EurLex-4K	**0.02±0.00**	0.02±0.00	0.02±0.00	**0.02±0.00**	0.02±0.00	0.02±0.00
AmazonCat-13K	**0.72±0.02**	0.82±0.05	1.09±0.03	0.35±0.00	**0.34±0.00**	0.34±0.00
Wiki10-30K	0.21±0.00	**0.20±0.01**	0.43±0.04	0.58±0.00	**0.50±0.00**	0.51±0.00
DeliciousLarge-200K	**2.58±0.15**	5.49±0.36	9.21±0.90	0.95±0.00	0.93±0.00	**0.91±0.00**
WikiLSHTC-325K	**2.95±0.15**	3.71±0.13	7.13±0.41	2.73±0.00	2.65±0.00	**2.62±0.00**
WikipediaLarge-500K	**16.10±0.44**	26.28±0.40	46.84±2.94	8.89±0.00	8.20±0.00	**8.13±0.00**
Amazon-670K	**0.56±0.00**	0.78±0.03	2.17±0.12	2.26±0.00	1.68±0.00	**1.60±0.00**
Amazon-3M	**7.07±0.56**	9.80±0.10	23.69±1.09	20.84±0.00	20.32±0.00	**20.17±0.00**

(a) Results for arity equal to 2, 16 or 64 and pre-leaf node degree equal to 100.

Pre-leaf degree	25	100	400	25	100	400
		$p@1$ [%]			T/N_{test} [ms]	
EurLex-4K	**80.62±0.12**	80.51±0.16	79.49±0.39	**0.15±0.00**	0.39±0.03	0.62±0.02
AmazonCat-13K	93.05±0.02	93.04±0.02	**93.07±0.02**	**0.14±0.02**	0.32±0.03	0.67±0.04
Wiki10-30K	85.28±0.08	85.36±0.09	**85.41±0.06**	**2.07±0.30**	5.35±0.32	11.43±0.21
DeliciousLarge-200K	49.56±0.03	49.55±0.05	**49.62±0.03**	**6.25±0.35**	9.89±0.89	18.23±0.72
WikiLSHTC-325K	61.42±0.03	61.96±0.03	**62.16±0.05**	**0.66±0.04**	1.77±0.11	4.21±0.10
WikipediaLarge-500K	65.72±0.12	**66.20±0.05**	65.95±0.10	**3.52±0.05**	6.67±0.23	20.13±1.07
Amazon-670K	41.83±0.02	**43.54±0.01**	43.21±0.03	**1.61±0.04**	4.13±0.28	11.74±0.46
Amazon-3M	46.07±0.01	46.09±0.02	**46.13±0.01**	**1.47±0.00**	3.26±0.08	11.45±0.22
		T_{train} [h]			M_{size} [GB]	
EurLex-4K	**0.01±0.00**	0.02±0.00	0.04±0.00	0.03±0.00	**0.02±0.00**	**0.02±0.00**
AmazonCat-13K	**0.36±0.03**	0.72±0.02	1.38±0.03	0.38±0.00	0.35±0.00	**0.34±0.00**
Wiki10-30K	**0.09±0.01**	0.21±0.00	0.32±0.01	0.73±0.00	0.58±0.00	**0.37±0.00**
DeliciousLarge-200K	**2.14±0.11**	2.58±0.15	4.18±0.23	1.14±0.00	0.95±0.00	**0.86±0.00**
WikiLSHTC-325K	**2.36±0.05**	2.95±0.15	4.83±0.29	3.28±0.00	2.73±0.00	**2.47±0.00**
WikipediaLarge-500K	**12.40±0.08**	16.10±0.44	34.15±0.46	10.60±0.01	8.89±0.00	**7.94±0.00**
Amazon-670K	**0.42±0.01**	0.56±0.00	1.01±0.05	2.20±0.00	2.26±0.00	**1.52±0.00**
Amazon-3M	**5.18±0.25**	7.07±0.56	16.41±0.52	24.00±0.00	20.84±0.00	**19.35±0.01**

(b) Results arity equal to 2 and pre-leaf node degree equal to 25, 100, or 400.

Table 9.7: Precision@k1, average prediction time per example, training time and model size for k-means trees if different depths with logistic loss.

the decreasing tree depth. This is, however, clearly visible only in the second experiment, in which we change the degree of pre-leaf nodes. Interestingly, the size of the resulting models does not significantly change over the different tree structures. The larger number of nodes is likely compensated by sparser models.

9.2.5 Ensemble of PLTs

In the last experiment focused on design choices, we analyze the predictive performance of small ensembles of PLTs. In Figure 9.3, we compare ensembles of size 3 and 5 to a single tree. Each tree is trained using 2-means clustering with the degree of pre-leaf nodes set to 100. The tree nodes are trained using LIBLINEAR with either logistic loss or squared hinge loss. The results show that the gain of using 3 trees instead of one is much greater than the gain of using 5 trees instead of 3. This makes the ensemble of size 3 a reasonable trade-off between predictive

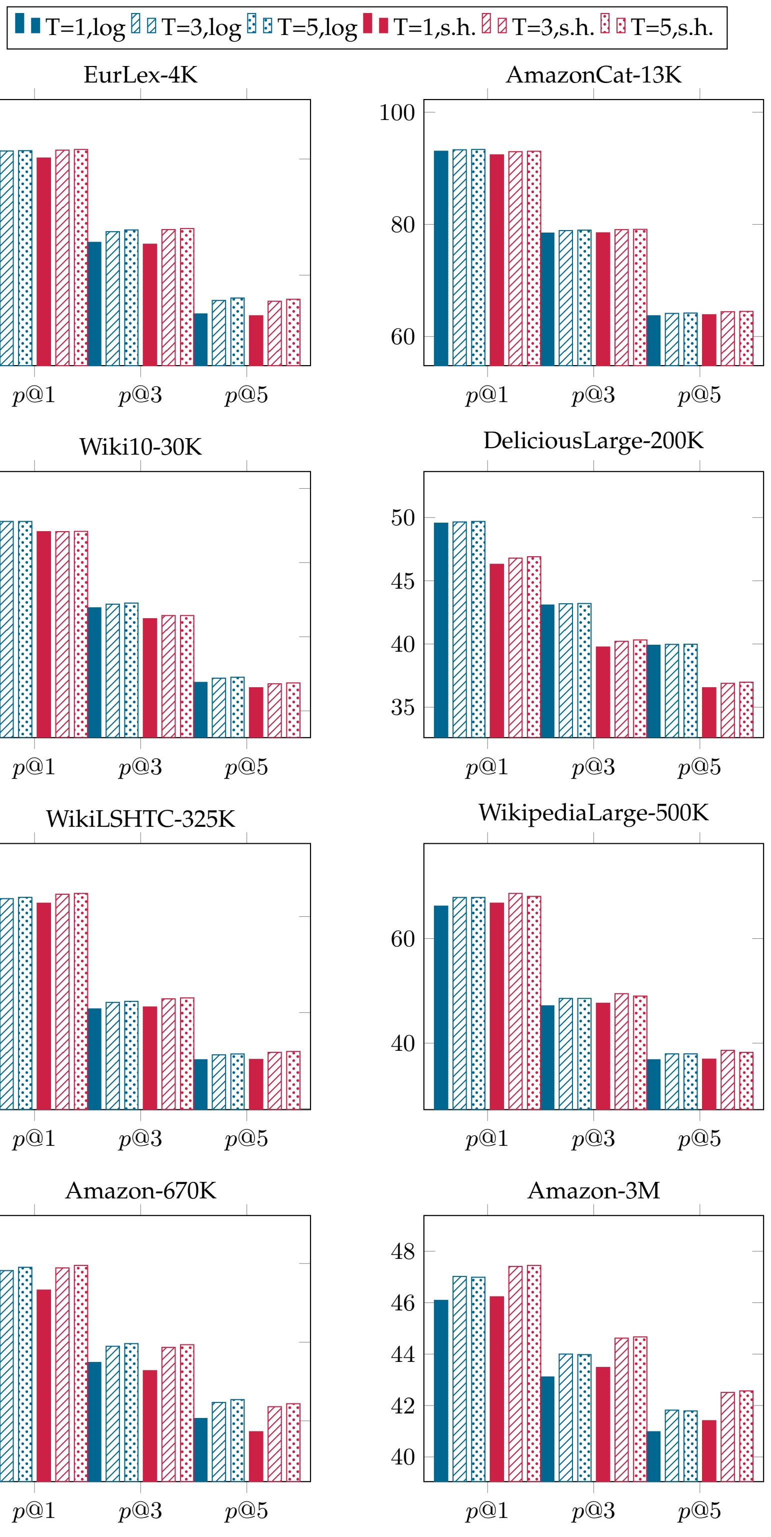


Figure 9.3: Precision@k for ensembles of $T = 1$, 3, and 5 trees trained using either logistic loss (log) or squared hinge loss (s.h.).

and computational cost, as the size of the models and computational cost grow linearly with the number of trees. It seems also that ensembles with squared hinge loss gain slightly more than the ensembles trained with logistic loss.

9.3 Generalized classification performance metrics

The previous experiments concern PLTs with top-k predictions suited for precision@k. In this section, we focus on threshold-based predictions and generalized performance metrics, discussed in Section 5.3. We constrain our analysis to Hamming loss, micro and macro F_1-measure. For each metric, we report the results of three approaches. The first one uses a fixed threshold of 0.5, which is theoretically optimal for Hamming loss. Remark, however, that for estimated probabilities the optimal threshold may be different. The second approach optimizes the micro F_1-measure or macro F_1 measure. In the case of micro-F_1-measure, it uses a global tuned threshold, as suggested by theory. For macro-F_1-measure, the theory suggests the use of a separate tuned threshold for each label. However, the tested approach uses a separate threshold for 1000 or 10000 most frequent labels, depending on the data set size, and a single threshold for the rest of them. This was necessary to reduce the run time of threshold tuning and resolve the problems with thresholds optimization for very rare labels. The third approach is uniform-cost search retrieving k labels, with k set separately for each data set to the average number of positive labels in the data set, see avg. $|\mathcal{L}_{\boldsymbol{x}}|$ in Table 9.1. However this approach differs from the one suggested by the theoretical results, we include it as it provides additional insights. In this experiment we use 70$\%$ of original training data to train a PLT model, and the remaining 30$\%$ to tune the thresholds using online F-measure optimization (OFO) [Busa-Fekete et al., 2015], similarly to [Jasinska et al., 2016]. We repeat computations 5 times and report the average results along with standard errors.

Table 9.8 presents the results. Notice that for Hamming loss the lower the value the better is the performance, while for F-measures it is the opposite, the higher the value, the better. Let us first focus on threshold-based approaches. As expected from the theoretical analysis, a procedure using thresholds suited for a given metric leads to significantly better results: threshold 0.5 performs best for Hamming loss, while the OFO-tuned thresholds usually outperform it in terms of F_1-measures. Interestingly, for some data sets the top-avg. $|\mathcal{L}_{\boldsymbol{x}}|$ prediction outperforms threshold tuning in terms of both F_1-measures. However, on the other ones, it severely underperforms compared to the theoretically grounded method. We hypothesize that the side information of the average number of labels that should be selected per instance may facilitate the choice of the correct ones. This, accompanied by the fact that it is hard to precisely tune the thresholds for infrequent labels, may lead to the observed advantage of top-avg. $|\mathcal{L}_{\boldsymbol{x}}|$ prediction over OFO tuned thresholds on some data sets. However, the top-avg. $|\mathcal{L}_{\boldsymbol{x}}|$ prediction is not guaranteed to perform well, while the threshold-based prediction is theoretically sound.

	Hamming loss[1]		
	thr=0.5	micro-OFO	top-avg. $\lvert\mathcal{L}_{\boldsymbol{x}}\rvert$
EurLex-4K	**4.20±0.00**	8.10±0.13	5.15±0.00
AmazonCat-13K	**2.77±0.00**	3.14±0.02	6.91±0.05
Wiki10-30K	**17.42±0.01**	24.92±0.84	25.21±0.03
DeliciousLarge-200K	**97.16±0.00**	584.20±0.38	157.75±0.22
WikiLSHTC-325K	**2.92±0.00**	4.87±0.43	04.03±0.00
WikipediaLarge-500K	**4.10±0.01**	4.72±0.30	6.18±0.01
Amazon-670K	**4.66±0.00**	6.90±0.23	6.65±0.00
Amazon-3M	**34.80±0.00**	69.55±0.59	53.18±0.00
	micro-F_1 [%]		
	thr=0.5	**micro-OFO**	top-avg. $\lvert\mathcal{L}_{\boldsymbol{x}}\rvert$
EurLex-4K	47.04±0.00	39.57±0.34	**50.00±0.00**
AmazonCat-13K	67.25±0.00	**67.95±0.17**	31.60±0.45
Wiki10-30K	25.01±0.06	31.63±0.88	**33.69±0.07**
DeliciousLarge-200K	0.83±0.00	**12.45±0.78**	9.00±0.13
WikiLSHTC-325K	32.67±0.14	30.74±0.20	**37.80±0.05**
WikipediaLarge-500K	32.49±0.22	**38.72±0.11**	37.30±0.11
Amazon-670K	19.73±0.13	30.39±0.72	**34.58±0.05**
Amazon-3M	13.23±0.03	25.00±0.15	**26.62±0.00**
	macro-F_1[%]		
	thr=0.5	**macro-OFO**	top-avg. $\lvert\mathcal{L}_{\boldsymbol{x}}\rvert$
EurLex-4K	44.65±0.00	46.35±0.95	**50.15±0.00**
AmazonCat-13K	27.53±0.00	**42.05±0.50**	16.49±0.23
Wiki10-30K	29.89±0.01	29.29±0.08	**31.51±0.05**
DeliciousLarge-200K	**11.18±0.00**	8.56±0.32	11.11±0.19
WikiLSHTC-325K	18.43±0.04	23.08±0.18	**27.22±0.03**
WikipediaLarge-500K	7.28±0.04	12.85±0.16	**18.44±0.07**
Amazon-670K	50.60±0.02	51.26±0.07	**52.29±0.09**
Amazon-3M	3.42±0.02	**18.33±0.00**	15.34±0.01

[1] multiplied by the number of labels to avoid presentation of very small numbers

Table 9.8: The results of PLTs with threshold-based predictions along with top-avg. $|\mathcal{L}_{\boldsymbol{x}}|$ predictions for Hamming loss, micro F_1-measures. The name of the column with the theoretically optimal tuning strategy for a given metric is given in bold.

9.4 Comparison to hierarchical softmax

In this experiment, we verify our theoretical findings from Sections 3.2 and 5.6. As we have shown, hierarchical softmax with the pick-one-label heuristic (HSM-POL) leads to a suboptimal solution with respect to precision@k. To demonstrate this empirically, we run two experiments. In the first one, we compare the performance of PLTs and HSM-POL on synthetic data. In the second experiment, we evaluate both algorithms on benchmark data sets. To conduct the experiments, we implemented HSM-POL in NAPKINXC. We made it as similar as possible to the implementation of PLTs, with the only differences coming from the model definition. For both algorithms, we use LIBLINEAR with L2-regularized logistic loss to train node classifiers. In the experiment on benchmark data, we additionally use weight pruning to reduce model sizes. This is not necessary for synthetic data. To simulate the pick-one-label heuristic in batch learning, we transform a multi-label observation with $||\boldsymbol{y}||_1$ positive labels to $||\boldsymbol{y}||_1$ weighted multi-class observations, each with a different positive label assigned and weight equal $\frac{1}{||\boldsymbol{y}||_1}$.

	HSM-POL	PLT	# losses	# ties	# wins	p-value
multi-label dependent	71.29±0.98	72.31±0.94	5	0	45	4.21e-09
multi-label independent	32.66±0.08	32.64±0.08	25	3	22	0.4799
multi-class	61.23±1.14	61.23±1.14	0	50	0	-

Table 9.9: Precision@1 of PLTs and HSM-POL on synthetic data. Reported are mean values over 50 runs along with standard errors, the number of wins, ties, and losses of PLTs, and p-value of the sign test.

The results for precision@1 are given in Table 9.9. We use three types of synthetic data generated from different distributions: multi-label with conditionally independent labels, multi-label with conditionally dependent labels, and multi-class. A detailed description of the data generation process is given in Appendix B.2. The presented values are averages over 50 runs along with standard errors. Notice, however, that the data generation processes may lead to very diverse problems, with a different level of noise. Therefore, standard errors indicate rather the diversity of the generated problems. To overcome this issue, we report the number of wins, ties, and losses, as well as p-values of the very conservative sign test. On data with conditionally dependent labels, PLTs clearly outperform HSM-POL as indicated by the p-value. This agrees with our theoretical results. On data with conditionally independent labels, both algorithms perform similarly without statistically significant differences. This also agrees with the theory, as we have proven that under label independence HSM-POL performs optimally for precision@k. The results on multi-class data completely match, as for this distribution, the PLT model boils down to HSM.

	p@1 [%]		r@1 [%]		r@5 [%]	
	HSM	PLT	HSM	PLT	HSM	PLT
EurLex-4K	67.89±0.26	**80.51±0.16**	13.64±0.06	**16.20±0.04**	44.64±0.45	**51.71±0.67**
AmazonCat-13K	88.19±0.16	**93.04±0.02**	24.73±0.06	**26.37±0.01**	69.38±0.09	**74.64±0.02**
Wiki10-30K	54.69±1.00	**85.36±0.09**	3.18±0.07	**5.06±0.01**	12.58±0.18	**18.26±0.02**
WikiLSHTC-325K	58.35±0.04	**61.96±0.03**	26.41±0.02	**27.41±0.01**	49.81±0.01	**52.96±0.03**
WikipediaLarge-500K	60.48±0.09	**66.20±0.05**	20.16±0.03	**21.50±0.01**	43.17±0.03	**47.12±0.03**
Amazon-670K	40.38±0.04	**43.54±0.01**	8.53±0.01	**9.01±0.01**	29.52±0.04	**32.83±0.02**

Table 9.10: Precision@1 and recall@k of hierarchical softmax with pick-one-label heuristic (HSM) and PLT on benchmark datasets.

Table 9.10 gives the results on benchmark data sets. The difference in performance between PLTs and HSM-POL is clearly visible. It is even more substantial than in the previous experiment. Besides precision@1, the table contains also results for recall@1. The pick-one-label heuristic should lead to optimal results for this metric, as shown by Menon et al. [2019] and in Chapter 3. Nevertheless, PLTs obtain better results also for this metric, but the difference is much smaller. This suggests that indeed HSM-POL can be well-suited for recall@k, but the pick-one-label heuristic may lead to corrupted learning problems in tree nodes.

As discussed in [Menon et al., 2019], there exist other strategies for optimizing recall@k, which may perform better than PLTs.

9.5 Online PLTs

We empirically verify online probabilistic label trees in which both node classifiers and tree structure are built incrementally. We implemented the OPLT algorithm, introduced in Chapter 7, in NAPKINXC. The tree is constructed using the simple complete tree policy from Algorithm 9. To train node classifiers, we use ADAGRAD with logistic loss. The incremental learning in the online setting requires quick access to model weights, preferably storing all of them at once in memory. Unfortunately, maintaining an array for all possible weights in a dense format would require, for many data sets, thousands of GB of memory. As described in Section 8.3, either hash maps, such as ROBIN HOOD, or feature hashing should be applied to overcome this problem. In the experiment, we compare both approaches.

	#features	#hashed features			RAM [GB]	
		64GB	128GB	256GB	ROBIN HOOD	dense vector
EurLex-4K	5000	$*$	$*$	$*$	0.6	0.2
AmazonCat-13K	203882	$*$	$*$	$*$	9	60
Wiki10-30K	101938	$*$	$*$	$*$	18	70
DeliciousLarge-200K	782585	13000	26000	52000	240	3593
WikiLSHTC-325K	1617899	8000	16000	32000	30	11754
WikipediaLarge-500K	2381304	5000	10000	20000	240	26670
Amazon-670K	135909	4000	8000	16000	36	2035
Amazon-3M	337067	1000	2000	4000	280	21187

Table 9.11: Number of features, hashed features for OPLT with complete tree policy and memory required to train OPLT with complete tree policy with ROBIN HOOD hash maps and dense vectors. With symbol '$*$' we denote data sets for which feature hashing is not needed to fit the available memory.

Feature hashing allows us to directly control the amount of memory used, but it may result in many unresolved collisions if the allocated space is too small. We consider setups with 64GB, 128GB, and 256GB of RAM. The ROBIN HOOD hash map avoids collisions, but does not allow restraining memory consumption. The number of hashed features which can be allocated without collisions is given in Table 9.11. We report this number for each data set and memory setup. It takes into account memory needed for $2m-1$ nodes, each containing model weights and cumulative gradients required by ADAGRAD, and auxiliary classifiers which number can be limited to m for the chosen tree building policy. Additionally, we present in the same table the amount of memory required by OPLT with ROBIN HOOD and OPLT with dense vectors.

To simulate the online/streaming setting, the OPLT algorithms run three times over training observations, each time permuted randomly. We evaluate the performance on the original test sets in terms of precision@1. The results are given in Table 9.12. For reference, we also present the results of a batch PLT trained with the logistic loss on a complete binary tree. OPLT with ROBIN HOOD performs similarly to PLT. This agrees with the results from Section 9.2.1 showing that incremental learning under logistic loss is competitive to its batch counterpart. Interestingly, ROBIN HOOD allows us to train OPLT in 256GB of RAM for all data sets, with the only exception of Amazon-3M for which 280GB is required. The performance of OPLT with feature hashing drops significantly for large data sets, even when using the same amount of memory as OPLT with ROBIN HOOD. One may observe that the smaller is the hashing space compared to the original feature space, the larger is the drop.

Algorithm	OPLT				PLT
Representation	feature hashing			ROBIN HOOD	
RAM	64GB	128GB	256GB	unlimited	unlimited
EurLex-4K	$*$	$*$	$*$	76.69±0.21	**76.82±0.35**
AmazonCat-13K	$*$	$*$	$*$	**91.35**±0.06	**91.20**±0.05
Wiki10-30K	$*$	$*$	$*$	**84.41±0.19**	82.74±0.14
DeliciousLarge-200K	44.61±0.03	44.52±0.04	44.58±0.06	46.29±0.05	**47.81±0.03**
WikiLSHTC-325K	32.18±0.04	34.71±0.01	36.90±0.00	**44.42±0.03**	43.91±0.02
WikipediaLarge-500K	25.39±0.03	29.04±0.02	32.90±0.06	**49.35±0.03**	47.25±0.02
Amazon-670K	23.18±0.05	26.64±0.02	29.53±0.05	**37.02±0.01**	35.12±0.03
Amazon-3M	10.31±0.01	14.18±0.01	18.49±0.03	37.80±0.01	**38.05±0.02**

Table 9.12: Performance of OPLT with different memory management strategies: feature hashing of 64GB, 128GB and 256GB, and ROBIN HOOD hash maps. Results of a batch counterpart are given for reference. With symbol '$*$' we denote data sets where feature hashing is not needed to fit the available memory.

A better tree building policy may improve the predictive performance of OPLT. By comparing the results presented here to the ones of k-means trees, we observe a large gap. The online tree-building algorithms are not able to fully eliminate it, but we believe that the regret can be much smaller. Also, memory usage could be improved by better utilization of auxiliary classifiers.

9.6 PLT vs. state-of-the-art

In the final part of the empirical study, we compare PLTs with state-of-the-art algorithms. In this comparison we focus not only on predictive performance, but also on the computational one, measured on the same hardware, therefore we need to limit the number of compared methods. Full comparison of precision@k of multiple state-of-the-art methods is given in Appendix B.6. Here, we use two decision tree methods: FASTXML and PFASTREXML, and two smart 1-VS-ALL methods: DISMEC and PPDSPARSE. Decision tree-based methods are computationally efficient but under-perform compared to smart 1-VS-ALL methods in terms of predictive performance. We describe these methods in more detail. FASTXML, introduced in [Prabhu and Varma, 2014], uses sparse linear classifiers in internal tree nodes, also trained using LIBLINEAR. Each linear classifier decides between two classes, the left or the right child. These two classes are initiated by a random assignment of training observations to the children nodes. In the next steps, the assignment is reshaped by optimizing the normalized discounted cumulative gain (NDCG) over both children. Once the assignment stabilizes, a sparse linear classifier is trained using logistic loss. To improve the overall accuracy FASTXML uses an ensemble of trees. PFASTREXML [Jain et al., 2016] is a modification of FASTXML that optimizes propensity scored NDCG at each tree node and re-ranks the predicted labels. DISMEC trains a single classifier per label under L_2 regularized squared hinge loss, also using LIBLINEAR. It prunes weights of final models at a threshold equal 0.01 to reduce the memory needed to store a 1-VS-ALL classifier. It uses distributed training over multiple cores and processors to speed up computations. PPDSPARSE [Yen et al., 2017], in turn, parallelizes PD-SPARSE [Yen et al., 2016] which optimizes a max-margin loss by exploiting the primal-dual sparsity, resulting from the use of the max-margin loss under L_1 regularization, given that for each training observations the set of highly scored incorrect labels is small. As instances of PLTs, we use PARABEL and NAPKINXC. For the former, we use an ensemble of three trees trained with squared hinge loss. This is the first label tree algorithm being competitive to state-of-the-art, as reported in [Prabhu et al., 2018]. For NAPKINXC, we use a configuration, suggested by the results of the previous experiments, which uses also an ensemble of three trees, but with arity of 16, providing a significant predictive performance boost over binary trees, at the same keeping training and prediction times reasonably low. For training node classifiers, we use LIBLINEAR with the logistic loss for 3 data sets (AmazonCat-13K, Wiki10-30K and DeliciousLarge-200K) and squared hinge loss for the rest of the data sets.

The results are given in Table 9.13. We report precision@k, training and prediction times, and model sizes. For each not deterministic algorithm, we repeat the experiment 5 times and report means with standard errors. We use original implementations of all competitors. The hyperparameters used to tune the final models are given in Appendix B.3. For DISMEC and PPDSPARSE we report the best results found in the literature, namely from [Babbar and Schölkopf, 2017, Yen et al., 2017, Prabhu et al., 2018, Bhatia et al., 2016]. We use the provided implementations to approximate training and prediction times on our hardware. From the results, we see that PLTs are indeed competitive to the 1-VS-ALL approaches, achieving the best precision@1 on 5 from 8 data sets and is only slightly worse on the rest of the data sets. They outperform the decision tree-based methods. PLTs are almost always the fastest in training and prediction and achieve the smallest model sizes. They can be even a thousand times faster in training and prediction than 1-VS-ALL. The variant of NAPKINXC used in this experiment outperforms PARABEL in terms of precision@k by sacrificing the computational performance of training and prediction. However, it can predict in an online setting at the same time often consuming less memory.

	p@1 [%]	p@3 [%]	p@5 [%]	T_{train} [h]	T/N_{test} [ms]	M_{size} [GB]
EurLex-4K						
FastXML	71.26±0.19	59.80±0.12	50.28±0.02	0.07±0.00	0.97±0.15	0.22±0.00
PfastreXML	70.21±0.09	59.26±0.10	50.59±0.08	0.08±0.00	1.30±0.09	0.26±0.00
PPD-Sparse	**83.83**	**70.72**	**59.21**	≈ 0.02	≈ 0.70	0.07
DiSMEC	83.67	70.70	59.14	≈ 0.70	≈ 4.60	0.04
Parabel-T=3	81.80±0.10	68.67±0.03	57.45±0.06	**0.02±0.00**	**0.93±0.04**	0.03±0.00
nXC-T=3	81.94±0.24	68.94±0.07	57.49±0.14	0.03±0.00	0.97±0.06	**0.02±0.00**
AmazonCat-13K						
FastXML	93.03±0.00	78.22±0.01	63.38±0.00	5.53±0.15	1.06±0.08	18.35±0.00
PfastreXML	85.62±0.01	75.31±0.00	62.83±0.01	5.45±0.12	**0.99±0.06**	19.01±0.00
PPD-Sparse	92.72	78.14	63.41	≈ 2.97	≈ 1.20	**0.50**
DiSMEC	92.72	78.11	63.40	≈ 138.60	≈ 2.9	1.50
Parabel-T=3	93.24±0.01	79.17±0.00	64.51±0.00	**0.64±0.03**	1.05±0.04	0.62±0.00
nXC-T=3	**93.37±0.05**	**79.01±0.03**	**64.27±0.04**	2.30±0.13	**0.99±0.10**	1.01±0.00
Wiki10-30K						
FastXML	82.97±0.02	67.58±0.07	57.68±0.03	0.23±0.01	8.21±0.52	0.54±0.00
PfastreXML	75.58±0.07	64.38±0.11	57.25±0.07	0.23±0.00	10.40±0.41	1.13±0.00
PPD-Sparse	73.80	60.90	50.40	≈ 1.20	≈ 22.00	0.80
DiSMEC	85.20	**74.60**	**65.90**	≈ 26.80	≈ 112.40	2.40
Parabel-T=3	84.49±0.05	72.57±0.04	63.66±0.10	**0.20±0.00**	**2.67±0.06**	**0.18±0.00**
nXC-T=3	**85.90±0.02**	74.45±0.11	64.84±0.09	0.39±0.01	11.76±0.19	2.16±0.00
DeliciousLarge-200K						
FastXML	43.17±0.03	38.70±0.01	36.22±0.02	3.86±0.09	12.27±0.36	6.95±0.00
PfastreXML	17.44±0.02	17.28±0.01	17.19±0.01	3.71±0.02	19.64±0.35	15.34±0.00
PPD-Sparse	45.05	38.34	34.90	≈ 17.00	≈ 64.00	3.40
DiSMEC	45.50	38.70	35.50	≈ 24000.00	≈ 68.20	160.10
Parabel-T=3	46.62±0.02	39.78±0.04	36.37±0.04	9.01±0.20	**2.61±0.03**	6.36±0.00
nXC-T=3	**49.65±0.03**	**43.18±0.02**	**39.97±0.01**	**7.90±0.48**	31.10±2.87	**2.86±0.00**
WikiLSHTC-325K						
FastXML	49.85±0.00	33.16±0.01	24.49±0.01	6.41±0.13	4.10±0.04	12.93±0.00
PfastreXML	58.50±0.02	37.69±0.01	27.57±0.01	6.25±0.13	4.00±0.20	14.20±0.00
PPD-Sparse	64.13	42.10	31.14	≈ 16.00	≈ 51.00	5.10
DiSMEC	64.94	42.71	31.50	≈ 2320.00	≈ 340.00	3.80
Parabel-T=3	64.95±0.02	43.21±0.02	32.01±0.01	**0.81±0.02**	**1.27±0.03**	3.10±0.00
nXC-T=3	**65.57±0.10**	**43.64±0.11**	**32.33±0.11**	7.10±0.13	1.70±0.13	**2.68±0.00**
WikipediaLarge-500K						
FastXML	49.32±0.03	33.48±0.03	25.84±0.01	51.48±0.65	15.35±0.56	59.69±0.01
PfastreXML	59.58±0.02	40.26±0.01	30.73±0.01	51.07±0.92	15.24±0.24	69.33±0.01
PPD-Sparse	70.16	50.57	39.66	≈ 26.00	≈ 130.00	4.00
DiSMEC	**70.20**	**50.60**	**39.70**	≈ 26800.00	≈ 1200.00	14.80
Parabel-T=3	68.66±0.06	49.48±0.05	38.60±0.04	**7.33±0.12**	**3.44±0.13**	5.69±0.00
nXC-T=3	69.24±0.20	49.82±0.16	38.81±0.14	41.11±1.34	5.53±0.10	**4.68±0.01**
Amazon-670K						
FastXML	36.90±0.02	33.22±0.01	30.44±0.01	2.80±0.03	8.57±0.20	9.54±0.00
PfastreXML	36.97±0.02	34.18±0.01	32.05±0.01	3.01±0.03	9.96±0.14	10.98±0.00
PPD-Sparse	45.32	40.37	36.92	≈ 2.00	≈ 90.00	6.00
DiSMEC	**45.37**	**40.40**	**36.96**	≈ 1830.00	≈ 380.00	3.80
Parabel-T=3	44.70±0.04	39.66±0.04	35.85±0.04	**0.39±0.00**	**1.57±0.05**	1.95±0.00
nXC-T=3	45.10±0.11	40.00±0.12	36.22±0.13	2.17±0.10	1.84±0.42	**1.66±0.00**
Amazon-3M						
FastXML	45.26±0.01	41.96±0.00	39.80±0.01	18.19±1.01	68.77±4.16	30.70±0.00
PfastreXML	32.62±0.01	32.67±0.01	32.35±0.01	19.07±0.92	78.83±3.93	41.88±0.00
PPD-Sparse	-	-	-	-	-	-
DiSMEC	47.77	44.96	42.80	≈ 18800.00	≈ 2050.00	39.70
Parabel-T=3	47.52±0.01	44.69±0.01	42.57±0.00	**5.20±0.01**	**1.53±0.02**	31.43±0.00
nXC-T=3	**47.83±0.09**	**45.08±0.09**	**42.98±0.09**	25.43±1.02	4.93±0.60	**28.08±0.00**

Table 9.13: PLTs compared to state-of-the-art algorithms.

10 Discussion and open research directions

In this chapter, we briefly discuss other ideas tested during the long-term work on extreme multi-label classification. We describe two other proposed algorithms: BR-TREES [Jasinska and Dembczyński, 2015] and LTLS [Jasinska and Karampatziakis, 2016]. We include the ideas related to PLTs that have not been published, but are intriguing and insightful. We discuss the limitations of PLTs in the context of other extreme multi-label classification challenges. Finally, we mention several open research directions related to PLTs.

10.1 Other proposed methods

Let us first describe two other methods proposed during the work on extreme multi-label classification. Both of them are based on imposing a structure: a label tree in the case of the BR-TREE or a graph in the case of LTLS. These methods were not analyzed as thoroughly as PLTs, however, they may still turn out to be interesting in the future.

10.1.1 BR-trees

Jasinska and Dembczyński [2015] have proposed, besides PLTs, a BR-TREES (BRTs) algorithm. BR (standing for *binary relevance*) is another name of the 1-VS-ALL method. BR-TREEs work as an index structure over the 1-VS-ALL classifiers, allowing for retrieval of top-scoring elements without the linear scan over all m binary classifiers. Like PLTs, BRTs are based on label trees, and each node of a BRT corresponds to a classifier. However, BRTs differ from PLTs in the formulation of problems solved by the node classifiers. The leaves of a BR-TREE correspond to binary classifiers from a 1-VS-ALL classifier. Each internal node classifier of a BR-TREE predicts whether there is a reachable positive prediction among the predictions of the 1-VS-ALL binary classifiers corresponding to the leaves in the subtree of the node. To define the problem for an internal BRT node, we use a specific weighting of examples based on predictions of the children of the node. This allows us to prove a regret bound with respect to Hamming loss, which is expressed as the regret of the 1-VS-ALL approach plus a sum of *weighted false negatives* among all the classifiers.

The initial experiments on small data sets have shown that BRTs obtain prediction quality competitive to PLTs, or even outperforming it. However, as BRTs do not filter the training observations based on their positive labels, they create larger models and are more time consuming to train than PLTs. Especially, they do not reduce the training time compared to a vanilla 1-VS-ALL classifier, as they require training such classifier to create the leaf classifiers. However, the training time of BRTs may be reduced by the use of techniques such as negative sampling. Furthermore, BRTs reduce the prediction time compared to the vanilla 1-VS-ALL classifier.

10.1.2 Log-Time Log-Space

Jasinska and Karampatziakis [2016] have proposed another algorithm for extreme classification, named LOG-TIME LOG-SPACE (LTLS), characterized by both space and time complexity being logarithmic in the number of labels. LTLS represents the multi-class/multi-label problem as a structured prediction problem. More precisely, it creates a specific directed graph with m paths from the source to the sink. An example graph is depicted in Figure 10.1. Each path from source to sink represents a label, and each edge corresponds to a classifier. The prediction with LTLS is done by finding the highest scoring paths using the Viterbi algorithm. LTLS can also be viewed as a specific embedding-based method that embeds the problem with m dimensions to a problem with as many dimensions as edges in the graph, using an efficient decompression scheme based on the Viterbi algorithm.

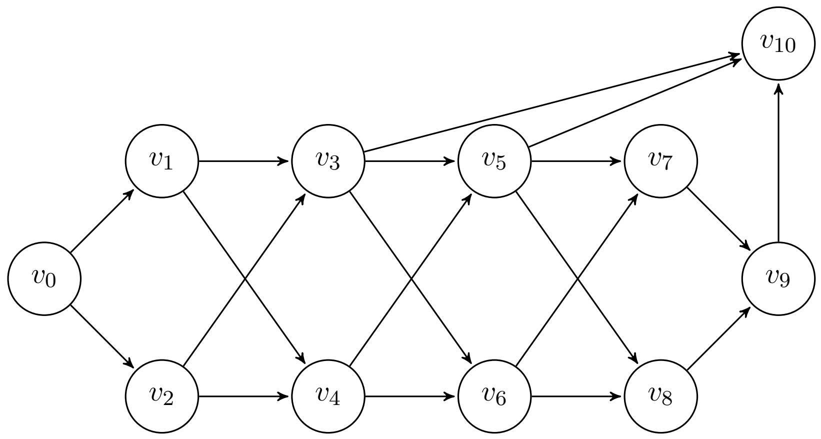


Figure 10.1: A graph with 19 edges and 22 paths from source (v_0) to sink (v_{10}), corresponding to $m = 22$ labels. There are four layers with two nodes in each layer.

We have proposed several methods of training a LTLS classifier. The first is based on incremental minimization of the ranking separation loss [Crammer and Singer, 2003, Yen et al., 2016]. For each training observation, we find the paths violating the margin: the highest scored negative path and the lowest scored positive path. We perform a positive update to the edge classifiers only on the positive path (predicting too low values), and negative updates to classifiers only on the negative path (predicting too high values). Another method, suitable only for multi-class problems, is based on minimization of the logistic loss in a conditional random fields manner [Lafferty et al., 2001]. Jasinska and Karampatziakis [2016] have experimented with the use of two kinds of edge classifiers: linear classifiers and a multi-layer neural network. Linear classifiers perform well in case of problems with very sparse features, while the use of non-linear models is necessary if the features are too dense for the linear classifiers to properly fit the heavily compressed problems constructed by LTLS.

The performance of LTLS is influenced by the assignment of labels to paths in the graph. This assignment plays a similar role to the tree structure in label tree-based methods. We hypothesize that the optimal path assignment is the one in which the paths of frequently co-occurring labels have many common edges. However, as finding such an assignment is computationally complex, Jasinska and Karampatziakis [2016] have proposed a greedy assignment of new labels to the highest scoring paths.

LTLS may be used for both multi-class and multi-label problems. Empirically, it outperforms LOMTREE [Choromanska and Langford, 2015] on multi-class problems. On multi-label problems, it underperforms compared to more computationally intensive extreme classification methods. However, it outperforms a simple baseline classifier of the same model size and prediction cost: a 1-VS-ALL classifier trained for as many most frequent labels as there are edges in the LTLS graph. LTLS has been extended by WIDE-LTLS [Evron et al., 2018], which uses a wider graph with more than two nodes in each layer. The use of more nodes in each layer allowed WIDE-LTLS to obtain better predictive performance. To increase the predictive performance of LTLS, we researched the correction of classification errors by the use of redundant paths. This did not lead to performance gains, however other methods of improving the performance by modifying the graph structure are possible. An interesting research direction related to LTLS is the minimization of the binary cross-entropy loss for multi-label problems. Another one is the use of LTLS as a deep network output layer.

10.2 Tree structure

Let us now move on to probabilistic label trees. The tree structure impacts both the predictive and computational performance of a PLT classifier. In this dissertation, we focused on several structures: a random baseline, a k-means-based built top-down using k-means clustering algorithm on simple representations of labels, and one built online with OPLTs using a simple tree-building policy.

However, during the work on PLTs, the problem of tree structure choice was approached from many other perspectives.

Efforts were made to formally define what is an optimal tree structure and to create an optimization procedure that delivers such. Multiple criteria must be taken into account when defining such an optimal tree. On one hand, we know that the L_1 error of estimation of conditional probabilities of labels $\hat{\eta}_j(\boldsymbol{x})$ accumulates over all node classifiers on $\mathrm{Path}(j)$, see Lemma 5.3. To minimize this bound by changing the tree structure, one may consider using a two-level tree with just a root and m leaves. If at least one label is positive for each observation, the regret of the root can be considered zero, and this degenerated PLT model boils down to a 1-VS-ALL. However, a PLT with such tree structure suffers from long training and prediction time, and large model size, which are exactly the problems of 1-VS-ALL PLTs aim at solving. On the other hand, when optimizing the training complexity, one ends up with a well-defined but NP-complete problem, as Busa-Fekete et al. [2019] show. However, an optimal tree with respect to the training complexity may not be the best performing one in terms of the quality of predictions. There exists an intriguing link between tree structure, the learning problems, and the training and prediction complexity. However, formally defining an optimal tree is out of the scope of this work. Instead, we recall two methods we tested during the long-term works on PLTs and extend the discussion of OPLT tree building policies.

10.2.1 Spectral tree

The initially [Jasinska and Dembczyński, 2015] proposed tree structure building method is based on an observation, that some label pairs co-occur more often than other pairs. The proposed method builds a tree structure top-down. To split the labels between two children of a node, it solves a min-cut graph problem on a graph with nodes corresponding to labels, with edges existing between co-occurring ones, weighted with the number of their co-occurrences. This method is called a *spectral* tree, as it uses spectral clustering, or more precisely, a Fiedler vector, being the eigenvector corresponding to the second-smallest eigenvalue of the Laplacian matrix of the label graph, to partition the graph. This method was tested on the smallest benchmark data sets and used in [Jasinska and Dembczyński, 2015]. Unfortunately, it did not scale to the commonly used benchmark data sets.

10.2.2 FastPLT

The inspiration for the other proposed approach is the alternating optimization procedure from FASTXML [Prabhu and Varma, 2014]. FASTXML is a decision-tree with logistic regression models in the decision nodes. The decision boundaries of the logistic regression models split the feature space and guide instances in the process of prediction. FASTXML simultaneously defines the splits of the training observations to the left and right children and trains the logistic regression models to predict the left/right class for the test instances. The training of the decision node classifier and splitting of the training observations are two alternating steps of the optimization procedure. This procedure starts with a random split of training observations to left and right. Then, to optimize a multi-criteria objective, it alternately solves a logistic regression problem for a fixed split of training observations and modifies the assignment of the left and right classes to training observations. If needed, FASTXML repeats the alternating optimization procedure several times until convergence to a local optimum.

This idea was applied in PLTs to build the tree top-down by minimizing the total logistic loss of neighboring classifiers. The proposed method simultaneously splits the labels among two children of a node and trains the two models corresponding to these nodes. It minimizes the total logistic loss of the two trained classifiers in an alternating optimization procedure. First, it randomly splits labels into two balanced clusters. Then, it trains two logistic regression classifiers using the current label split. Finally, it reassigns labels to two balanced clusters in a way minimizing the total logistic loss given the fixed logistic regression classifiers. It repeats the logistic regression training and labels reassignment until no significant improvement of the total logistic loss.

The proposed methods have several flaws. First, it leads to overfitting and requires regularization or early stopping based on a validation set. Moreover, empirical observations have shown that the optimized criterion, total logistic loss with the balanced split constraint, easily leads to imbalanced quality of predictions between clusters. Also, the undesirable split of labels according to their prior probability tends to give low values of the minimized criterion. Due to the long run times of this approach and the aforementioned flaws, this method was no longer investigated.

10.2.3 Online PLT tree building policies

In this work, we describe only the most basic OPLT tree building policy. However, other possibilities were researched, starting from the application of CPT policies from [Beygelzimer et al., 2009a]. Such a simple policy extends the leaf, see Figure 7.1(d), with the highest score given current observation with the new label. Such policy leads to undesirable properties of the constructed PLT tree: labels occurring together in the observations from the initial training steps easily end up in the most distant tree parts. This shows the importance of the proposed additional nodes extensions, see Figure 7.1 (b) and (c), which are missing in the original CPT building procedure. However, it also shows the complexity of the online tree building problem.

Jasinska-Kobus et al. [2020c] give additional insights about OPLT, presenting the ongoing work on the policies. They include a policy based on a trade-off between balancedness of the tree and the fit of the new observations to the existing classifiers. It is also an extension of the CPT policy that also trades-off these two criteria. Other options like building a k-means based tree using a sample of training data, and extending it with the OPLT methods, were researched by other co-authors, and are described in [Jasinska-Kobus et al., 2020d].

10.3 ϵ-approximate prediction

The computational efficiency of the PLT classifiers has always been important, so other faster prediction algorithms with provable guarantees were researched. [Dembczyński et al., 2016] analyses multi-class probabilistic classifier trees and proposes the ϵ-approximate prediction algorithm for approximate inference of the highest-scoring class. This algorithm enables bounding the prediction cost by a function of the distance, ϵ, from the optimal prediction. The ϵ-approximate prediction algorithm and the mentioned bound are suited for multi-class classification. This algorithm was adopted to multi-label problems to reduce the prediction time of the uniform-cost search based prediction algorithm for PLTs. Unfortunately, in the multi-label case, we observed a significant drop in predictive performance and just a minor drop in prediction time. Therefore we did not continue the research on the ϵ-approximate prediction algorithms.

10.4 Limitations of PLTs in the context of other extreme classification challenges

In this section, we discuss PLTs in the context of extreme classification challenges that we skipped in the main work. We consider the problem of rare labels and the problem of positive-unlabeled labels.

10.4.1 The problem of rare labels

The performance of the node classifiers determines the performance of a PLT classifier. To train well-performing node classifiers, one needs enough data. However, as Table 9.1 shows, the average number of observations per label may be very small. Consider the learning problem in a leaf node with only one neighbor. Assume that the label corresponding to the leaf is very rare and the one corresponding to the leaf's neighbor is frequent. In such a case, the binary classification problem corresponding to the leaf is extremely imbalanced. If both labels are rare, the classification problem is balanced, but there is too little training data to properly fit a binary classifier. In both cases the quality of the node classifiers may be very poor. A possible solution to this problem is treating rare labels differently from the frequent ones, for example, by using the nearest neighbor classifier for those labels. However, we did not explore this research direction.

Another aspect of the problem of rare labels is the impact of performance metric choice. The most popular performance metrics: precision@k and NDCG@k, take into account only the k predicted labels. However, frequently an observation contains several frequent and several infrequent ones. Therefore, to achieve good performance in terms of @k metrics is often enough to be able to predict only the frequent ones. This way, the problem of estimation of conditional probabilities of rare labels is often overlooked. The use of metrics like macro-F_1-measure, analyzed in detail in this work, can help in diagnosing this problem and foster the research on the estimation of probabilities of rare labels. Interestingly, using a data set prepared from a Wikipedia dump on our own, we observed that the use of the threshold-based prediction suited for macro-F_1-measure can also lead to the prediction of labels that are not given as relevant, but are, in fact, relevant. This leads us to another extreme multi-label classification problem: the positive-unlabeled labels.

10.4.2 The positive-unlabeled labels problem

Another characteristic of extreme multi-label classification problems is that frequently some of the truly relevant labels are given as irrelevant, i.e., they are not observed. PLTs treat negative labels as truly negative, and the existence of the positive-unlabeled labels makes the process of solving the binary problems harder. Currently, PLTs do not take this problem into account, however, doing so might lead to an improvement in their overall performance.

Also, we do not consider the propensity scored variants of performance metrics. However, it is possible to adjust PLTs to some of them, as we discuss below. Propensity q_j of label j is the probability that label j is positive (observed) given that it is truly relevant. Propensity scored metrics are defined in [Jain et al., 2016]. Consider propensity scored precision@k:

$$p_{\mathrm{q}}@k(\boldsymbol{y}, \boldsymbol{h}_{@k}(\boldsymbol{x})) = \frac{1}{k} \sum_{j \in \hat{\mathcal{L}}_{\boldsymbol{x}}} \frac{1}{q_j} [\![y_j = 1]\!] \,.$$

Notice that the Bayes optimal classifier for propensity scored precision@k is determined by the conditional probabilities of labels scaled by the inverse of the label propensity. Given that the propensities (or their estimates) are given in the time of prediction, and of course in the time of evaluation to compute the propensity scored metrics, propensity scored precision@k might be optimized using PLTs. This would require a customized inference procedure for PLTs, ranking the labels according to their estimates of conditional probabilities scaled by label-wise inverse propensities.

10.5 Open research directions

There exist many other open problems related to probabilistic label trees. One of them includes making PLTs suitable for more performance metrics. In Chapter 3 we discussed recall@k and NDCG@k, which are not optimized by PLTs. As long applying the pick-one-label heuristic gives the Bayes classifier for recall@k, we observed that PLTs using original data perform better. However, using other methods to make PLTs suitable for recall@k may lead to further improvement of performance. On the other hand, NDCG@k requires sorting according to specific marginal values. Making PLTs suitable for NDCG@k would be a major improvement, however, it is unclear how it should be done. Another important direction is to make PLTs suitable for the propensity scored variants of the performance metrics, as discussed in the previous section.

An interesting direction of development of PLTs is the use of other probabilistic binary classifiers than linear ones as the node classifiers. The node classifiers can be, for example, the outputs of a deep network. Such an idea was initially explored by EXTREMETEXT and ATTENTIONXML. Also, other probabilistic binary classifiers could be plugged in as the node classifiers. This includes also extending the idea of estimating the node probabilities via multi-class classification from Section 8.8.

PLTs also can be applied in other applications. The quality of PLTs was not yet tested in recommendation problems, which are also considered as extreme classification. If PLTs prove to perform well, they may become a computationally efficient alternative to other more advanced recommendation approaches. PLTs could also be used to sample negatives during 1-VS-ALL classifier training, similarly to hierarchical softmax used in [Bamler and Mandt, 2020]. In such a case, PLTs would not be the final classifier, but a tool allowing for faster training of a more complex one.

Since the first publication, PLTs have become a popular approach to extreme multi-class classification. They were applied in many classifiers [Prabhu et al., 2018, Khandagale et al., 2019, You et al., 2019], and analyzed in other works [Zhuo et al., 2020]. Therefore we expect that even more research directions related to PLTs will appear in near future.

11 Summary

In this dissertation, we investigated probabilistic label trees, a computationally efficient and statistically well-justified model for solving extreme multi-label classification problems. The in-depth analysis shows that PLTs can scale logarithmically with the number of labels and are suitable for optimizing a wide spectrum of performance metrics commonly used in extreme multi-label classification.

The main goal of this dissertation, as stated in Section 1.4, was to show the existence of a class of statistically consistent learning algorithms for extreme multi-label classification whose computational complexity scales sub-linearly with the number of labels. We believe that this goal has been achieved. In support of this claim, we summarize below the relevant contributions of this dissertation.

We described and analyzed the proposed class of learning algorithms. We gave batch and incremental training algorithms working with the tree structure given in advance. Then, we analyzed in-depth its consistency. We derived the bound of L_1 estimation error of conditional probabilities of labels via L_1 errors of the node classifiers constituting a PLT classifier. By applying the bound of the L_1 error of node classifiers in terms of their regret with respect to a strongly proper composite loss, we connected the learning algorithm used to train node classifiers, and their regret, with the L_1 estimation error of conditional probabilities of labels. We showed the relevant regret bounds for precision@k, DCG@k, and generalized classification performance metrics. This way we demonstrated the statistical consistency of PLTs with respect to these popular metrics, for which the optimal decisions are determined through conditional probabilities of labels. We also described an online learning algorithm building the tree structure simultaneously with the classifiers' training, creating a PLT classifier equivalent to the one trained incrementally.

We analyzed the computational complexity of the proposed methods. We demonstrated non-trivial results related to the complexity of training and prediction. We considered a computational cost, defined as the number of PLT nodes processed during prediction or training. Training-wise, we demonstrated conditions under which PLTs have training complexity logarithmic in the number of labels. Prediction-wise, we considered the computational complexity of threshold-based, beam-search-based, and uniform-cost-based prediction algorithms, and showed that under certain assumptions these are sublinear in the number of labels. Additionally, we showed that OPLT is the same as the complexity of an incremental PLT algorithm.

Finally, we showed that the proposed approach not only good theoretical properties but also can be efficiently implemented in various settings achieving state-of-the-art results both in terms of predictive and computational performance. The number of works following the PLTs approach confirms the validity of this method.

Bibliography

S. Agarwal. Surrogate regret bounds for bipartite ranking via strongly proper losses. *Journal of Machine Learning Research*, 15:1653–1674, 2014.

R. Agrawal, A. Gupta, Y. Prabhu, and M. Varma. 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*, page 13–24, New York, NY, USA, 2013. Association for Computing Machinery.

R. Babbar and B. Schölkopf. Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, page 721–729, New York, NY, USA, 2017. Association for Computing Machinery.

R. Babbar and B. Schölkopf. Data scarcity, robustness and extreme multi-label classification. *Machine Learning, Special Issue of the ECML PKDD 2019 journal Track*, 108, 2019.

R. Babbar and B. Schölkopf. Adversarial extreme multi-label classification. *CoRR*, abs/1803.01570, 2018.

R. Bamler and S. Mandt. Extreme classification via adversarial softmax approximation. In *International Conference on Learning Representations*, 2020.

P. Bartlett. Cs281b/stat241b. statistical learning theory. lecture 2., 2014. URL `https://www.stat.berkeley.edu/~bartlett/courses/2014spring-cs281bstat241b/lectures/02-notes.pdf`.

S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *Advances in Neural Information Processing Systems 23*, pages 163–171. Curran Associates, Inc., 2010.

A. Beygelzimer, J. Langford, Y. Lifshits, G. Sorkin, and A. Strehl. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-*

Fifth Conference on Uncertainty in Artificial Intelligence, page 51–58, Arlington, Virginia, USA, 2009a. AUAI Press.

A. Beygelzimer, J. Langford, and P. Ravikumar. Error-correcting tournaments. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory*, page 247–262, Berlin, Heidelberg, 2009b. Springer-Verlag.

A. Beygelzimer, H. Daumé, J. Langford, and P. Mineiro. Learning reductions that really work. *Proceedings of the IEEE*, 104:136–147, 2016.

K. Bhatia, H. Jain, P. Kar, M. Varma, and P. Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems 28*, pages 730–738. Curran Associates, Inc., 2015.

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

R. Busa-Fekete, B. Szörényi, K. Dembczynski, and E. Hüllermeier. Online f-measure optimization. In *Advances in Neural Information Processing Systems 28*, pages 595–603. Curran Associates, Inc., 2015.

R. Busa-Fekete, K. Dembczynski, A. Golovnev, K. Jasinska, M. Kuznetsov, M. Sviridenko, and C. Xu. On the computational complexity of the probabilistic label tree algorithms. *CoRR*, abs/1906.00294, 2019.

P. Celis, P.-A. Larson, and J. I. Munro. Robin hood hashing. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*, page 281–288, USA, 1985. IEEE Computer Society.

W. Chang, H. Yu, K. Zhong, Y. Yang, and I. S. Dhillon. A modular deep learning approach for extreme multi-label text classification. *CoRR*, abs/1905.02331, 2019.

A. E. Choromanska and J. Langford. Logarithmic time online multiclass prediction. In *Advances in Neural Information Processing Systems 28*, pages 55–63. Curran Associates, Inc., 2015.

K. Crammer and Y. Singer. A family of additive online algorithms for category ranking. *Journal of Machine Learning Research*, 3:1025–1058, 2003.

C. De Boom, S. Van Canneyt, T. Demeester, and B. Dhoedt. Representation learning for very short texts using weighted word embedding aggregation. *Pattern Recognition Letters*, 80:150–156, 2016.

O. Dekel and O. Shamir. Multiclass-multilabel classification with more classes than examples. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 137–144, Chia Laguna Resort, Sardinia, Italy, 2010. PMLR.

K. Dembczyński, W. Cheng, and E. Hüllermeier. Bayes optimal multilabel classification via probabilistic classifier chains. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, page 279–286, Madison, WI, USA, 2010. Omnipress.

K. Dembczyński, W. Kotłowski, and E. Hüllermeier. Consistent multilabel ranking through univariate loss minimization. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, page 1347–1354, Madison, WI, USA, 2012. Omnipress.

K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence and loss minimization in multi-label classification. *Machine Learning*, 88: 5–45, 2012.

K. Dembczyński, W. Kotłowski, W. Waegeman, R. Busa-Fekete, and E. Hüllermeier. Consistency of probabilistic classifier trees. In *ECML PKDD 2016 : machine learning and knowledge discovery in databases*, pages 511–526. Springer, 2016.

J. Deng, S. Satheesh, A. C. Berg, and F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems 24*, pages 567–575. Curran Associates, Inc., 2011.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

J. Duchi and Y. Singer. Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10:2899–2934, 2009.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12: 2121–2159, 2011.

I. Evron, E. Moroshko, and K. Crammer. Efficient loss-based decoding on graphs for extreme classification. In *Advances in Neural Information Processing Systems 31*, pages 7233–7244. Curran Associates, Inc., 2018.

R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

J. Fox. *Applied regression analysis, linear models, and related methods*. Sage, 1997.

W. Gao and Z. Zhi-Hua. On the consistency of multi-label learning. *Artificial Intelligence*, 199-200:22–44, 2013.

E. Grave, A. Joulin, M. Cissé, D. G. Facebook AI Research, and H. Jégou. Efficient softmax approximation for gpus. In *Proceedings of the 34th International Conference on Machine Learning - volume 70*, page 1302–1310. JMLR.org, 2017.

C. Guo, A. Mousavi, X. Wu, D. N. Holtmann-Rice, S. Kale, S. Reddi, and S. Kumar. Breaking the glass ceiling for embedding-based classifiers for large output spaces. In *Advances in Neural Information Processing Systems 32*, pages 4943–4953. Curran Associates, Inc., 2019.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

H. Jain, Y. Prabhu, and M. Varma. 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*, page 935–944, New York, NY, USA, 2016. Association for Computing Machinery.

K. Jasinska. Efficient exact batch prediction for label trees. In *Extreme Multilabel Classification for Social Media at The Web Conference*, 2018.

K. Jasinska and K. Dembczyński. Consistent label tree classifiers for extreme multi-label classification. In *The ICML Workshop on Extreme Classification*, 2015.

K. Jasinska and K. Dembczyński. Bayes optimal prediction for ndcg@k in extreme amulti-label classification. In *From Multiple Criteria Decision Aid to Preference Learning Workshop*, 2018.

K. Jasinska and N. Karampatziakis. Log-time and log-space extreme classification. *CoRR*, abs/1611.01964, 2016.

K. Jasinska, K. Dembczynski, R. Busa-Fekete, K. Pfannschmidt, T. Klerx, and E. Hullermeier. Extreme f-measure maximization using sparse probability estimates. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1435–1444, New York, USA, 2016. PMLR.

K. Jasinska, K. Dembczyński, and N. Karampatziakis. Extreme classification under limited space and time budget. *Schedae Informaticae*, 2017. doi: 10.4467/20838476SI.16.001.6182.

K. Jasinska-Kobus, M. Wydmuch, K. Dembczyński, M. Kuznetsov, and R. Busa-Fekete. Probabilistic label trees for extreme multi-label classification. *Journal of Machine Learning Research (in review)*, 2020a.

K. Jasinska-Kobus, M. Wydmuch, K. Dembczyński, M. Kuznetsov, and R. Busa-Fekete. Probabilistic label trees for extreme multi-label classification. *CoRR*, abs/2009.11218, 2020b.

K. Jasinska-Kobus, M. Wydmuch, D. Thiruvenkatachari, and K. Dembczyński. Online probabilistic label trees. *CoRR*, abs/2007.04451, 2020c.

K. Jasinska-Kobus, M. Wydmuch, D. Thiruvenkatachari, and K. Dembczyński. Online probabilistic label trees. *AISTATS 2020 (in review)*, 2020d.

Y. Jernite, A. Choromanska, and D. Sontag. Simultaneous learning of trees and representations for extreme classification and density estimation. In *Proceedings of the 34th International Conference on Machine Learning - volume 70*, page 1665–1674. JMLR.org, 2017.

A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: volume 2, Short Papers*, pages 427–431, Valencia, Spain, 2017. Association for Computational Linguistics.

S. Khandagale, H. Xiao, and R. Babbar. Bonsai - diverse and shallow trees for extreme multi-label classification. *CoRR*, abs/1904.08249, 2019.

W. Kotłowski and K. Dembczyński. Surrogate regret bounds for generalized classification performance metrics. *Machine Learning*, 10:549–572, 2017.

O. O. Koyejo, N. Natarajan, P. K. Ravikumar, and I. S. Dhillon. Consistent multilabel classification. In *Advances in Neural Information Processing Systems 28*, pages 3321–3329. Curran Associates, Inc., 2015.

M. Kurzynski. On the multistage bayes classifier. *Pattern Recognition*, 21:355–365, 1988.

J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 282–289, 2001.

J. Langford, A. Strehl, and L. Li. Vowpal wabbit, 2007. URL `http://hunch.net/~vw/`.

C.-L. Li and H.-T. Lin. Condensed filter tree for cost-sensitive multi-label classification. In *Proceedings of the 31st International Conference on Machine Learning*, pages 423–431, Bejing, China, 2014. PMLR.

J. Liu, W.-C. Chang, Y. Wu, and 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*, page 115–124, New York, NY, USA, 2017. Association for Computing Machinery.

M. Majzoubi and A. Choromanska. Ldsm: Logarithm-depth streaming multi-label decision trees. *CoRR*, abs/1905.10428, 2019.

A. K. Menon, A. S. Rawat, S. Reddi, and S. Kumar. Multilabel reductions: what is my loss optimising? In *Advances in Neural Information Processing Systems 32*, pages 10600–10611. Curran Associates, Inc., 2019.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

F. Morin and Y. Bengio. Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 246–252. Society for Artificial Intelligence and Statistics, 2005.

J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, 2014. Association for Computational Linguistics.

Y. Prabhu and 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*, page 263–272, New York, NY, USA, 2014. Association for Computing Machinery.

Y. Prabhu, A. Kag, S. Harsola, R. Agrawal, and M. Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*, page 993–1002, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee.

S. Puthiya Parambath, N. Usunier, and Y. Grandvalet. Optimizing f-measures by cost-sensitive classification. In *Advances in Neural Information Processing Systems 27*, pages 2123–2131. Curran Associates, Inc., 2014.

P. Ravikumar, A. Tewari, and E. Yang. On ndcg consistency of listwise ranking methods. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 618–626, Fort Lauderdale, FL, USA, 2011. PMLR.

S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, New Jersey, USA, 2009.

W. Siblini, P. Kuntz, and F. Meyer. CRAFTML, an efficient clustering-based random forest for extreme multi-label learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4664–4673, Stockholmsmässan, Stockholm Sweden, 2018. PMLR.

Y. Tagami. 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*, page 455–464, New York, NY, USA, 2017. Association for Computing Machinery.

F. Tai and H.-T. Lin. Multilabel classification with principal label space transformation. *Neural Computation*, 24:2508–2542, 2012.

A. Tewari and P. Bartlett. An overview of learning theory, 2013. URL `http://dept.stat.lsa.umich.edu/~tewaria/research/tewari13learning.pdf`.

G. Tsoumakas, I. Katakis, and I. Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *Proceedings of ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, 2008.

K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, page 1113–1120, New York, NY, USA, 2009. Association for Computing Machinery.

J. Weston, A. Makadia, and H. Yee. Label partitioning for sublinear ranking. In *Proceedings of the 30th International Conference on Machine Learning*, pages 181–189, Atlanta, Georgia, USA, 2013. PMLR.

M. Wydmuch, K. Jasinska, M. Kuznetsov, R. Busa-Fekete, and K. Dembczynski. A no-regret generalization of hierarchical softmax to extreme multi-label classification. In *Advances in Neural Information Processing Systems 31*, pages 6355–6366. Curran Associates, Inc., 2018.

I. E. Yen, X. Huang, W. Dai, P. Ravikumar, I. Dhillon, and E. Xing. 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*, page 545–553. Association for Computing Machinery, 2017.

I. E.-H. Yen, X. Huang, P. Ravikumar, K. Zhong, and I. Dhillon. Pd-sparse : A primal and dual sparse approach to extreme multiclass and multilabel classification. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 3069–3077, New York, New York, USA, 2016. PMLR.

R. You, Z. Zhang, Z. Wang, S. Dai, H. Mamitsuka, and S. Zhu. Attentionxml: Label tree-based attention-aware deep model for high-performance extreme multi-label text classification. In *Advances in Neural Information Processing Systems 32*, pages 5820–5830. Curran Associates, Inc., 2019.

J. Zhuo, Z. Xu, W. Dai, H. Zhu, H. Li, J. Xu, and K. Gai. Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning*, Vienna, Austria, 2020. PMLR.

A Omitted proofs

A.1 Proofs of the results from Sections 3.2 and 3.3

To prove the Theorem 3.2 and Theorem 3.6, we will first prove a lemma generalizing those two results.

Lemma A.1. *Given conditionally independent labels, a label ranking $\pi(\eta(\boldsymbol{x}))$ according to their conditional probabilities $\eta_j(\boldsymbol{x})$, and a ranking of labels $\pi(\delta)$ according to scores δ,*

$$\delta_j(\boldsymbol{x}) = \sum_{\boldsymbol{y}:y_j=1} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})\,. \tag{A.1}$$

where $n(||\boldsymbol{y}||_1)$ is a function depending only on $||\boldsymbol{y}||_1 = \sum_{j'=1}^{m} y_{j'}$, are the same.

Proof. To prove the theorem we show that for conditionally independent labels the order of labels induced by the conditional probabilities $\eta_j(\boldsymbol{x})$ is the same as the order induced by the values of $\delta_j(\boldsymbol{x})$. In other words, for any two labels $i, j \in \{1, \ldots, m\}$, $i \neq j$, $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Leftrightarrow \delta_i(\boldsymbol{x}) \geq \delta_j(\boldsymbol{x})$.

Let $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x})$. Then in the summation over all $\boldsymbol{y}$ in (A.1), which can be rewritten in the following way

$$\delta_j(\boldsymbol{x}) = \sum_{\boldsymbol{y}\in\mathcal{Y}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})y_j,$$

we consider four subsets of $\mathcal{Y}$, creating a partition of this set:

$$\mathcal{S}_{i,j}^{u,w} = \{\boldsymbol{y} \in \mathcal{Y} : y_i = u \wedge y_j = w\}, u, w \in \{0, 1\}.$$

The subset $\mathcal{S}_{i,j}^{0,0}$ does not play any role because $y_i = y_j = 0$ and therefore does not contribute to the final sum. Then $\delta_j(\boldsymbol{x})$ can be written in the following way:

$$\delta_i(\boldsymbol{x}) = \sum_{\boldsymbol{y}:\mathcal{S}_{i,j}^{1,0}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) + \sum_{\boldsymbol{y}\in\mathcal{S}_{i,j}^{1,1}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \tag{A.2}$$

$$\delta_j(\boldsymbol{x}) = \sum_{\boldsymbol{y}:\mathcal{S}_{i,j}^{0,1}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) + \sum_{\boldsymbol{y}\in\mathcal{S}_{i,j}^{1,1}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \tag{A.3}$$

The contribution of elements from $\mathcal{S}_{i,j}^{1,1}$ is equal for both $\delta_i(\boldsymbol{x})$ and $\delta_j(\boldsymbol{x})$. It is so because the value of $n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$ is the same for all $\boldsymbol{y} \in \mathcal{S}_{i,j}^{1,1}$: the conditional joint probabilities $\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$ are fixed and they are multiplied by the same factors $n(||\boldsymbol{y}||_1)$.

Consider now the contributions of $\mathcal{S}_{i,j}^{1,0}$ and $\mathcal{S}_{i,j}^{0,1}$ to the relevant sums. By the definition of $\mathcal{Y}$, $\mathcal{S}_{i,j}^{1,0}$, and $\mathcal{S}_{i,j}^{0,1}$, there exists bijection $b_{i,j} : \mathcal{S}_{i,j}^{1,0} \to \mathcal{S}_{i,j}^{0,1}$, such that for each $\boldsymbol{y}' \in \mathcal{S}_{i,j}^{1,0}$ there exists $\boldsymbol{y}'' \in \mathcal{S}_{i,j}^{0,1}$ equal to $\boldsymbol{y}'$ except on the i-th and the j-th position.

Notice that because of the conditional independence assumption the joint probabilities of elements in $\mathcal{S}_{i,j}^{1,0}$ and $\mathcal{S}_{i,j}^{0,1}$ are related to each other. Let $\boldsymbol{y}'' = b_{i,j}(\boldsymbol{y}')$, where $\boldsymbol{y}' \in \mathcal{S}_{i,j}^{1,0}$ and $\boldsymbol{y}'' \in \mathcal{S}_{i,j}^{0,1}$. The joint probabilities are:

$$\mathbf{P}(\boldsymbol{y}'|\boldsymbol{x}) = \eta_i(\boldsymbol{x})(1-\eta_j(\boldsymbol{x})) \prod_{l\in\mathcal{L}\setminus\{i,j\}} \eta_l(\boldsymbol{x})^{y_l}(1-\eta_l(\boldsymbol{x}))^{1-y_l}$$

and

$$\mathbf{P}(\boldsymbol{y}''|\boldsymbol{x}) = (1-\eta_i(\boldsymbol{x}))\eta_j(\boldsymbol{x}) \prod_{l\in\mathcal{L}\setminus\{i,j\}} \eta_l(\boldsymbol{x})^{y_l}(1-\eta_l(\boldsymbol{x}))^{1-y_l}.$$

One can easily notice the relation between these probabilities:

$$\mathbf{P}(\boldsymbol{y}'|\boldsymbol{x}) = \eta_i(\boldsymbol{x})(1-\eta_j(\boldsymbol{x}))q_{i,j}$$

and

$$\mathbf{P}(\boldsymbol{y}''|\boldsymbol{x}) = (1-\eta_i(\boldsymbol{x}))\eta_j(\boldsymbol{x})q_{i,j},$$

where $q_{i,j} = \prod_{l\in\mathcal{L}\setminus\{i,j\}} \eta_l(\boldsymbol{x})^{y_l}(1-\eta_l(\boldsymbol{x}))^{1-y_l} \geq 0$. Consider now the difference of these two probabilities:

$$\begin{aligned}
\mathbf{P}(\boldsymbol{y}'|\boldsymbol{x}) - \mathbf{P}(\boldsymbol{y}''|\boldsymbol{x}) &= \eta_i(\boldsymbol{x})(1-\eta_j(\boldsymbol{x}))q_{i,j} - (1-\eta_i(\boldsymbol{x}))\eta_j(\boldsymbol{x})q_{i,j} \\
&= q_{i,j}(\eta_i(\boldsymbol{x})(1-\eta_j(\boldsymbol{x})) - (1-\eta_i(\boldsymbol{x}))\eta_j(\boldsymbol{x})) \\
&= q_{i,j}(\eta_i(\boldsymbol{x}) - \eta_i(\boldsymbol{x})\eta_j(\boldsymbol{x}) - \eta_j(\boldsymbol{x}) + \eta_i(\boldsymbol{x})\eta_j(\boldsymbol{x})) \\
&= q_{i,j}(\eta_i(\boldsymbol{x}) - \eta_j(\boldsymbol{x})).
\end{aligned}$$

From the above we see that $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Rightarrow \mathbf{P}(\boldsymbol{y}'|\boldsymbol{x}) \geq \mathbf{P}(\boldsymbol{y}''|\boldsymbol{x})$. Due to the properties of the bijection $b_{i,j}$, the number of positive labels in $\boldsymbol{y}'$ and $\boldsymbol{y}''$ is the same and $n(||\boldsymbol{y}'||_1) = n(||\boldsymbol{y}''||_1)$, therefore we also get $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Rightarrow \sum_{\boldsymbol{y}:\mathcal{S}_{i,j}^{1,0}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) \geq \sum_{\boldsymbol{y}:\mathcal{S}_{i,j}^{0,1}} n(||\boldsymbol{y}||_1)\mathbf{P}(\boldsymbol{y}|\boldsymbol{x})$, which finally based on (A.2) and (A.3) gives us $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Rightarrow \delta_i(\boldsymbol{x}) \geq \delta_j(\boldsymbol{x})$.

The implication in the other side, i.e., $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Leftarrow \mathbf{P}(\boldsymbol{y}'|\boldsymbol{x}) \geq \mathbf{P}(\boldsymbol{y}''|\boldsymbol{x})$

holds obviously for $q_{i,j} > 0$. For $q_{i,j} = 0$, we can notice, however, that $\mathbf{P}(\boldsymbol{y}'|\boldsymbol{x})$ and $\mathbf{P}(\boldsymbol{y}''|\boldsymbol{x})$ do not contribute to the appropriate sums as they are zero, and therefore we can follow a similar reasoning as above, concluding that $\eta_i(\boldsymbol{x}) \geq \eta_j(\boldsymbol{x}) \Leftarrow \delta_i(\boldsymbol{x}) \geq \delta_j(\boldsymbol{x})$.

Thus for conditionally independent labels, the order of labels induced by conditional probabilities $\eta_j(\boldsymbol{x})$ is equal to the order induced by $\delta_j(\boldsymbol{x})$. □

Having Lemma A.1, we can easily prove Corollary 3.3 and Theorem 3.6, which are special cases of the previous result.

Theorem 3.2. *Given conditionally independent labels, $\eta_j(\boldsymbol{x})$ and $\eta'_j(\boldsymbol{x})$, $j \in \mathcal{L}$ induce the same order of labels.*

Proof. Notice that the values of $\eta'_j(\boldsymbol{x})$:

$$\eta'_j(\boldsymbol{x}) = \mathbf{P}'(y_j = 1 \mid \boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}} \frac{y_j}{\sum_{j'=1}^m y_{j'}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} \frac{1}{\sum_{j'=1}^m y_{j'}} \mathbf{P}(\boldsymbol{y} \mid \boldsymbol{x}),$$

have the form of (A.1) with $n(a) = \frac{1}{a}$. The rest follows from Lemma A.1. □

Theorem 3.6. *Given conditionally independent labels, $\eta_j(\boldsymbol{x})$ and $\Delta_j(k, \boldsymbol{x})$, $j \in \mathcal{L}$ induce the same order of labels.*

Proof. Notice that the values of $\Delta_j(k, \boldsymbol{x})$:

$$\Delta_j(k, \boldsymbol{x}) = \sum_{\boldsymbol{y} \in \mathcal{Y}} y_j N_k(\boldsymbol{y}) \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}) = \sum_{\boldsymbol{y}: y_j = 1} N_k(\boldsymbol{y}) \mathbf{P}(\boldsymbol{y}|\boldsymbol{x}),$$

have the form of (A.1) with $n(a) = (\sum_{r=1}^{\min(k,a)} g(r))^{-1}$, which values depend only on a, given fixed k. The rest follows from Lemma A.1. □

A.2 Proofs of the results from Section 5.1

Before we prove Lemma 5.3 and Theorem 5.4, we first present an additional lemma. This lemma concerns expectation of $\eta_{\mathrm{pa}(v')}(\boldsymbol{x}) \, |\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')|$, the weighted L_1 error in node v used in upper bounds from Lemma 5.1 and Corollary 5.2. We express this expectation by the expected L_1 error in node v multiplied by $\mathbf{P}(z_{\mathrm{pa}(\boldsymbol{x})} = 1)$.

Lemma A.2. *For any tree T and distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, the following holds for $v \in V_T$:*

$$\begin{aligned}\mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x})} &\left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \, |\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|\right] = \\ &= \mathbf{P}(z_{\mathrm{pa}(v)} = 1) \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x} | z_{\mathrm{pa}(v)} = 1)} \left[|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|\right] ,\end{aligned}$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$.

Proof. By using the definition of expectation, replacing $\eta_{\mathrm{pa}(v)}(\boldsymbol{x})$ by its definition, applying Bayes' theorem, and rearranging terms, we obtain:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x})\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\right] = \\
&\quad = \int \mathbf{P}(\boldsymbol{x})\eta_{\mathrm{pa}(v)}(\boldsymbol{x})\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\,d\boldsymbol{x} \\
&\quad = \int \mathbf{P}(\boldsymbol{x})\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\,d\boldsymbol{x} \\
&\quad = \mathbf{P}(z_{\mathrm{pa}(v)}=1)\int \mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\,d\boldsymbol{x}\,.
\end{aligned}$$

Since

$$\int \mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\,d\boldsymbol{x}$$

is nothing else than the expected L_1 estimation error in node v, denoted by

$$\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\right]\,,$$

we obtain the final result. □

Based on this result, we prove Lemma 5.3.

Lemma 5.3. *For any tree T and distribution $\mathbf{P}(\boldsymbol{x},\boldsymbol{y})$ the following holds for $j\in\mathcal{L}$:*

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x})-\hat{\eta}_j(\boldsymbol{x})|\right] \\
&\quad\le \sum_{v\in\mathrm{Path}(l_j)} \mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\right]\,,
\end{aligned}$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)}=1)=1$.

Proof. Take expectation of both hand sides of (5.2) and use linearity of expectation for the right hand side:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x})-\hat{\eta}_j(\boldsymbol{x})|\right] \\
&\quad\le \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\sum_{v\in\mathrm{Path}(l_j)} \eta_{\mathrm{pa}(v)}(\boldsymbol{x})\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\right] \\
&\quad= \sum_{v\in\mathrm{Path}(l_j)} \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x})\,|\eta(\boldsymbol{x},v)-\hat{\eta}(\boldsymbol{x},v)|\right]
\end{aligned}$$

The rest follows from Lemma A.2. □

Using the above results, we finally prove Theorem 5.4. It bounds the expectation of the L_1-estimation error averaged over all labels by the expected L_1-estimation errors of node classifiers. The expectation is defined over the entire distribution $\Pr(\boldsymbol{x})$. We present the result in a general form of a weighted average as such form is used later in the proofs for the generalized performance metrics.

Theorem 5.4. *For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, and weights $W_j \in R$, $j \in \{1, \ldots, m\}$, the following holds:*

$$\frac{1}{m}\sum_{j=1}^{m} W_j \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \leq \frac{1}{m}\sum_{v\in V} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')|\right] \sum_{j\in L_v} W_j \,, \quad (5.3)$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$. For $W_j = 1$, $j \in \{1, \ldots, m\}$, we have:

$$\frac{1}{m}\sum_{j=1}^{m} \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \leq \frac{1}{m}\sum_{v\in V} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v') - \hat{\eta}(\boldsymbol{x}, v')|\right] |L_v| \,.$$

Proof. From Lemma 5.3 we obtain:

$$\frac{1}{m}\sum_{j=1}^{m} W_j \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \leq \frac{1}{m}\sum_{j=1}^{m} W_j \sum_{v\in\mathrm{Path}(l_j)} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|\right] .$$

The RHS can be further transformed to:

$$\frac{1}{m}\sum_{j=1}^{m} \sum_{v\in\mathrm{Path}(l_j)} W_j \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|\right] = \frac{1}{m}\sum_{v\in V} \mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)|\right] \sum_{j\in L_v} W_j \,.$$

where the last equation follows from the fact that each $v \in V_T$ appears in the double sum $|L_v|$ times (where $|L_v|$ is the number of leaves in a subtree rooted in v; in other words, this is the number of paths from leaves to the root that contain node v). Changing the double sum to sum over all nodes and multiplying the expected L_1 estimation error for v by $\sum_{j\in L_v} W_j$ gives the final result. □

A.3 Proofs of the results from Chapter 5.2

We present the proof of Theorem 5.6. It expresses the bound from Theorem 5.4 in terms of node regrets of a strongly proper composite loss function.

Theorem 5.6. *For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, weights $W_j \in R$, $j \in \{1, \ldots, m\}$, a strongly proper composite loss function ℓ_c, and a set of scoring functions f_v, $v \in V_T$,* *the following holds:*

$$\frac{1}{m}\sum_{j=1}^{m} W_j \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \le \frac{\sqrt{2}}{m\sqrt{\lambda}}\sum_{v\in V}\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v)}\sum_{j\in L_v} W_j\,, \tag{5.5}$$

where for the root node $\mathbf{P}(z_{\mathrm{pa}(r_T)}=1)=1$, *and* $\mathrm{reg}_{\ell_c}(f_v)$ *is the expected* ℓ_c*-regret of* f_v *taken over* $\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)}=1)$. *For* $W_j = 1$, $j\in\{1,\ldots,m\}$, *we have:*

$$\frac{1}{m}\sum_{j=1}^{m}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\right] \le \frac{\sqrt{2}}{m\sqrt{\lambda}}\sum_{v\in V}|L_v|\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v)}\,. \tag{5.6}$$

Proof. As ψ is an invertible function satisfying $f(\boldsymbol{x}) = \psi(\mathbf{P}(y=1\,|\,\boldsymbol{x}))$, we can assume that $\hat{\eta}(\boldsymbol{x}, v) = \psi^{-1}(f_v(\boldsymbol{x}))$. We then obtain from (2.3):

$$\eta_{\mathrm{pa}(v)}(\boldsymbol{x})\,|\eta(\boldsymbol{x},v) - \hat{\eta}(\boldsymbol{x},v)| \le \eta_{\mathrm{pa}(v)}(\boldsymbol{x})\sqrt{\frac{2}{\lambda}}\sqrt{\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\,, \tag{A.4}$$

for any $v\in V_T$. We take the expectation with respect to $\mathbf{P}(\boldsymbol{x})$ of (A.4). Based on Lemma A.2, given in Appendix A.2, the left hand side is equal to:

$$\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x},v) - \hat{\eta}(\boldsymbol{x},v)|\right]\,.$$

For the left hand side we obtain the following upper bound:

$$\begin{aligned}
&\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x})\sqrt{\frac{2}{\lambda}}\sqrt{\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\right] = \\
&= \sqrt{\frac{2}{\lambda}}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\sqrt{\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\right] \\
&= \sqrt{\frac{2}{\lambda}}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})^2\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\right] \\
&\le \sqrt{\frac{2}{\lambda}}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\right]\,.
\end{aligned}$$

Using Jensen's inequality we further get:

$$\begin{aligned}
&\sqrt{\frac{2}{\lambda}}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})}\right] \le \\
&\le \sqrt{\frac{2}{\lambda}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})\right]}
\end{aligned}$$

The next step is similarly to the proof of Lemma 5.1. We use first the definition of expectation, then Bayes' theorem, and finally we rearrange the terms:

$$\begin{aligned}
&\sqrt{\frac{2}{\lambda}\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}\left[\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})\right]} = \\
&= \sqrt{\frac{2}{\lambda}\int \mathbf{P}(\boldsymbol{x})\mathbf{P}(z_{\mathrm{pa}(v)}=1|\boldsymbol{x})\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})d\boldsymbol{x}} \\
&= \sqrt{\frac{2}{\lambda}\int \mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})d\boldsymbol{x}} \\
&= \sqrt{\frac{2}{\lambda}\mathbf{P}(z_{\mathrm{pa}(v)}=1)\int \mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})d\boldsymbol{x}}\,.
\end{aligned}$$

Notice that:

$$\int \mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell}(f_v\,|\,\boldsymbol{x})d\boldsymbol{x} = \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)=1})}\left[\mathrm{reg}_{\ell_c}(f_v\,|\,\boldsymbol{x})\right]$$

This is the expected regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v\,|\,z_{\mathrm{pa}(v)}=1)$, denoted by $\mathrm{reg}_{\ell_c}(f_v)$. We thus obtain the following by taking the expectation of (A.4):

$$\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}|z_{\mathrm{pa}(v)}=1)}\left[|\eta(\boldsymbol{x},v')-\hat{\eta}(\boldsymbol{x},v')|\right] \le \sqrt{\frac{2}{\lambda}}\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)}=1)\mathrm{reg}_{\ell_c}(f_v)}$$

By using the above in (5.3) from Theorem 5.4, we obtain the final result. □

A.4 Proofs of the results from Section 5.3

This appendix contains proofs of Theorems 5.7 and 5.8 which state the regret bounds of PLTs for generalized performance metrics. We start with presenting two other results being the building blocks of the main proofs. Both are based on [Kotłowski and Dembczyński, 2017]. The first one states that the regret for a cost-sensitive binary classification can be upper-bounded by L_1 estimation error of the conditional probabilities by using a proper threshold that corresponds to the misclassification cost. The second one shows that the regret of the generic function $\Psi(\mathrm{FP},\mathrm{FN})$ can be upper-bounded by the regret of the cost-sensitive binary classification with costs being a function of the optimal value of $\Psi(\mathrm{FP},\mathrm{FN})$.

Given a real number $\alpha \in [0,1]$, let us first define an α-cost-sensitive loss function for a single binary label y, $\ell_\alpha : \{0,1\}\times\{0,1\}\to[0,2]$, as:

$$\ell_\alpha(y,\hat{y}) = 2\alpha[\![y=0]\!][\![\hat{y}=1]\!] + 2(1-\alpha)[\![y=1]\!][\![\hat{y}=0]\!]$$

The cost-sensitive loss assigns different costs of misclassification depending on whether the label is relevant ($y=1$) or not ($y=-1$). The multiplier of 2 makes $\ell_{0.5}(y,\hat{y})$ to be the typical binary 0/1 loss. Given classifier h, the α-cost-sensitive risk of h is:

$$R_\alpha(h) = \mathbb{E}_{(y,\boldsymbol{x})\sim\mathbf{P}(y,\boldsymbol{x})}[\ell_\alpha(y,\hat{y})] = 2\alpha\mathrm{FP}(h) + 2(1-\alpha)\mathrm{FN}(h) \tag{A.5}$$

The α-cost-sensitive regret of h is then:

$$\mathrm{reg}_\alpha(h) = R_\alpha(h) - R_\alpha(h^*_\alpha)\,, \tag{A.6}$$

where $h^*_\alpha = \arg\min_h R_\alpha(h)$.

Proposition A.3. *For any distribution* $\mathbf{P}$ *over* $(y,\boldsymbol{x}) \in \{0,1\}\times\mathcal{X}$*, with* $\eta(\boldsymbol{x}) = \mathbf{P}(y = 1\,|\,\boldsymbol{x})$*, any* $\alpha \in [0,1]$*, and classifier* h*, such that* $h(\boldsymbol{x}) = [\![\hat{\eta}(\boldsymbol{x}) > \alpha]\!]$ *with* $\hat{\eta}(\boldsymbol{x}) \in [0,1]$*, the following holds:*

$$\mathrm{reg}_\alpha(h) \leq 2\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x}))}[|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|]$$

Proof. The proof is a part of the derivation of the bound from Proposition 2 in Kotłowski and Dembczyński [2017]. Given $\eta \in [0,1]$ and $h \in [0,1]$, the conditional α-cost-sensitive risk is:

$$R_\alpha(h\,|\,\boldsymbol{x}) = \mathbb{E}_{y\sim\mathbf{P}(y\,|\,\boldsymbol{x})}\left[\ell_\alpha(y,h)\right] = 2\alpha(1-\eta)[\![h=1]\!] + 2(1-\alpha)\eta[\![h=0]\!]\,.$$

Let $h^*_\alpha \in \arg\min_h \mathrm{risk}_\alpha(\eta,h)$. It is easy to check that one of possible solutions is

$$h^*_\alpha = [\![\eta > \alpha]\!]\,. \tag{A.7}$$

The α-conditional cost-sensitive regret is

$$\mathrm{reg}_\alpha(h\,|\,\boldsymbol{x}) = R_\alpha(h\,|\,\boldsymbol{x}) - R_\alpha(h^*_\alpha\,|\,\boldsymbol{x}).$$

If $h = h^*_\alpha$, then $\mathrm{reg}_\alpha(\eta,h) = 0$, otherwise, $\mathrm{reg}_\alpha(\eta,h) = 2|\eta-\alpha|$, so

$$\mathrm{reg}_\alpha(h\,|\,\boldsymbol{x}) = 2[\![h \neq h^*_\alpha]\!]|\eta-\alpha|.$$

In the statement of the theorem, we assume $h(\boldsymbol{x}) = [\![\hat{\eta}(\boldsymbol{x}) > \alpha]\!]$, for some $\hat{\eta}(\boldsymbol{x}) \in [0,1]$, that is, $h(\boldsymbol{x})$ has the same form as $h^*_\alpha(\boldsymbol{x})$ in (A.7). For such $h(\boldsymbol{x})$ we have:

$$\mathrm{reg}_\alpha(h\,|\,\boldsymbol{x}) \leq 2|\eta - \hat{\eta}|\,.$$

This statement trivially holds when $h = h^*_\alpha$. If $h \neq h^*_\alpha$, then η and $\hat{\eta}$ are on the opposite sides of α, hence $|\eta-\alpha| \leq |\eta-\hat{\eta}|$.

The unconditional statement is obtained by taking the expectation with respect to $\boldsymbol{x}$ of both sides of the above equation:

$$\mathrm{reg}_\alpha(h) = \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[\mathrm{reg}_\alpha(h\,|\,\boldsymbol{x})] \leq 2\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|]\,.$$

□

The second result is a modified version of Proposition 1 from [Kotłowski and Dembczyński, 2017], which in turn generalizes Proposition 6 in [Puthiya Parambath et al., 2014].

Proposition A.4. *Let* Ψ *be a linear-factorial function as defined in* (3.1) *with the denominator bounded away from 0 by* γ *as in* (3.2). *Take any real values* FP, FN *and* FP*, FN* *in the domain of* Ψ *such that:*

$$\Psi(\mathrm{FP}^*, \mathrm{FN}^*) - \Psi(\mathrm{FP}, \mathrm{FN}) \geq 0\,.$$

Then, we obtain:

$$\Psi(\mathrm{FP}^*, \mathrm{FN}^*) - \Psi(\mathrm{FP}, \mathrm{FN}) \leq C(\alpha^*_\Psi(\mathrm{FP} - \mathrm{FP}^*) + (1 - \alpha^*_\Psi)(\mathrm{FN} - \mathrm{FN}^*))$$

where:

$$\alpha^*_\Psi = \frac{\Psi(\mathrm{FP}^*, \mathrm{FN}^*)b_1 - a_1}{\Psi(\mathrm{FP}^*, \mathrm{FN}^*)(b_1 + b_2) - (a_1 + a_2)}\,,$$

and

$$C = \frac{1}{\gamma}\left(\Psi(\mathrm{FP}^*, \mathrm{FN}^*)\,(b_1 + b_2) - (a_1 + a_2)\right) > 0\,.$$

Proof. For the sake of clarity, we use a shorthand notation $\Psi^* = \Psi(\mathrm{FP}^*, \mathrm{FN}^*)$, $\Psi = \Psi(\mathrm{FP}, \mathrm{FN})$, $A = a_0 + a_1\mathrm{FP} + a_2\mathrm{FN}$, $B = b_0 + b_1\mathrm{FP} + b_2\mathrm{FN}$, for the numerator and denominator of Ψ, and analogously A^* and B^* for Ψ^*. With this notation, we have:

$$\begin{aligned}
\Psi^* - \Psi = \frac{\Psi^* B - A}{B} &= \frac{\Psi^* B - A - \overbrace{(\Psi^* B^* - A^*)}^{=0}}{B} \\
&= \frac{\Psi^*(B - B^*) - (A - A^*)}{B} \\
&= \frac{(\Psi^* b_1 - a_1)(\mathrm{FP} - \mathrm{FP}^*) + (\Psi^* b_2 - a_2)(\mathrm{FN} - \mathrm{FN}^*)}{B} \\
&\leq \frac{(\Psi^* b_1 - a_1)(\mathrm{FP} - \mathrm{FP}^*) + (\Psi^* b_2 - a_2)(\mathrm{FN} - \mathrm{FN}^*)}{\gamma}\,, \qquad \text{(A.8)}
\end{aligned}$$

where the last inequality follows from the assumptions that $B \geq \gamma$ and $\Psi^* - \Psi \geq 0$. Since Ψ is non-increasing in FP and FN, we have:

$$\frac{\partial \Psi^*}{\partial \mathrm{FP}^*} = \frac{a_1 B^* - b_1 A^*}{(B^*)^2} = \frac{a_1 - b_1 \Psi^*}{B^*} \leq 0$$

and similarly $\frac{\partial \Psi^*}{\partial \mathrm{FN}^*} = \frac{a_2 - b_2\Psi^*}{B^*} \leq 0$. This and the assumption $B^* \geq \gamma$ implies that both $\Psi b_1 - a_1$ and $\Psi^* b_2 - a_2$ are non-negative. If we normalize them by defining:

$$\alpha^*_\Psi = \frac{\Psi^* b_1 - a_1}{\Psi^*(b_1 + b_2) - (a_1 + a_2)}\,,$$

we obtain then from (A.8):

$$\Psi^* - \Psi \leq C(\alpha^*_\Psi(\mathrm{FP} - \mathrm{FP}^*) + (1 - \alpha^*_\Psi)(\mathrm{FN} - \mathrm{FN}^*))$$

with C being $\frac{1}{\gamma}\left(\Psi^*\,(b_1 + b_2) - (a_1 + a_2)\right)$. □

With the above results we can prove the main theorems of Section 5.3.

Theorem 5.7. *Let $\tau_j^* = \arg\max_\tau \Psi(h_{j,\tau})$, for each $j \in \mathcal{L}$, and $\boldsymbol{\tau}^* = (\tau_1^*, \tau_2^*, \ldots, \tau_m^*)$. For any tree T and distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, the classifier $\boldsymbol{h}_{\boldsymbol{\tau}^*}$ achieves the following upper bound on its Ψ_{macro}-regret:*

$$\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\tau}^*}) \le \frac{\sqrt{2}}{m\sqrt{\lambda}} \sum_{v \in V} \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)} \sum_{j \in L_v} C_j\,,$$

where $C_j = \frac{1}{\gamma}(\Psi(h_\Psi^*, j)(b_1 + b_2) - (a_1 + a_2))$, for each $j \in \mathcal{L}$, with γ defined in (3.2), $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ *for the root node, and* $\mathrm{reg}_{\ell_c}(f_v)$ *is the expected ℓ_c-regret of f_v taken over $\mathbf{P}(\boldsymbol{x}, z_v \mid z_{\mathrm{pa}(v)} = 1)$.*

Proof. From the definitions of the macro-average performance measure (3.3) and the regret of Ψ_{macro} (5.7), as well as from Proposition A.4 we have for any $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_m(\boldsymbol{x}))$ that:

$$\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}) \le \frac{1}{m} \sum_{j=1}^{m} C_j(\alpha_\Psi^*(\mathrm{FP}_j - \mathrm{FP}_j^*) + (1 - \alpha_\Psi^*)(\mathrm{FN}_j - \mathrm{FN}_j^*))\,,$$

with FP_j and FN_j being the false positives and false negatives of h_j. It can be easily notice $(\alpha_\Psi^*(\mathrm{FP}_j - \mathrm{FP}_j^*) + (1 - \alpha_\Psi^*)(\mathrm{FN}_j - \mathrm{FN}_j^*))$ is half of the α_Ψ^*-regret (A.6) for label j. Therefore, we can write:

$$\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}) \le \frac{1}{2m} \sum_{j=1}^{m} C_j \mathrm{reg}_{\alpha_\Psi^*}(h_j)\,.$$

If we now take $h_j = h_{j,\alpha_{\Psi,j}^*}$, then by using Proposition A.3 and the bound (5.5) from Theorem 5.6 we obtain:

$$\begin{aligned}\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\alpha}_\Psi^*}) &\le \frac{1}{m} \sum_{j=1}^{m} C_j \mathbb{E}_{\boldsymbol{x} \sim \mathbf{P}(\boldsymbol{x}))}[|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|] \\ &\le \frac{\sqrt{2}}{m\sqrt{\lambda}} \sum_{v \in V} \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)} \sum_{j \in L_v} C_j\,.\end{aligned}$$

Finally, since

$$\boldsymbol{\tau}^* = \arg\max_{\boldsymbol{\tau}} \Psi_{\mathrm{macro}}(\boldsymbol{h}_{\boldsymbol{\tau}}) = \arg\min_{\boldsymbol{\tau}} \mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\tau}})\,,$$

we have that $\mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\tau}^*}) \le \mathrm{reg}_{\Psi_{\mathrm{macro}}}(\boldsymbol{h}_{\boldsymbol{\alpha}_\Psi^*})$. □

Theorem 5.8. *Let $\boldsymbol{h}_\tau = (h_{1,\tau}, h_{2,\tau}, \ldots, h_{m,\tau})$ be a classifier which shares the same threshold τ over all labels $j \in \mathcal{L}$. For any tree T, distribution $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$, and $\tau^* = \arg\max_\tau \Psi_{\mathrm{micro}}(\boldsymbol{h}_\tau)$, classifier $\boldsymbol{h}_{\tau^*}$ achieves the following upper bound on its Ψ_{micro}-regret:*

$$\mathrm{reg}_{\Psi_{micro}}(\boldsymbol{h}_{\tau^*}) \le \frac{C}{m}\sqrt{\frac{2}{\lambda}} \sum_{v \in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,,$$

where $C = \frac{1}{\gamma}(\Psi_{\mathrm{micro}}(\boldsymbol{h}_\Psi^*)(b_1 + b_2) - (a_1 + a_2))$ with γ defined in (3.2), $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ *for the root node, and* $\mathrm{reg}_{\ell_c}(f_v)$ *is the expected ℓ_c-regret of f_v taken over*

$\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)} = 1)$.

Proof. Using Proposition A.4, which applies to any real values FP, FN, FP*, FN*, and from definitions of the micro-averaged performance measure (3.4) and the regret of Ψ_{micro} (5.8), we can write for any $\boldsymbol{h}(\boldsymbol{x}) = (h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \ldots, h_m(\boldsymbol{x}))$ that:

$$\begin{aligned}\mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}) &= \Psi(\bar{\mathrm{FP}}(\boldsymbol{h}^*_{\alpha^*_\Psi}), \bar{\mathrm{FN}}(\boldsymbol{h}^*_{\alpha^*_\Psi})) - \Psi(\bar{\mathrm{FP}}(\boldsymbol{h}), \bar{\mathrm{FN}}(\boldsymbol{h})) \\ &\le C\big(\alpha^*_\Psi(\bar{\mathrm{FP}}(\boldsymbol{h}) - \bar{\mathrm{FP}}(\boldsymbol{h}^*_{\alpha^*_\Psi})) + (1-\alpha^*_\Psi)(\bar{\mathrm{FN}}(\boldsymbol{h}) - \bar{\mathrm{FN}}(\boldsymbol{h}^*_{\alpha^*_\Psi}))\big) \\ &= \frac{C}{m}\sum_{j=1}^m \alpha^*_\Psi(\mathrm{FP}_j(h_j) - \mathrm{FP}_j(\boldsymbol{h}^*_{\alpha^*_\Psi})) + (1-\alpha^*_\Psi)(\mathrm{FN}_j(h_j) - \bar{\mathrm{FN}}_j(\boldsymbol{h}^*_{\alpha^*_\Psi}))\,.\end{aligned}$$

Further from α-cost-sensitive risk (A.5) and regret (A.6), we have:

$$\begin{aligned}\mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}) &\le \frac{C}{2m}\sum_{j=1}^m \Big(R_\alpha(h_j) - R_\alpha(\boldsymbol{h}^*_{\alpha^*_\Psi})\Big) \\ &= \frac{C}{2m}\sum_{j=1}^m \mathrm{reg}_\alpha(h_j)\,.\end{aligned}$$

If we now take $h_j = h_{j,\alpha^*_\Psi}$, for all $j \in \mathcal{L}$, then by using Proposition A.3 we obtain:

$$\mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}_{\alpha^*_\Psi}) \le \frac{C}{m}\sum_{j=1}^m \mathbb{E}_{\boldsymbol{x}}[|\eta(\boldsymbol{x}) - \hat{\eta}(\boldsymbol{x})|].$$

By using the bound (5.6) from Theorem 5.6 we have:

$$\mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}) \le \frac{C}{m}\sqrt{\frac{2}{\lambda}}\sum_{v\in V}|L_v|\sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,.$$

The theorem now follows from noticing that

$$\tau^* = \arg\max_\tau \Psi_{\text{micro}}(\boldsymbol{h}_\tau) = \arg\min_\tau \mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}_\tau)\,,$$

we have that $\mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}_{\tau^*}) \le \mathrm{reg}_{\Psi_{\text{micro}}}(\boldsymbol{h}_{\alpha^*_\Psi})$. □

A.5 Proofs of the results from Section 5.5

Theorem 5.11. *For any distribution $\mathbf{P}(\boldsymbol{y}\,|\,\boldsymbol{x})$ and classifier $\boldsymbol{h} \in \mathcal{H}^m$ the following holds:*

$$\begin{aligned}\mathrm{reg}_{D@k}(\boldsymbol{h}_{@k}\,|\,\boldsymbol{x}) &= \sum_{i\in\mathcal{L}^*_k}\eta_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}^*_{D@k})) - \sum_{j\in\hat{\mathcal{L}}_k}\eta_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}_{@k})) \\ &\le 2\max_j|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\, IDCG@k(k)\,,\end{aligned}$$

where IDCG@k(k) $= \sum_{r=1}^{k} g(r)$*,* $\mathcal{L}_k^*$ *is the set of labels on first* k *ranks in ranking* $\pi(\boldsymbol{h}_{D@k}^*)$*, and* $\hat{\mathcal{L}}_k$ *is the set of labels on first* k *ranks in ranking* $\boldsymbol{\pi}(\boldsymbol{h})$*.*

Proof. Let us add and subtract the following two terms, $\sum_{i\in\mathcal{L}_k^*} \hat{\eta}_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*))$ and $\sum_{j\in\hat{\mathcal{L}}_k} \hat{\eta}_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}_{@k}))$, to the regret and reorganize the expression in the following way:

$$
\begin{aligned}
\mathrm{reg}_{D@k}(\boldsymbol{h} \,|\, \boldsymbol{x}) = & \underbrace{\sum_{i\in\mathcal{L}_k^*} \eta_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*)) - \sum_{i\in\mathcal{L}_k^*} \hat{\eta}_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*))}_{\le \sum_{i\in\mathcal{L}_k^*}|\eta_i(\boldsymbol{x})-\hat{\eta}_i(\boldsymbol{x})|g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*))} \\
& + \underbrace{\sum_{j\in\hat{\mathcal{L}}_k} \hat{\eta}_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h})) - \frac{1}{k}\sum_{j\in\hat{\mathcal{L}}_k} \eta_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}))}_{\le \frac{1}{k}\sum_{j\in\hat{\mathcal{L}}_k}|\hat{\eta}_j(\boldsymbol{x})-\eta_j(\boldsymbol{x})|g(\pi_j^{-1}(\boldsymbol{h}))} \\
& + \underbrace{\sum_{i\in\mathcal{L}_k^*} \hat{\eta}_i(\boldsymbol{x})g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*)) - \frac{1}{k}\sum_{j\in\hat{\mathcal{L}}_k} \hat{\eta}_j(\boldsymbol{x})g(\pi_j^{-1}(\boldsymbol{h}))}_{\le 0} \\
& \le \sum_{i\in\mathcal{L}_k^*} |\eta_i(\boldsymbol{x}) - \hat{\eta}_i(\boldsymbol{x})|\, g(\pi_i^{-1}(\boldsymbol{h}_{D@k}^*)) + \sum_{j\in\hat{\mathcal{L}}_k} |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|\, g(\pi_j^{-1}(\boldsymbol{h}))
\end{aligned}
$$

Next we bound each L_1 error, $|\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|$, by $\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|$. Notice that $|\hat{\mathcal{L}}_k| = k$ and each label in $\hat{\mathcal{L}}_k$ appears at only one, unique, rank in $\boldsymbol{\pi}(\boldsymbol{h})$. Therefore $\sum_{j\in\hat{\mathcal{L}}_k} g(\pi_j^{-1}(\boldsymbol{h})) = \sum_{r=1}^{k} g(r)$. The same reasoning applies to $\mathcal{L}_k^*$ and $\boldsymbol{\pi}(\boldsymbol{h}_{D@k}^*)$. Therefore

$$
\mathrm{reg}_{D@k}(\boldsymbol{h} \,|\, \boldsymbol{x}) \le 2 \max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \sum_{r=1}^{k} g(r)\,.
$$

□

Theorem 5.12. *For any tree* T *and distribution* $\mathbf{P}(\boldsymbol{x}, \boldsymbol{y})$*, classifier* $\boldsymbol{h}(\boldsymbol{x})$ *achieves the following upper bound on its DCG@k regret:*

$$
\mathrm{reg}_{D@k}(\boldsymbol{h}) \le IDCG@k(k)\frac{2\sqrt{2}}{\sqrt{\lambda}} \sum_{v\in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)}\,.
$$

where IDCG@k(k) $= \sum_{r=1}^{k} g(r)$*,* $\mathbf{P}(z_{\mathrm{pa}(r_T)} = 1) = 1$ *for the root node, and* $\mathrm{reg}_{\ell_c}(f_v)$ *is the expected* ℓ_c*-regret of* f_v *taken over* $\mathbf{P}(\boldsymbol{x}, z_v \,|\, z_{\mathrm{pa}(v)} = 1)$*.*

Proof. By taking expectation over $\mathbf{P}(\boldsymbol{x})$ of the bound from Theorem 5.11 and replacing the max operator by sum, that is, $\max(a, b) \le a + b$, for $a, b \ge 0$, we obtain:

$$\begin{aligned}
\mathrm{reg}_{D@k}(\boldsymbol{h}) &= \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[\mathrm{reg}_{D@k}(\boldsymbol{h}_{@k} \mid \boldsymbol{x})] \\
&\leq \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[2\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})| \sum_{r=1}^{k} g(r)] \\
&\leq 2\sum_{r=1}^{k} g(r)\mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})}[\max_j |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|] \\
&\leq 2IDCG@k(k)\sum_{j=1}^{m} \mathbb{E}_{\boldsymbol{x}\sim\mathbf{P}(\boldsymbol{x})} |\eta_j(\boldsymbol{x}) - \hat{\eta}_j(\boldsymbol{x})|] \,.
\end{aligned}$$

Next, by applying (5.6) from Theorem 5.6, we get the statement:

$$\mathrm{reg}_{D@k}(\boldsymbol{x})) \leq IDCG@k(k)\frac{2\sqrt{2}}{\sqrt{\lambda}} \sum_{v\in V} |L_v| \sqrt{\mathbf{P}(z_{\mathrm{pa}(v)} = 1)\mathrm{reg}_{\ell_c}(f_v)} \,.$$

□

A.6 Proofs of the results from Chapter 6

Proposition 6.1. *For any tree T and vector $\boldsymbol{y}$ it holds that:*

$$c(T, \boldsymbol{y}) \leq 1 + \|\boldsymbol{y}\|_1 \cdot \mathrm{depth}_T \cdot \deg_T \,,$$

where $\mathrm{depth}_T = \max_{v\in L_T} \mathrm{len}_v - 1$ *is the depth of the tree, and* $\deg_T = \max_{v\in V_T} \deg_v$ *is the highest degree of a node in T.*

Proof. First notice that a training observation is always used in the root node, either as a positive observation $(\boldsymbol{x}, 1)$, if $\|\boldsymbol{y}\|_1 > 0$, or as a negative observation $(\boldsymbol{x}, 0)$, if $\|\boldsymbol{y}\|_1 = 0$. Therefore the cost is bounded by 1. If $\|\boldsymbol{y}\|_1 > 0$, the training observation is also used as a positive observation in all the nodes on paths from the root to leaves corresponding to labels j for which $y_j = 1$ in $\boldsymbol{y}$. As the root has been already counted, we have at most $\mathrm{depth}_T = \max_v \mathrm{len}_v - 1$ such nodes for each positive label in $\boldsymbol{y}$. Moreover, the training observation is used as a negative observation in all siblings of the nodes on the paths determined above, unless it is already a positive observation in the sibling node. The highest degree of a node in the tree is $\deg_T$. Taking the above into account, the cost $c(T, \boldsymbol{y})$ is bounded from above by $1 + \|\boldsymbol{y}\|_1 \cdot \mathrm{depth}_T \cdot \deg_T$. The bound is tight, for example, if $\|\boldsymbol{y}\|_1 = 1$ and T is a perfect $\deg_T$-ary tree (all non-leaf nodes have an equal degree and the paths to the root from all leaves are of the same length). □

Theorem 6.4. *For Algorithm 3 with all thresholds τ_v, $v \in V_T$, set to τ and any $\boldsymbol{x} \in \mathcal{X}$, we have that:*

$$c_\tau(T, \boldsymbol{x}) \leq 1 + \lfloor \hat{P}/\tau \rfloor \cdot \mathrm{depth}_T \cdot \deg_T \,, \tag{6.2}$$

where $\hat{P}$ is a constant upper-bounding $\sum_{j=1}^{m} \hat{\eta}_j(\boldsymbol{x})$, $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, and $\deg_T = \max_{v \in V_T} \deg_v$.

Proof. The proof is similar to the one of Theorem 6.1 in [Busa-Fekete et al., 2019]. As stated before the theorem, we assume that the estimates are properly normalized, that is, they satisfy:

$$\hat{\eta}_v(\boldsymbol{x}) \le \min\left\{1, \sum_{v' \in \mathrm{Ch}(v)} \hat{\eta}_{v'}(\boldsymbol{x})\right\}, \tag{A.9}$$

and

$$\max\left\{\hat{\eta}_{v'}(\boldsymbol{x}), v' \in \mathrm{Ch}(v)\right\} \le \hat{\eta}_v(\boldsymbol{x}). \tag{A.10}$$

Consider the subtree T' of T, which consists of all nodes $v \in V_T$ for which $\hat{\eta}_v(\boldsymbol{x}) \ge \tau$. If there are no such nodes, from the pseudocode of Algorithm 3, we see that only the root classifier is called. The upper-bound (6.2) in this case obviously holds. However, it might not be tight as $\hat{\eta}_v(\boldsymbol{x}) < \tau$ does not imply $\hat{P} \le \tau$ because of (A.9).

If T' has at least one node, Algorithm 3 visits each node of T' (calls a corresponding classifier and adds the node to a stack), since for each parent node we have (A.10). Moreover, Algorithm 3 visits all children of nodes T' (some of them are already in T'). Let the subtree T'' consist of all nodes of T' and their child nodes. Certainly $T' \subseteq T'' \subseteq T$. To prove the theorem we count first the number of nodes in T' and then the number of nodes in T'', which gives as the final result.

If the number of nodes in T' is greater than or equal to 1, then certainly r_T is in T'. Let us consider next the number of leaves of T'. Observe that $\sum_{v \in L_{T'}} \hat{\eta}_v(\boldsymbol{x}) \le \hat{P}$. This is because $\sum_{v \in L_{T'}} \hat{\eta}_v(\boldsymbol{x}) \le \sum_{v \in L_T} \hat{\eta}_v(\boldsymbol{x}) \le \hat{P}$, that is, $v \in L_{T'}$ might be an internal node in T and its $\hat{\eta}_v(\boldsymbol{x})$ is at most the sum of probability estimates of the leaves underneath v according to (A.9). From this we get the following upper bound on the number of leaves in T':

$$|L_{T'}| \le \lfloor \hat{P}/\tau \rfloor. \tag{A.11}$$

Since the degree of internal nodes in T' might be 1, to upper-bound the number of all nodes in T' we count the number of nodes on all paths from leaves to the root, but counting the root node only once:

$$|V_{T'}| \le 1 + \sum_{v \in L_{T'}} (\mathrm{len}_v - 1).$$

Next, notice that for each $v \in T'$ its all siblings are in T'' unless v is the root node. This is because if a non-root node v is in T' then its parent is also in T' according to (A.10) and T'' contains all child nodes of nodes in T'. The rest of nodes in T'' are the child nodes of leaves of T', unless a leaf of T' is also a leaf of T. Therefore, we have

$$|V_{T''}| \le 1 + \sum_{v \in L_{T'}} \deg_T(\mathrm{len}_v - 1) + \sum_{v \in L_{T'}} \deg_T [\![v \notin L_T]\!],$$

with $\deg_T$ being the highest possible degree of a node. Since (A.11) and

$$\mathrm{len}_v - 1 + [\![v \notin L_T]\!] \leq \mathrm{depth}_T \,,$$

that is, the longest path cannot be longer than the depth of the tree plus 1, we finally get:

$$|V_{T''}| \leq 1 + \lfloor \hat{P}/\tau \rfloor \cdot \mathrm{depth}_T \cdot \deg_T \,.$$

This ends the proof as the number of nodes in T'' is equivalent to the number of calls to the node classifiers, that is, $c_\tau(T, \boldsymbol{x})$. □

Theorem 6.6. *Using the notation above, it holds that*

$$C_{\mathbf{P}(\boldsymbol{x}),\tau}(T) \leq \frac{1}{\tau}\Big(C_{\mathbf{P}}(T) + \sum_{v \in V_T} \mathbb{E}_{\boldsymbol{x}} \left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \cdot |\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)| \right] \cdot |V_v \backslash L_v| \cdot \deg_v \Big) - \frac{1-\tau}{\tau} .$$

Proof. We can upper bound the expected inference cost as follows:

$$\mathbb{E}_{\boldsymbol{x}}\left[c_\tau(T, \boldsymbol{x}) \right] = \mathbb{E}_{\boldsymbol{x}} \left[1 + \sum_{v \in V_T} \mathbb{I}\left\{ \hat{\eta}_v(\boldsymbol{x}) \geq \tau \right\} \deg_v \right] \tag{A.12}$$

$$\leq \mathbb{E}_{\boldsymbol{x}} \left[1 + \sum_{v \in V_T} \frac{\hat{\eta}_v(\boldsymbol{x})}{\tau} \deg_v \right]$$

$$\leq 1 + \sum_{v \in V_T} \frac{\mathbb{E}_{\boldsymbol{x}}\left[\eta_v(\boldsymbol{x}) + |\hat{\eta}_v(\boldsymbol{x}) - \eta_v(\boldsymbol{x})| \right]}{\tau} \deg_v$$

$$\leq \frac{1}{\tau} \left(1 + \sum_{v \in V_T} \mathbb{E}_{\boldsymbol{x}} \left[\eta_v(\boldsymbol{x}) + \sum_{v' \in \mathrm{Path}(v)} \eta_{\mathrm{pa}(v')}(\boldsymbol{x}) \cdot |\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)| \right] \cdot \deg_v \right) - \frac{1-\tau}{\tau} \tag{A.13}$$

$$\leq \frac{1}{\tau}\Big(C_{\mathbf{P}}(T) + \sum_{v \in V_T} \mathbb{E}_{\boldsymbol{x}} \left[\eta_{\mathrm{pa}(v)}(\boldsymbol{x}) \cdot |\eta(\boldsymbol{x}, v) - \hat{\eta}(\boldsymbol{x}, v)| \right] \cdot |T(v)| \cdot \deg_v \Big) - \frac{1-\tau}{\tau}, \tag{A.14}$$

where (A.13) follows from Lemma 5.1. □

Theorem 6.7. *For Algorithm 4 retrieving k top-scoring labels and any $\boldsymbol{x} \in \mathcal{X}$ we have that:*

$$c_k(T, \boldsymbol{x}) \leq 1 + (k + c - 1) \cdot \mathrm{depth}_T \cdot \deg_T,$$

where c, $c \geq 1$, is an integer for which $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x}) > \frac{1}{c} \sum_{i=k+1}^{m} \hat{\eta}_{\pi_i(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$, $\hat{\boldsymbol{\eta}}(\boldsymbol{x})$ denotes the vector of estimates of conditional label probabilities using the normalization (4.9), *$\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, and $\deg_T = \max_{v \in V_T} \deg_v$.*

Proof. The prediction cost $c_k(T, \boldsymbol{x})$ of Algorithm 4 is the number of calls to the node classifiers. Algorithm 4 calls the classifiers in the root and in all the children of internal nodes v such that $\hat{\eta}_v(\boldsymbol{x})$ is greater than or equal to the k-th highest conditional probability estimate $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$. We denote the set of such internal nodes as $\mathcal{A}$,

$$\mathcal{A} = \left\{ v \in V_T \setminus L_T : \hat{\eta}_v(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x}) \right\} .$$

Algorithm 4 visits calls the classifier in the root, and in the children of nodes in $\mathcal{A}$. Therefore, to upper-bound the cost $c_k(T, \boldsymbol{x})$ we upper-bound the number of nodes in $\mathcal{A}$, upper-bound the number of children of nodes from $\mathcal{A}$, and add 1 to the count to account for the root.

If $\hat{\eta}_v(\boldsymbol{x})$ are properly normalized, then $\hat{\eta}_v(\boldsymbol{x})$ is less than or equal to the sum of $\hat{\eta}_j(\boldsymbol{x})$ in leaf nodes L_v, i.e. $\hat{\eta}_v(\boldsymbol{x}) \leq \sum_{j \in L_v} \hat{\eta}_j(\boldsymbol{x})$ for each $v \in V_T$. Estimate $\hat{\eta}_v(\boldsymbol{x})$ is greater than or equal to $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ only if $\sum_{j \in L_v} \hat{\eta}_j(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$. Let $\hat{\mathcal{Y}}_k$ denote the set of k highest scoring labels retrieved by Algorithm 4, $\hat{\mathcal{Y}}_k = \{\pi_r(\hat{\boldsymbol{\eta}}(\boldsymbol{x})) : r = 1, \ldots, k\}$.

Consider an internal node v. Node v may have a predicted label in its subtree labels, $\mathcal{L}_v \cap \hat{\mathcal{Y}}_k \neq \emptyset$, or may not have, $\mathcal{L}_v \cap \hat{\mathcal{Y}}_k = \emptyset$. In the first case, if the node v is a predecessor of a top-k scoring label, $\hat{\eta}_v(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ always holds, as $\hat{\eta}_v(\boldsymbol{x}) \geq \hat{\eta}_j(\boldsymbol{x})$ for each $j \in \mathcal{L}_v$. In the second case, $\sum_{j \in L_v} \hat{\eta}_j(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ is a necessary condition for $\hat{\eta}_v(\boldsymbol{x}) > \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ to hold. Consider internal nodes on a single tree level. There are at most k predecessors of top-k scoring labels on a single tree level. Those nodes are always included in $\mathcal{A}$. Consider the nodes on this tree level that are not predecessors of s top-k scoring label. Let us denote the sum of $\hat{\eta}_v(\boldsymbol{x})$ for these nodes as $\mathcal{P}$. From the assumption that $\hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x}) > \frac{1}{c} \sum_{i=k+1}^{m} \hat{\eta}_{\pi_i(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$, we have $\mathcal{P} < c \cdot \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$ (notice the strict inequality). Therefore, on a single tree level there are at most $c - 1$ nodes not being predecessors of top-k labels for which $\hat{\eta}_v(\boldsymbol{x}) \geq \hat{\eta}_{\pi_k(\hat{\boldsymbol{\eta}}(\boldsymbol{x}))}(\boldsymbol{x})$. Therefore in total on a single tree level, there are at most $k + c - 1$ nodes included in the set $\mathcal{A}$. There are depth_T internal tree levels. Therefore $|\mathcal{A}| \leq (k + c - 1) \cdot \mathrm{depth}_T$. Given the set $\mathcal{A}$, the number of nodes with classifiers called by Algorithm 4 is upper-bounded by: $1 + |\mathcal{A}| \cdot \deg_T \leq 1 + (k + c - 1) \cdot \mathrm{depth}_T \cdot \deg_T$, what concludes the proof. $\square$

Theorem 6.10. *For Algorithm 5 with beam B and any $\boldsymbol{x} \in \mathcal{X}$, we have that:*

$$c_B(T, \boldsymbol{x}) \leq 1 + B \cdot \deg_T \cdot \mathrm{depth}_T$$

where $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$, *and* $\deg_T = \max_{v \in V_T} \deg_v$.

Proof. Algorithm 5 visits the root and all the children of the root (there are up to $\deg_T$ such children). Then, among the $\deg_T$ children of the root, it selects B highest scoring ones, and visits $B \deg_T$ children of these nodes on the next level. Such operation is repeated $(\mathrm{depth}_T - 1)$ times, until $B \deg_T$ leafs are finally evaluated. Therefore we get: $c_B(T, \boldsymbol{x}) \leq 1 + \deg_T + B \cdot \deg_T \cdot (\mathrm{depth}_T - 1) \leq 1 + B \cdot \deg_T \cdot \mathrm{depth}_T$. $\square$

The prediction cost bounds could possibly be made tighter by considering the level of root's children separately. However, such formulation of the bounds and the proofs is less clear, and makes the bounds harder to interpret and compare with each other. Therefore we present the bounds this way, and only mention this possibility.

A.7 Proofs of the results from Chapter 7

Theorem 7.3 concerns two properties, the properness and the efficiency, of an OPLT algorithm. We first prove that the OPLT algorithm satisfies each of the properties in two separate lemmas. The final proof of the theorem is then straightforward.

Lemma A.5. OPLT *is a proper* OPLT *algorithm.*

Proof. We need to show that for any $\mathcal{S}$ and t the two of the following hold. Firstly, that the set L_{T_t} of leaves of tree T_t built by OPLT correspond to $\mathcal{L}_t$, the set of all labels observed in S_t. Secondly, that the set H_t of classifiers trained by OPLT is exactly the same as $H = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_t)$, that is, the set of node classifiers trained incrementally by Algorithm 6 on $\mathcal{D} = \mathcal{S}_t$ and tree T_t given as input parameter. We will prove it by induction with the base case for $\mathcal{S}_0$ and the induction step for $\mathcal{S}_t$, $t \geq 1$, with the assumption that the statement holds for $\mathcal{S}_{t-1}$.

For the base case of $\mathcal{S}_0$, tree T_0 is initialized with the root node r_T with no label assigned and set H_0 of node classifiers with a single classifier assigned to the root. As there are no observations, this classifier receives no updates. Now, notice that IPLT.TRAIN, run on T_0 and $\mathcal{S}_0$, returns the same set of classifiers H that contains solely the initialized root node classifier without any updates (assuming that initialization procedure is always the same). There are no labels in any sequence of 0 observations and also T_0 has no label assigned.

The induction step is more involved as we need to take into account the internal loop which extends the tree with new labels. Let us consider two cases. In the first one, observation $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ does not contain any new label. This means that the tree T_{t-1} will not change, that is, $T_{t-1} = T_t$. Moreover, node classifiers from H_{t-1} will get the same updates for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ as classifiers in IPLT.TRAIN, therefore $H_t = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_t)$. It also holds that $l_j \in L_{T_t}$ iff $j \in \mathcal{L}_t$, since $\mathcal{L}_{t-1} = \mathcal{L}_t$. In the second case, observation $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ has $m' = |\mathcal{L}_{\boldsymbol{x}_t} \setminus \mathcal{L}_{t-1}|$ new labels. Let us make the following assumption for the UPDATETREE procedure, which we later prove that it indeed holds. Namely, we assume that the set $H_{t'}$ of classifiers after calling the UPDATETREE procedure is the same as the one being returned by IPLT.TRAIN$(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$, where T_t is the extended tree. Moreover, leaves of T_t correspond to all observed labels seen so far. If this is the case, the rest of the induction step is the same as in the first case. All updates to classifiers in $H_{t'}$ for $(\boldsymbol{x}_t, \mathcal{L}_{\boldsymbol{x}_t})$ are the same as in IPLT.TRAIN. Therefore $H_t = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_t)$.

Now, we need to show that the assumption for the UPDATETREE procedure holds. To this end, we also use induction, this time on the number m' of new labels. For the base case, we take $m' = 1$. The induction step is proved for $m' > 1$ with the assumption that the statement holds for $m' - 1$.

For $m' = 1$ we need to consider two scenarios. In the first scenario, the new label is the first label in the sequence. This label will be then assigned to the root node r_T. So, the structure of the tree does not change, that is, $T_{t-1} = T_t$. Furthermore, the set of classifiers also does not change, since the root classifier has already been initialized. It might be negatively updated by previous observations. Therefore, we have $H_{t'} = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$. Furthermore, all observed labels are appropriately assigned to the leaves of T_t. In the second scenario, set $\mathcal{L}_{t-1}$ is not empty. We need to consider in this scenario the three variants of tree extension illustrated in Figure 7.1.

In the first variant, tree T_{t-1} is extended by one leaf node only without any additional ones. ADDNODE creates a new leaf node v'' with the new label assigned to the tree. After this operation, the tree contains all labels from $\mathcal{S}_t$. The new leaf v'' is added as a child of the selected node v. This new node is initialized as $\hat{\eta}(v'') = \text{INVERSECLASSIFIER}(\hat{\theta}(v))$. Recall that INVERSECLASSIFIER creates a wrapper that inverts the behavior of the base classifier. It predicts $1 - \hat{\eta}$, where $\hat{\eta}$ is the prediction of the base classifier, and flips the updates, that is, positive updates become negative and negative updates become positive. From the definition of the auxiliary classifier, we know that $\hat{\theta}(v)$ has been trained on all positives updates of $\hat{\eta}(v)$. So, $\hat{\eta}(v'')$ is initialized with a state as if it was updated negatively each time $\hat{\eta}(v)$ was updated positively in sequence S_{t-1}. Notice that in S_{t-1} there is no observation labeled with the new label. Therefore $\hat{\eta}(v'')$ is the same as if it was created and updated using IPLT.TRAIN. There are no other operations on T_{t-1}, so we have that $H_{t'} = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$.

In the second variant, tree T_{t-1} is extended by internal node v' and leaf node v''. The internal node v' is added in INSERTNODE. It becomes a parent of all child nodes of the selected node v and the only child of this node. Thus, all leaves of the subtree of v do not change. Since v' is the root of this subtree, its classifier $\hat{\eta}(v')$ should be initialized as a copy of the auxiliary classifier $\hat{\theta}(v)$, which has accumulated all updates from and only from observations with labels assigned to the leaves of this subtree. The addition of the leaf node v'' can be analyzed as in the first variant. Since nothing else has changed in the tree and in the node classifiers, we have that $H_{t'} = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$. Moreover, the tree contains the new label, so the statement holds.

The third variant is similar to the second one. Tree T_{t-1} is extended by two leaf nodes v' and v'' being children of the selected node v. Insertion of leaf v' is similar to the insertion of node v' in the second variant, with the difference that v does not have any children and its label has to be reassigned to v'. The new classifier in v' is initialized as a copy of the auxiliary classifier $\hat{\theta}(v)$, which contains all updates from and only from observations with the label assigned previously to v. Insertion of v'' is exactly the same as in the second variant. From the above, we conclude that $H_{t'} = \text{IPLT.TRAIN}(T_t, A_{\text{online}}, \mathcal{S}_{t-1})$ and that T_t contains all labels from T_{t-1} and the new label. In this way we prove the base case.

The induction step is similar to the second scenario of the base case. The only difference is that we do not extend tree T_{t-1}, but an intermediate tree with $m' - 1$ new labels already added. Because of the induction hypothesis, the rest of the analysis of the three variants of tree extension is exactly the same. This ends the proof that the assumption for the inner loop holds. At the same time, it finalizes the entire proof. □

Lemma A.6. OPLT *is an efficient* OPLT *algorithm.*

Proof. The OPLT maintains one additional classifier per each node in comparison to IPLT. Hence, for a single observation there is at most one update more for each positive node. Furthermore, the time and space cost of the complete tree building policy is constant per a single label, if implemented with an array list. In this case, insertion of any new node can be made in amortized constant time, and the space required by the array list is linear in the number of nodes. Concluding the above, the time and space complexity of OPLT is in constant factor of C_t and C_s, the time and space complexity of IPLT respectively. This proves that OPLT is an efficient OPLT algorithm. □

Theorem 7.3. OPLT *is a proper and efficient* OPLT *algorithm.*

Proof. The theorem directly follows from Lemma A.5 and Lemma A.6. □

B Implementation and experimental setup

B.1 Batch uniform-cost search

We include a simplified pseudocode of batch uniform-cost search algorithm, published originally in [Jasinska, 2018]. This algorithm retrieves k highest scored labels for N_{test} test instances, processing all of them at once. It is useful if the time of evaluation of a single classifier scales sublinearly with the number of instances for which it is evaluated. This is the case of the PARABEL [Prabhu et al., 2018] implementation of the PLTs model, which transforms a linear model from a sparse format to a dense format before evaluating it. PARABEL originally uses batch beam search. The introduced method shows how to efficiently implement uniform-cost-search-based prediction in the batch setting. Its advantage over the batch beam search is that it delivers exact top k predictions.

The proposed prediction algorithm, given in Algorithm 14, runs uniform-cost search simultaneously for all examples in a batch. It uses an additional data structure, designed for this application, named MULTIQUEUE. This structure stores N_{test} priority queues and keeps track of the frequency of top elements among them. The pseudocode of this data structure is given in Algorithm 13. We firstly briefly describe how the MULTIQUEUE works, and secondly, give the details of the batch uniform-cost search algorithm.

The MULTIQUEUE consists of N_{test} priority queues $\mathcal{Q}_i$, and a master priority queue $\mathcal{Q}$, aggregating the top elements from all the N_{test} queues. Each queue $\mathcal{Q}_i$ sorts elements according to some criterion i. The master queue $\mathcal{Q}$ sorts the elements from queues $\mathcal{Q}_i$ according to the number of criteria on which they are on-top and is capable of providing a list of their indices. In case of use of this data structure as a part of the batch prediction algorithm, N_{test} is the number of instances in a batch, each $\mathcal{Q}_i$ is related to a single test instance and tracks the state of the uniform-cost search for this instance. The elements in the queues are nodes of the label tree, and the priority is the intermediate probability product in

Algorithm 13 MULTIQUEUE

```
function INIT $(n, t)$    ▷ Initialize with the number of instances $n$ and the number of nodes $t$
for $i = 1, \dots n$ do
    $\mathcal{Q}_i = \emptyset$    ▷ Initialize $n$ priority queues
$\mathcal{Q}$ = PRIORITYQUEUEWITHRANDOMACCESS()    ▷ Initialize one priority queue allowing for random access
function PUSH $(i, v)$
$e_{\text{prevTop}} = \mathcal{Q}_i$.top()
$\mathcal{Q}_i$.push($v$)
$e_{\text{currTop}} = \mathcal{Q}_i$.top()
DECREASEPRIORITY($e_{\text{prevTop}}, i$)
INCREASEPRIORITY($e_{\text{currTop}}, i$)
function POP $(i)$
$e_{\text{prevTop}} = \mathcal{Q}_i$.top()
$\mathcal{Q}_i$.pop()
$e_{\text{currTop}} = \mathcal{Q}_i$.top()
DECREASEPRIORITY($e_{\text{prevTop}}, i$)
INCREASEPRIORITY($e_{\text{currTop}}, i$)
function TOPANDPOP $(i)$
top = $\mathcal{Q}$.top()
$\mathcal{Q}$.pop()
$\mathcal{E} = \emptyset$
for $i \in$ top.queues do
    $\mathcal{E}$.add($\mathcal{Q}_i$.pop())
return $\mathcal{E}$
function DECREASEPRIORITY $(v, i)$
$\mathcal{Q}_v$.remove($i$)
$\mathcal{Q}$.decrease($v$)
function INCREASEPRIORITY $(v, i)$
$\mathcal{Q}_v$.add($i$)
$\mathcal{Q}$.increase($v$)
function REMOVE $(v)$
delete $\mathcal{Q}_v$
$\mathcal{Q}$.remove($v$)
```

a given node. Therefore, $\mathcal{Q}$ sorts the nodes of the tree according to the number of instances with the highest intermediate product in this node at this step of the search. However, this is just one of possible sorting criteria.

The exact batch inference Algorithm 14 works as follows. The batch size N_{test} defines how many instances are processed at once. For each batch the prediction algorithm creates a MULTIQUEUE of size N_{test}. Initially, the root node is added to the queues $\mathcal{Q}_i$ of all the instances. Then, until for all instances top k labels are retrieved, the prediction algorithm queries the MULTIQUEUE for the node v to process and a list of instances (queues $\mathcal{Q}_i$) with this node on-top, $\mathcal{I}_v$. If v is a leaf node, the algorithm predicts the corresponding label for all the instances from the list. Otherwise, for each child v' of the node v, it evaluates the child node classifier for all elements in $\mathcal{I}_v$ to obtain a vector of predictions $\hat{\boldsymbol{\eta}}(\boldsymbol{v}')$.Finally, the child nodes and their intermediate estimates $\hat{\eta}_{v,i} \cdot \hat{\eta}(v')_i$, $i \in \mathcal{I}_v$ are added to the relevant queues $\mathcal{Q}_\rangle$ in the MULTIQUEUE. This algorithm visits exactly as many nodes as, or in other words, calculates as many node classifier instance-wise

Algorithm 14 PLT.PREDICTBATCHUCSEARCH$(T, H, k, N_{\text{test}}, X)$

```
1: $\hat{\boldsymbol{Y}} = [\boldsymbol{0}]^n$
2: $\mathcal{MQ} = \textsc{MultiQueue}(N_{\text{test}}, |V|)$
3: for $i = 1, \dots, N_{\text{test}}$ do
4:     $\mathcal{MQ}.\textsc{push}(i, (1, 0, false))$
5: $\boldsymbol{c}_{\text{retrieved}} = [0]^n$    ▷ Set the retrieved count for each instance to zero
6: $\mathcal{I}_{\text{removed}} = \emptyset$.    ▷ Set the set of processed instances to an empty set
7: while $|\mathcal{I}_{\text{removed}}| \neq N_{\text{test}}$ do
8:     $\mathcal{I}_{\text{delete}} = \emptyset$
9:     $v, \mathcal{I}_v, \mathcal{E}_v = \mathcal{MQ}.\textsc{topAndPop}()$
10:    $\hat{\boldsymbol{\eta}}_{\boldsymbol{v}}$ is the $\hat{\eta}_v$ for each instance from $\mathcal{I}_v$, extracted from $\mathcal{E}_v$
11:    if $v$ is a leaf then
12:        for $i \in \mathcal{I}_v$ do
13:            $\hat{\boldsymbol{Y}}_{i,v} = 1$
14:            $\boldsymbol{c}_{\text{retrieved}}[i] + +$
15:            if $\boldsymbol{c}_{\text{retrieved}}[i] = k$ then
16:                $\mathcal{I}_{\text{delete}}.\text{add}(i)$
17:        for $i \in \mathcal{I}_{\text{delete}}$ do
18:            $\mathcal{I}_{\text{removed}}.\text{add}(i)$
19:            $\mathcal{MQ}.\textsc{remove}(i)$
20:    else
21:        for $v' \in \text{Ch}(v)$ do    ▷ For all its child nodes
22:            $\hat{\boldsymbol{\eta}}(\boldsymbol{v}') = \textsc{batchPredict}(v', X, \mathcal{I}_v)$    ▷ Evaluate $v'$ for all instances from $\mathcal{I}_v$
23:            for $i \in \mathcal{I}_v$ do    ▷ For each instance from $\mathcal{I}_v$
24:                $\hat{\eta}_{v'} = \hat{\eta}_{v,i} \times \hat{\eta}(v')_i$    ▷ Compute its intermediate probability product
25:                $\mathcal{MQ}.\textsc{push}((i, (\hat{\eta}_{v'}, v', [\![' v \text{ is a leaf}]\!]))$    ▷ And push to the queue.
26: return $\hat{\boldsymbol{Y}}$.
```

predictions as, the online uniform-cost search applied N_{test} times.

B.2 Synthetic data

All synthetic models use linear models parameterized by a weight vector $\boldsymbol{w}$ of size d. The values of the vector are sampled uniformly from a d-dimensional sphere of radius 1. Each observation $\boldsymbol{x}$ is a vector sampled from a d-dimensional disc of the same radius. To create the multi-class data, we associate a weight vector $\boldsymbol{w}_j$ with each label $j \in \{1, \dots, m\}$. This model assigns probabilities to labels at point $\boldsymbol{x}$ using softmax,

$$\eta_j(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_j^\top \boldsymbol{x})}{\sum_{j'=1}^m \exp(\boldsymbol{w}_{j'}^\top \boldsymbol{x})},$$

and draws the positive label according to this probability distribution.

The multi-label data with conditionally independent labels are created similarly to the multi-class data. The difference lays is normalization as the conditional probabilities do not have to sum up to 1. To get a probability of the j-th label, we use the logistic transformation:

$$\eta_j(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_j^\top \boldsymbol{x})}{1 + \exp(\boldsymbol{w}_j^\top \boldsymbol{x})}.$$

Then, we assign a label to an observation by:

$$y_j = [\![r < \eta_j(\boldsymbol{x})]\!],$$

where the random value r is sampled uniformly and independently from range $[0,1]$, for each instance $\boldsymbol{x}$ and label $j \in \{1, \ldots, m\}$.

Generation of the multi-label data with conditionally dependent labels is more involved. We follow the mixing matrix model previously used to a similar purpose in [Dembczyński et al., 2012]. This model is based on m latent scoring functions generated by $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m)$. The $m \times m$ mixing matrix $\boldsymbol{M}$ introduces dependencies between noise $\boldsymbol{\epsilon}$, which stands for the source of randomness in the model. The models $\boldsymbol{w}_j$ are sampled from a sphere of radius 1, as in previous cases. The values in the mixing matrix $\boldsymbol{M}$ are sampled uniformly and independently from $[-1, 1]$. The random noise vector $\boldsymbol{\epsilon}$ is sampled from $N(0, 0.25)$. The label vector $\boldsymbol{y}$ is then obtained by element-wise evaluation of the following expression:

$$\boldsymbol{y} = [\![\boldsymbol{M}(\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{\epsilon}) > 0]\!]$$

Notice that if $\boldsymbol{M}$ was an identity matrix the model would generate independent labels.

In the experiments, we used the following parameters of the synthetic models: $d = 3$, $n = 100000$ instances (with a $1:1$ split to training and test subsets), and $m = 32$ labels.

B.3 Hyperparameters

In this section, we report all the hyperparameters values used for all the reported experiments.

hyperparameter	description	values
c	LIBLINEAR cost co-efficient, inverse regularization	$\{1\}$
ϵ	LIBLINEAR tolerance of termination criterion	$\{0.01\}$
Δ	threshold value for pruning linear classifiers weights	$\{0.01\}$

Table B.1: Hyperparameters of algorithms used in the experiments in Chapter 9 for DiSMEC.

hyperparameter	description	values
t	number of trees	$\{50\}$
c	SVM weight co-efficient	$\{1.0\}$
l	number of label-probability pairs to retrain in a leaf	$\{100\}$
m	maximum allowed instances in a lead node	$\{10\}$
γ	γ parameter in tail label classifier (PFASTREXML only)	$\{30\}$
α	trade-off parameter between PFASTREXML and tail classifier scores (PFASTREXML only)	$\{0.8\}$
A	parameter of the propensity model (PFASTREXML only)	$\{0.5, 0.55, 0.6\}$
B	parameter of the propensity model (PFASTREXML only)	$\{0.4, 1.5, 2.6\}$

Table B.2: Hyperparameters of algorithms used in the experiments in Chapter 9 for FASTXML and PFASTREXML.

hyperparameter	description	values
λ_1	L1 regularization weight	$\{0.01, 0.1, 1\}$
c	cost of each sample	$\{1\}$
τ	degree of asynchronization	$\{0.1, 1, 10\}$
m	maximum number of iterations allowed	$\{30\}$

Table B.3: Hyperparameters of algorithms used in the experiments in Chapter 9 for PPDSPARSE.

hyperparameter	description	values
ensemble	number of trees in ensemble	$\{1, 3\}$
k-means ϵ	tolerance of termination criterion of the k-means clustering used for the tree building procedure	$\{0.0001\}$
max leaves	maximum degree of pre-leaf nodes	$\{25, 100, 400\}$
c	LIBLINEAR cost co-efficient, inverse regularization strength (PARABEL, NXC only)	$\{1, 8, 16, 32\}$
ϵ	LIBLINEAR tolerance of termination criterion (PARABEL, NXC only)	$\{0.1\}$
Δ	threshold value for pruning weights (PARABEL, NXC only)	$\{0.1, 0.2, 0.3, 0.5\}$
max iter	maximum iterations of LIBLINEAR (PARABEL only)	$\{20\}$
arity	arity of tree nodes, k for k-means clustering (XT, NXC only)	$\{2, 16, 64\}$
η	learning rate for SGD or Adagrad (XT, NXC only)	$\{0.02, 0.2, 0.5, 1\}$
epochs	number of passes over data set when training with incremental algorithm (XT, NXC only)	$\{1, 3, 10\}$
AdaGrad ϵ	determines initial learning rate (NXC only)	$\{0.01, 0.001\}$
λ_2	L2 regularization weight (XT only)	$\{0.001, 0.002, 0.003\}$
dim	size of hidden representation (XT only)	$\{500, 1000\}$

Table B.4: Hyperparameters of different PLTs implementations: PARABEL, EXTREMETEXT (XT) and NAPKINXC PLT and OPLT (NXC), used in the experiments in Chapter 9
.

B.4 Tree depth impact for the squared hinge loss

We present additional results concerning different tree shapes, namely the tree depth, for the squared hinge loss.

Arity	2	16	64	2	16	64
	$p@1$ [%]			T/N_{test} [ms]		
EurLex-4K	80.17±0.27	81.62±0.15	**81.68±0.24**	0.24±0.02	**0.18±0.01**	0.27±0.02
AmazonCat-13K	92.40±0.04	92.39±0.06	**92.46±0.07**	**0.19±0.00**	0.23±0.01	0.31±0.03
Wiki10-30K	84.17±0.10	84.30±0.07	**84.62±0.07**	**2.87±0.08**	2.95±0.20	4.14±0.28
DeliciousLarge-200K	46.30±0.07	46.19±0.08	**46.31±0.06**	**10.07±0.23**	12.59±0.51	11.83±0.48
WikiLSHTC-325K	62.78±0.03	64.17±0.05	**64.61±0.04**	**0.86±0.06**	0.90±0.07	1.42±0.05
WikipediaLarge-500K	66.77±0.08	**68.16±0.10**	68.02±0.01	**2.86±0.07**	4.41±0.12	5.55±0.00
Amazon-670K	43.31±0.03	43.88±0.05	**44.03±0.05**	**1.32±0.08**	1.73±0.15	2.68±0.17
Amazon-3M	46.23±0.01	46.98±0.01	**47.33±0.00**	**1.96±0.05**	2.39±0.09	2.56±0.00
	T_{train} [h]			M_{size} [GB]		
EurLex-4K	**0.01±0.00**	0.01±0.00	0.01±0.00	**0.00±0.00**	0.00±0.00	0.01±0.00
AmazonCat-13K	**0.29±0.00**	0.38±0.04	0.47±0.03	0.19±0.00	**0.18±0.00**	0.18±0.00
Wiki10-30K	**0.11±0.00**	0.16±0.00	0.36±0.02	0.06±0.00	0.05±0.00	**0.04±0.00**
DeliciousLarge-200K	**5.51±0.29**	9.07±0.76	12.23±0.78	1.82±0.00	1.77±0.00	**1.74±0.00**
WikiLSHTC-325K	**1.60±0.06**	2.18±0.07	3.85±0.17	0.97±0.00	0.90±0.00	**0.88±0.00**
WikipediaLarge-500K	**9.48±0.33**	15.91±0.55	28.68±0.74	1.78±0.00	1.52±0.00	**1.49±0.00**
Amazon-670K	**0.40±0.01**	0.64±0.02	1.57±0.04	0.63±0.00	0.55±0.00	**0.52±0.00**
Amazon-3M	**5.44±0.13**	9.82±0.23	20.82±0.00	9.86±0.00	9.36±0.00	**9.24±0.00**

Table B.5: Precision@k1, average prediction time per example, training time and model size for k-means trees of different depths, with squared hinge loss. Results for arity equal to 2, 16 or 64 and pre-leaf node degree equal to 100.

Pre-leaf degree	25	100	400	25	100	400
	$p@1$ [%]			T/N_{test} [ms]		
EurLex-4K	**80.31±0.19**	80.17±0.27	79.21±0.56	**0.09±0.01**	0.24±0.02	0.54±0.02
AmazonCat-13K	92.36±0.04	92.40±0.04	**92.53±0.08**	**0.12±0.02**	0.19±0.00	0.51±0.06
Wiki10-30K	83.72±0.12	**84.17±0.10**	83.90±0.14	**1.92±0.17**	2.87±0.08	6.46±0.27
DeliciousLarge-200K	46.24±0.06	46.30±0.07	**46.37±0.10**	**4.53±0.46**	10.07±0.23	17.12±1.12
WikiLSHTC-325K	61.96±0.03	62.78±0.03	**63.19±0.03**	**0.51±0.03**	0.86±0.06	1.78±0.03
WikipediaLarge-500K	66.01±0.10	66.77±0.08	**66.90±0.06**	**2.29±0.03**	2.86±0.07	6.14±0.11
Amazon-670K	42.93±0.02	**43.31±0.03**	43.25±0.04	**1.12±0.06**	1.32±0.08	2.43±0.08
Amazon-3M	45.84±0.02	46.23±0.01	**46.72±0.01**	**1.27±0.10**	1.96±0.05	5.61±0.23
	T_{train} [h]			M_{size} [GB]		
EurLex-4K	**0.00±0.00**	0.01±0.00	0.02±0.00	**0.00±0.00**	0.00±0.00	0.00±0.00
AmazonCat-13K	**0.17±0.02**	0.29±0.00	0.70±0.05	0.21±0.00	0.19±0.00	**0.18±0.00**
Wiki10-30K	**0.10±0.00**	0.11±0.00	0.30±0.02	0.12±0.00	**0.06±0.00**	0.08±0.00
DeliciousLarge-200K	**3.08±0.13**	5.51±0.29	8.03±0.57	2.30±0.00	1.82±0.00	**1.56±0.00**
WikiLSHTC-325K	**1.54±0.03**	1.60±0.06	2.28±0.10	1.20±0.00	0.97±0.00	**0.84±0.00**
WikipediaLarge-500K	**7.41±0.14**	9.48±0.33	19.32±0.31	2.23±0.00	1.78±0.00	**1.49±0.00**
Amazon-670K	**0.39±0.01**	0.40±0.01	0.68±0.02	0.80±0.00	0.63±0.00	**0.51±0.00**
Amazon-3M	**4.80±0.29**	5.44±0.13	12.45±0.74	11.80±0.00	9.86±0.00	**8.60±0.00**

Table B.6: Precision@k1, average prediction time per example, training time and model size for k-means trees of different depths, with squared hinge loss. Results for arity equal to 2 and pre-leaf node degree equal to 25, 100, or 400.

B.5 Weight pruning

In all the experiments, we used a threshold of 0.1 for weight pruning. We present results for higher values of threshold and analyze their impact on the predictive and computational performance of PLTs. Tables B.7 and B.8 report results for logistic and squared hinge loss.

	0.1	0.3	0.5	0.1	0.3	0.5
		$p@1$ [%]			$p@5$ [%]	
EurLex-4K	**80.51±0.16**	80.48±0.13	80.13±0.17	**53.33±0.68**	53.23±0.69	52.80±0.69
AmazonCat-13K	**93.04±0.02**	93.02±0.02	92.94±0.03	**63.70±0.02**	63.67±0.01	63.58±0.02
Wiki10-30K	85.36±0.09	**85.46±0.07**	85.35±0.06	**63.84±0.07**	63.75±0.06	63.62±0.09
DeliciousLarge-200K	**49.55±0.05**	48.60±0.11	46.35±0.18	**39.90±0.02**	39.55±0.04	38.31±0.15
WikiLSHTC-325K	**61.96±0.03**	61.95±0.03	61.82±0.03	**30.19±0.02**	30.18±0.02	30.11±0.02
WikipediaLarge-500K	**66.20±0.05**	65.95±0.11	65.52±0.07	**36.83±0.01**	36.65±0.04	36.40±0.02
Amazon-670K	**43.54±0.01**	43.23±0.02	42.67±0.02	**35.15±0.03**	34.81±0.03	34.16±0.02
Amazon-3M	**46.09±0.02**	45.94±0.01	45.57±0.01	**40.98±0.01**	40.82±0.01	40.35±0.01
		T/N_{test} [ms]			M_{size} [GB]	
EurLex-4K	0.39±0.03	**0.38±0.05**	0.38±0.04	0.02±0.00	**0.01±0.00**	**0.01±0.00**
AmazonCat-13K	0.32±0.03	0.21±0.01	**0.17±0.01**	0.35±0.00	0.17±0.00	**0.10±0.00**
Wiki10-30K	5.35±0.32	3.95±0.14	**2.80±0.11**	0.58±0.00	0.17±0.00	**0.09±0.00**
DeliciousLarge-200K	9.89±0.89	5.25±0.27	**3.03±0.30**	0.95±0.00	0.18±0.00	**0.06±0.00**
WikiLSHTC-325K	1.77±0.11	1.21±0.09	**1.02±0.04**	2.73±0.00	1.38±0.00	**0.89±0.00**
WikipediaLarge-500K	6.67±0.23	4.71±0.15	**4.15±0.03**	8.89±0.00	2.90±0.00	**1.55±0.00**
Amazon-670K	4.13±0.28	2.53±0.06	**2.20±0.11**	2.26±0.00	0.77±0.00	**0.42±0.00**
Amazon-3M	3.26±0.08	2.54±0.11	**2.03±0.09**	20.84±0.00	8.54±0.00	**4.61±0.00**

Table B.7: Precision@k for $k = 1, 5$, average prediction times, and model sizes with different thresholds of weight pruning for logistic loss.

	0.1	0.2	0.3	0.1	0.2	0.3
		$p@1$ [%]			$p@5$ [%]	
EurLex-4K	**80.17±0.27**	78.32±0.19	74.52±0.44	**53.01±0.87**	50.45±0.97	45.99±0.96
AmazonCat-13K	**92.40±0.04**	92.11±0.03	90.99±0.13	**63.88±0.02**	63.37±0.02	61.49±0.05
Wiki10-30K	**84.17±0.10**	79.33±0.24	76.47±0.47	**63.12±0.04**	59.75±0.16	49.12±0.44
DeliciousLarge-200K	**46.30±0.07**	43.63±0.19	37.75±0.40	**36.54±0.07**	35.14±0.03	31.65±0.64
WikiLSHTC-325K	**62.78±0.03**	60.77±0.06	56.83±0.13	**30.25±0.02**	29.12±0.02	27.05±0.05
WikipediaLarge-500K	**66.77±0.08**	63.88±0.00	59.62±0.13	**36.94±0.02**	34.95±0.00	32.25±0.05
Amazon-670K	**43.31±0.03**	40.59±0.02	35.67±0.06	**34.31±0.03**	31.14±0.04	26.49±0.04
Amazon-3M	**46.23±0.01**	44.74±0.00	39.87±0.02	**41.41±0.01**	39.67±0.00	35.16±0.02
		T/N_{test} [ms]			M_{size} [GB]	
EurLex-4K	**0.24±0.02**	0.24±0.02	0.28±0.02	**0.00±0.00**	0.00±0.00	0.00±0.00
AmazonCat-13K	0.19±0.00	0.14±0.02	**0.11±0.01**	0.19±0.00	0.07±0.00	**0.03±0.00**
Wiki10-30K	2.87±0.08	1.69±0.05	**1.17±0.06**	0.06±0.00	**0.01±0.00**	0.01±0.00
DeliciousLarge-200K	10.07±0.23	4.06±0.25	**1.59±0.18**	1.82±0.00	0.38±0.00	**0.15±0.00**
WikiLSHTC-325K	0.86±0.06	0.55±0.02	**0.45±0.01**	0.97±0.00	0.24±0.00	**0.10±0.00**
WikipediaLarge-500K	2.86±0.07	**1.99±0.00**	2.05±0.07	1.78±0.00	0.39±0.00	**0.17±0.00**
Amazon-670K	1.32±0.08	**1.01±0.02**	1.17±0.02	0.63±0.00	0.18±0.00	**0.10±0.00**
Amazon-3M	1.96±0.05	1.09±0.00	**0.95±0.02**	9.86±0.00	2.31±0.00	**0.89±0.00**

Table B.8: Precision@k for $k = 1, 5$, average prediction times, and model sizes with different thresholds of weight pruning for squared hinge loss.

We observe that for logistic loss a more aggressive pruning can be beneficial. Precision@k decreases only slightly, while testing time can be reduced almost by two, and the model size even by 4. For squared hinge loss, precision@k drops more substantially, but the model size can be even reduced by a factor of 10.

Weight pruning has also been investigated by Prabhu et al. [2018], with similar outcomes to those presented here.

B.6 Precision@$1, 3, 5$ of state-of-the-art methods

Table B.9 gives the precision@$1, 3, 5$ of methods compared in Table 9.13. It additionally includes the precision@$1, 3, 5$ of other state-of-the-art methods discussed in Section 1.2. Besides the results of decision-tree-based FastXML, PfastreXML, CRAFTML, and LdSM, and PLT-based Parabel and napkinXC, we include the results of smart 1-vs-All PPDSparse, DiSMEC, and ProXML, embedding-based AnnexML and GLaS, and deep-learning-based X-BERT and AttentionXML. For these methods we present the best performance results found in the literature [Guo et al., 2019, You et al., 2019, Tagami, 2017, Chang et al., 2019, Majzoubi and Choromanska, 2019, Siblini et al., 2018, Babbar and Schölkopf, 2018, Bhatia et al., 2016].

	p@1 [%]	p@3 [%]	p@5 [%]	p@1 [%]	p@3 [%]	p@5 [%]
	EurLex-4K			AmazonCat-13K		
FastXML	71.26	59.80	50.28	93.03	78.22	63.38
PfastreXML	70.21	59.26	50.59	85.62	75.31	62.83
CRAFTML	78.81	65.21	53.71	92.78	78.48	63.58
LdSM	–	–	–	93.87	75.41	57.86
PPDSparse	**83.83**	70.72	**59.21**	92.72	78.14	63.41
DiSMEC	83.67	70.70	59.14	92.72	78.11	63.40
ProXML	83.40	**70.90**	59.10	–	–	–
AnnexML	79.66	64.94	53.52	93.55	78.38	63.32
GLaS	77.50	65.01	54.37	**94.21**	**79.70**	**64.84**
Parabel-T=3	81.80	68.67	57.45	93.24	79.17	64.51
nXC-T=3	81.94	68.94	57.49	93.37	79.01	64.27
X-BERT	86.00	74.52	62.64	95.17	80.65	65.19
AttentionXML	87.12	73.99	61.92	95.92	82.41	67.31
	Wiki10-30K			DeliciousLarge-200K		
FastXML	82.97	67.58	57.68	43.17	38.70	36.22
PfastreXML	75.58	64.38	57.25	17.44	17.28	17.19
CRAFTML	85.19	73.17	63.27	47.87	41.28	38.01
LdSM	83.74	71.74	61.51	45.26	40.53	38.23
PPDSparse	73.80	60.90	50.40	45.05	38.34	34.90
DiSMEC	85.20	**74.60**	**65.90**	45.50	38.70	35.50
ProXML	–	–	–	–	–	–
AnnexML	**86.50**	74.28	63.19	46.66	40.79	37.64
GLaS	–	–	–	46.40	40.49	38.10
Parabel-T=3	84.49	72.57	63.66	46.62	39.78	36.37
nXC-T=3	85.90	74.45	64.84	**49.65**	**43.18**	**39.97**
X-BERT	85.75	75.19	65.13	–	–	–
AttentionXML	87.47	78.48	69.37	–	–	–

	p@1 [%]	p@3 [%]	p@5 [%]	p@1 [%]	p@3 [%]	p@5 [%]
	WikiLSHTC-325K			WikipediaLarge-500K		
FastXML	49.85	33.16	24.49	49.32	33.48	25.84
PfastreXML	58.50	37.69	27.57	59.58	40.26	30.73
CRAFTML	56.57	34.73	25.03	–	–	–
LdSM	–	–	–	–	–	–
PPDSparse	64.13	42.10	31.14	70.16	50.57	39.66
DiSMEC	64.94	42.71	31.50	**70.20**	**50.60**	**39.70**
ProXML	63.60	41.50	30.80	68.80	48.90	37.90
AnnexML	63.36	40.66	29.79	64.22	43.15	32.79
GLaS	65.46	**$\underline{45.44}$**	**$\underline{34.51}$**	69.91	49.08	38.35
Parabel-T=3	64.95	43.21	32.01	68.66	49.48	38.60
nXC-T=3	**$\underline{65.57}$**	43.64	32.33	69.24	49.82	38.81
X-BERT	–	–	–	67.87	46.73	35.97
AttentionXML	–	–	–	$\underline{76.95}$	$\underline{58.42}$	$\underline{46.14}$
	Amazon-670K			Amazon-3M		
FastXML	36.90	33.22	30.44	45.26	41.96	39.80
PfastreXML	36.97	34.18	32.05	32.62	32.67	32.35
CRAFTML	37.34	33.31	30.62	–	–	–
LdSM	42.63	38.09	34.70	–	–	–
PPDSparse	45.32	40.37	36.92	–	–	–
DiSMEC	45.37	40.40	36.96	47.77	44.96	42.80
ProXML	43.50	38.70	35.30	–	–	–
AnnexML	42.09	36.65	32.76	**49.30**	**45.55**	**43.11**
GLaS	**46.38**	**42.09**	**38.56**	–	–	–
Parabel-T=3	44.70	39.66	35.85	47.52	44.69	42.57
nXC-T=3	45.10	40.00	36.22	47.83	45.08	42.98
X-BERT	–	–	–	–	–	–
AttentionXML	$\underline{47.58}$	$\underline{42.61}$	$\underline{38.92}$	$\underline{50.86}$	$\underline{48.04}$	$\underline{45.83}$

Table B.9: Precision of PLTs compared to state-of-the-art algorithms. We separate the deep learning based methods which use raw text data. We mark with **bold** the best result among non-deep methods, and underline the best result among all methods. For clarity, we skip the standard errors.

Let us first focus on the non-deep methods. While there is no single best performing method, notice that some methods tend to obtain better results than other. Decision-tree-based methods under-perform compared to methods designed under other principles. Smart 1-vs-All methods, modern embedding based methods, and PLTs obtain highest precision, with no single method outperforming others. Deep-learning-based methods advantage from raw text representation and usually outperform methods using sparse features.

C Notation

Observations	
m	number of labels
$\mathcal{L}$	set of m possible labels
j	a label
$\mathcal{X}$	an instance space
$\boldsymbol{x}$	an instance
$\mathcal{L}_{\boldsymbol{x}}$	a set of positive labels
$\boldsymbol{y}$	a label vector
$\lvert\lvert\boldsymbol{y}\rvert\rvert_1$	number of positive labels in $\boldsymbol{y}$, $\lvert\mathcal{L}_{\boldsymbol{x}}\rvert$
y_j	if label j is positive or not
$\mathcal{Y} = \{0,1\}^m$	all possible label vectors
$(\boldsymbol{x},\boldsymbol{y})$	an observation
$\mathbf{P}(\boldsymbol{y}\lvert\boldsymbol{x})$	conditional (given $\boldsymbol{x}$) joint probability of a label vector
$\eta_j(\boldsymbol{x})$	conditional probability of label j, $\mathbf{P}(y_j = 1 \,\lvert\, \boldsymbol{x})$
$\hat{\eta}_j(\boldsymbol{x})$	estimate of $\eta_j(\boldsymbol{x})$
$\mathcal{D}$	training set $\{(\boldsymbol{x}_i,\boldsymbol{y}_i)\}_{i=1}^n$
n	number of observations in $\mathcal{D}$
$\mathcal{S}$	sequence of training observations in online setting
Losses, risks, regrets	
ℓ	loss
ℓ_c	strongly proper composite loss
$R_\ell(h)$	risk of h under ℓ
$\mathrm{reg}_\ell(h)$	regret of h under ℓ
$\mathrm{reg}_\ell(h\lvert\boldsymbol{x})$	conditional regret of h under ℓ
ψ	link function

Tree	
T	tree
V_T	the set of tree nodes
$\mathcal{I}(T)$	set of internal nodes of tree T
L_T	the set of tree leafs
v	tree node
r_T	the root of tree T
l_j	the leaf node corresponding to label j
L_v	the set of leafs in the subtree of T rooted in an inner node v
$\mathcal{L}_v$	the set of labels corresponding to leaf nodes in L_v
$\mathrm{Ch}(v)$	children of node v
$\mathrm{pa}(v)$	parent of node v
$\mathrm{Path}(v)$	the path from node v to the root
$\eta_j(\boldsymbol{x})$	conditional probability of label, $\mathbf{P}(y_j = 1 \vert \boldsymbol{x})$
len_v	length of the path, the number of nodes on the path
deg_v	degree of node v, $\vert\mathrm{Ch}(v)\vert$
deg_T	$\max_{v \in V_T} \mathrm{deg}_v$
depth_T	depth of the tree, $\mathrm{depth}_T = \max_{v \in L_T} \mathrm{len}_v - 1$
PLT	
$\boldsymbol{z}$	vector of length $\vert V_T\vert$ corresponding to $\boldsymbol{y}$ in which $z_v = [\![\sum_{j \in \mathcal{L}_v} y_j \geq 1]\!]$
$\eta(\boldsymbol{x}, v)$	$\mathbf{P}(z_v = 1 \vert z_{\mathrm{pa}(v)} = 1, \boldsymbol{x})$ for non-root nodes, $\mathbf{P}(z_v = 1 \,\vert\, \boldsymbol{x})$ for the root
$\hat{\eta}(v)$	node classifier
H_T	set of classifiers
$\theta(v)$	an auxiliary node classifier in OPLT
Θ_T	set of auxiliary classifiers
A	learning algorithm
A_{online}	online learning algorithm
Prediction algorithms, predictions, computational cost	
k	number of predicted labels with highest scores
$\hat{\mathcal{L}}_{\boldsymbol{x}}$	set of predicted labels
$\boldsymbol{\pi}(\boldsymbol{s})$	ranking of labels according to diminishing scores $\boldsymbol{s}$
$\pi_j(\boldsymbol{s})$	rank of label j in ranking $\boldsymbol{\pi}(\boldsymbol{s})$
$\pi_r^{-1}(\boldsymbol{s})$	label at rank r in ranking $\boldsymbol{\pi}(\boldsymbol{s})$
B	beam width
$c(T, \boldsymbol{y})$	training cost per observation
$C_{\mathbf{P}}(T)$	expected training cost
$c_\tau(T, \boldsymbol{x})$	prediction cost per observation using Algorithm 3
$C_{\mathbf{P}(\boldsymbol{x}),\tau}(T)$	expected prediction cost using Algorithm 3
$c_k(T, \boldsymbol{x})$	prediction cost per observation using Algorithm 4
$c_B(T, \boldsymbol{x})$	prediction cost per observation using Algorithm 5
Evaluation	
T_{train}	training time
T/N_{test}	average prediction time per example
M_{size}	model size
avg. $\vert\mathcal{L}_{\boldsymbol{x}}\vert$	average number of positive labels per example

Streszczenie

Uczenie maszynowe to dziedzina nauki z pogranicza informatyki, teorii informacji i statystyki. Rozwój uczenia maszynowego dotyczy zarówno jego aspektów teoretycznych, jak i praktycznych, a kolejne obszary zastosowania uczenia maszynowego otwierają nowe obszary badań teoretycznych. Wyróżnia się trzy główne obszary uczenia maszynowego: uczenie nadzorowane, uczenie nienadzorowane oraz uczenie ze wzmocnieniem. Niniejsza rozprawa dotyczy problemu klasyfikacji, będącego obiektem badań uczenia nadzorowanego.

W problemach klasyfikacji zadaniem jest poprawne wskazanie etykiety (lub zbioru etykiet) dla instancji, będącej reprezentacją pewnego obiektu, na podstawie jej cech. Poprawność tego wskazania określa się za pomocą funkcji straty. Aby rozwiązać problem klasyfikacji zazwyczaj trenuje się klasyfikator będący funkcją przypisującą instancjom etykiety na podstawie ich cech. Taki klasyfikator jest wynikiem wykonania algorytmu uczącego, działającego na danych treningowych, składających się z zaobserwowanych instancji z przypisanymi etykietami. Pożądaną własnością algorytmu uczącego jest to, że mając do dyspozycji coraz większą próbkę danych treningowych dostarcza on klasyfikator ze stratą coraz bliższą najniższej możliwej stracie, osiąganej przez tak zwany klasyfikator bayesowski. Taką własność określamy statystyczną zgodnością.

Klasyfikacja ekstremalna

W standardowych problemach klasyfikacji liczba istniejących etykiet jest nieduża. Jednak w wielu nowoczesnych obszarach zastosowania uczenia maszynowego mogą istnieć nawet miliony etykiet. Problemy z tak dużą liczą etykiet rozważane są w dziedzinie klasyfikacji ekstremalnej. Przykładowymi problemami klasyfikacji ekstremalnej są tagowanie dokumentów tekstowych [Dekel and Shamir, 2010], rekomendacja słów kluczowych dla reklam internetowych [Prabhu and Varma, 2014], rekomendacja wideo [Weston et al., 2013], czy predykcja kole-

jnego słowa w zdaniu [Mikolov et al., 2013]. Aby lepiej zrozumieć jak te problemy można przedstawić jako problem klasyfikacji przeanalizujmy następujące przykłady. W przypadku tagowania dokumentów tekst stanowi cechy, a kategorie to etykiety. W przypadku rekomendacji słów kluczowych dla reklam internetowych, cechy są tworzone na podstawie strony docelowej reklamy, a zapytania do wyszukiwarki stanowią etykiety. W przypadku rekomendacji wideo, użytkownik i wideo mogą być zamiennie używane jako cechy i etykiety. We wszystkich tych problemach liczba możliwych etykiet jest bardzo duża.

Celem klasyfikacji ekstremalnej jest predykcja określonej liczby k adekwatnych etykiet lub stworzenie ich rankingu dla danej instancji. Jakość predykcji często jest mierzona za pomocą precyzji na k-tym miejscu (*precision*@k), zdefiniowanej jako udział pozytywnych etykiet wśród k przewidzianych, czy NDCG@k, będącej miarą typową dla problemów rangowania, która przypisuje poprawnie przewidzianym etykietom zysk malejący wraz z rangą etykiety. Innymi miarami używanymi w klasyfikacji ekstremalnej są czułość na k-tym miejscu i makro-uśredniana miara F_1.

Z uwagi na liczbę istniejących etykiet, efektywność obliczeniowa algorytmów uczenia i predykcji w klasyfikacji ekstremalnej odgrywa większą rolę niż w problemach mniejszej skali. W klasyfikacji ekstremalnej nawet algorytmy skalujące się liniowo z liczbą etykiet mogą być zbyt wolne aby mogły być użyteczne. Przykładem takiego algorytmu jest standardowe rozwiązanie problemów wieloetykietowych, polegające na nauczeniu niezależnego klasyfikatora dla każdej etykiety z osobna. Takie podejście nazywane jest 1-VS-ALL (jeden przeciwko wszystkim/innym). Charakteryzuje się ono liniową wobec liczby etykiet złożonością czasową i pamięciową. Taka złożoność jest zbyt duża w wielu praktycznych zastosowaniach. Przykładowo, rozważmy problem z 10^6 etykietami. Załóżmy, że pojedynczy klasyfikator można wytrenować w jedną sekundę. W takim przypadku trening 10^6 klasyfikatorów zająłby ponad 11 dni. Zauważmy również, że w sytuacji gdy w zbiorze treningowym znajduje się bardzo wiele cech i obserwacji, zakładany czas treningu wynoszący jedną sekundę może być zdecydowanie zbyt niski. Również predykcja z użyciem tego podejścia jest czasochłonna, ponieważ wymaga ewaluacji 10^6 klasyfikatorów. Ponadto, gdy rozważymy również rozmiar modelu, przy założeniu istnienia 10^5 cech i użycia gęstych klasyfikatorów liniowych, łatwo uzyskujemy rozmiary modelu rzędu setek gigabajtów. Widzimy, że konieczne jest stworzenie bardziej zaawansowanych rozwiązań, cechujących się dobrą jakością predykcji i niższą od liniowej złożonością.

Zauważmy, że klasyfikacja ekstremalna stawia również inne, nie tylko obliczeniowe, wyzwania nieobecne w standardowych problemach uczenia. Przykładowo, dla wielu etykiet w zbiorach uczących znajduje się bardzo mało obserwacji. Spotyka się również problemy, w których etykiety nie posiadają żadnych obserwacji w zbiorze uczącym. Jest to tak zwany problem uczenia z zera próbek. Ponadto, dane treningowe są zazwyczaj niskiej jakości, ponieważ jest niemożliwe aby ręcznie zweryfikować wszystkie możliwe etykiety nawet dla pojedynczej obserwacji. Często dane uczące są uzyskiwane nie wprost, co prowadzi do całego spektrum problemów. W niniejszej pracy skupiamy się jednak przede wszystkim na jakości predykcji oraz efektywności obliczeń, a nie na wyżej wymienionych wyzwaniach.

Istniejące metody

W dziedzinie klasyfikacji ekstremalnej zaproponowanych zostało wiele metod mających na celu osiągnięcie wysokiej jakości predykcji przy niskich czasach uczenia i predykcji. Standardowym podejściem cechującym się dużą wydajnością i jakością predykcji są drzewa decyzyjne. Jednakże w klasyfikacji ekstremalnej nie można zastosować standardowych drzew decyzyjnych właśnie ze względu na ich koszty obliczeniowe, wysokie w przypadku problemów ekstremalnych [Agrawal et al., 2013], wynikające z konieczności obliczenia kryterium podziału wierzchołka. Algorytm FASTXML [Prabhu and Varma, 2014] redukuje ten koszt przez użycie klasyfikatorów liniowych we wierzchołkach decyzyjnych drzewa. Te klasyfikatory są wynikiem naprzemiennej optymalizacji specyficznej wielokryterialnej funkcji celu. FASTXML używa wielu drzew w celu poprawy jakości predykcji. Idea FASTXML została rozszerzona w PFASTREXML [Jain et al., 2016] i CRAFTML [Siblini et al., 2018], które modyfikują optymalizowane kryterium lub sposób jego optymalizacji. W pełni przyrostową metodę budowy drzewa dla problemów wieloklasowych rozważono w [Choromanska and Langford, 2015], proponując LOMTREE. Z kolei w [Majzoubi and Choromanska, 2019] zaproponowano algorytm LDSM dla problemów wieloklasowych, budujący drzewo wierzchołek po wierzchołku w częściowo przyrostowy sposób. Przez długi czas metody oparte na drzewach decyzyjnych osiągały najlepsze wyniki pod względem jakości predykcji i wydajności obliczeniowej.

Innym podejściem stosowanym w klasyfikacji ekstremalnej są zanurzenia. Metody tego typu redukują oryginalną przestrzeń wyjść do przestrzeni o mniejszej liczbie wymiarów i tworzą modele regresyjne w tej zredukowanej przestrzeni [Tai and Lin, 2012], a następnie transformują predykcje z przestrzeni zredukowanej do oryginalnej. Różnią się one sposobem wykonania tej kompresji i dekompresji. W [Bhatia et al., 2015] zaproponowano rzadkie lokalne zanurzenia (*sparse local embeddings*, SLEEC) używające klasyfikatora k najbliższych sąsiadów w przestrzeni zredukowanej do dekompresji predykcji. Jakość predykcji algorytmu SLEEC na zbiorach porównawczych jest porównywalna z jakością innych algorytmów. Jednak istotną wadą tej metody są duże rozmiary modeli i długie czasy treningu i predykcji. ANNEXML [Tagami, 2017] poprawia efektywność predykcji przez użycie grafu k najbliższych sąsiadów. Ostatnio, GLAS [Guo et al., 2019] osiągnął konkurencyjną jakość predykcji i zredukował czas potrzebny na ich uzyskanie poprzez użycie informacji o współwystąpieniach etykiet oraz szybkich metod wyszukiwania największego iloczynu skalarnego.

Jakość predykcji metody 1-VS-ALL długo była uważana za trudną do osiągnięcia metodami mniej kosztownymi obliczeniowo. Tak zwane sprytne metody 1-VS-ALL trenują jeden binarny klasyfikator dla każdej etykiety, ale redukują koszty obliczeniowe przez użycie rozproszonych obliczeń i ucinania wag (DISMEC, [Babbar and Schölkopf, 2017]), odpowiednich metod optymalizacji i predykcji (PD-SPARSE, [Yen et al., 2016]; PPD-SPARSE, [Yen et al., 2017]), czy obu (PROXML, [Babbar and Schölkopf, 2019]). Te metody osiągają bardzo wysoka jakość predykcji, jednak ich czasy obliczeń i treningu nadal pozostają znacznie dłuższe niż czasy osiągane przez inne metody.

Model probabilistycznych drzew etykiet (*probabilistic label trees*, PLT), będący pierwszą metodą wieloetykietowej klasyfikacji ekstremalnej opartą na drzewach etykiet, został zaproponowany w [Jasinska et al., 2016]. Drzewa etykiet umożliwiają efektywne przybliżenie modelu 1-VS-ALL, przy krótszych czasach predykcji i treningu. Drzewa etykiet różnią się znacznie od drzew decyzyjnych, ponieważ w drzewie etykiet każda ścieżka odpowiada dokładnie jednej etykiecie, a nie fragmentowi przestrzeni cech. Podejście oparte na modelu PLT zostało później wykorzystane w takich algorytmach jak EXTREMETEXT Wydmuch et al. [2018], PARABEL [Prabhu et al., 2018], BONSAI TREE [Babbar and Schölkopf, 2019], czy ATTENTIONXML [You et al., 2019]. Powyższe algorytmy należą do najbardziej popularnych i uznanych algorytmów wieloetykietowej klasyfikacji ekstremalnej.

Metody oparte na uczeniu głębokim również zostały zastosowane do klasyfikacji ekstremalnej. Metody te, zastosowane do danych tekstowych, używają oryginalnej reprezentacji danych, podczas gdy pozostałe metody używają rzadkich reprezentacji tekstu. Z tego względu trudno wprost porównać jakość predykcji tych metod do jakości predykcji pozostałych. Również ze względu na użycie obliczeń na kartach graficznych, czasy treningu i predykcji nie są wprost porównywalne. Pierwszą metodą tego typu w wieloetykietowej klasyfikacji ekstremalnej jest XML-CNN [Liu et al., 2017]. Nie dość, że osiąga on gorszą jakość predykcji niż inne metody, to charakteryzuje się bardzo długimi czasami treningu i predykcji. Wspomniany wcześniej ATTENTIONXML [You et al., 2019] używa płytkiego PLT i specyficznego mechanizmu wieloetykietowej uwagi. Również X-BERT [Chang et al., 2019] może być traktowany jako PLT z klasyfikatorami we wierzchołkach wewnętrznych opartymi o sieci głębokie, a dokładniej o przedtrenowany model BERT [Devlin et al., 2018], oraz liniowe w liściach.

Wiele metod zostało zaproponowanych w celu rozwiązania problemów stawianych przez wieloetykietową klasyfikację ekstremalną. Jakkolwiek te metody pozwalają osiągnąć wysoką jakość predykcji szybciej niż naiwnie zaaplikowana metoda 1-VS-ALL, to niewiele z nich zostało przeanalizowanych pod względem statystycznej zgodności ze względu na optymalizowane miary oceny, czy ze względu na złożoność obliczeniową.

Motywacje

Motywację zaproponowanych probabilistycznych drzew etykiet stanowi prosta obserwacja: okazuje się, że optymalne predykcje, czyli tak zwane klasyfikatory bayesowskie, ze względu na precyzję na k-tym miejscu i inne popularne metryki, można określić za pomocą prawdopodobieństw warunkowych (ze względu na cechy) etykiet. A zatem, estymacja tych prawdopodobieństw i użycie odpowiedniej reguły decyzyjnej mogłyby zagwarantować statystyczną zgodność zaproponowanej metody. Z tej perspektywy, problem wieloetykietowej klasyfikacji ekstremalnej wydaje się problemem efektywnej estymacji prawdopodobieństw i efektywnego wnioskowania.

Taka efektywność estymacji prawdopodobieństw warunkowych etykiet może być uzyskana przez zorganizowanie etykiet w drzewo, w którym każdej etykiecie odpowiada jeden liść, czyli w tak zwane drzewo etykiet. Tego typu metody są używane w klasyfikacji wieloklasowej. Przykładem jest hierarchiczny softmaks [Morin and Bengio, 2005], używany w sieciach głębokich, między innymi w przetwarzaniu języka naturalnego [Mikolov et al., 2013]. Co ciekawe, podobne algorytmy były zaproponowane niezależnie w innych dziedzinach. W statystyce znane są jako *nested dichotomies* [Fox, 1997], w wieloklasowej regresji jako drzewa prawdopodobieństw warunkowych [Beygelzimer et al., 2009b], a w rozpoznawaniu wzorców jako wieloetapowe klasyfikatory [Kurzynski, 1988]. Jednakże to podejście nie zostało wcześniej zastosowane w wieloetykietowej klasyfikacji ekstremalnej.

Cel i kontrybucje

Ze względu na przedstawione wcześniej motywacje, sformułowano następującą hipotezę rozprawy:

> Istnieje klasa statystycznie zgodnych algorytmów uczenia dla wieloetykietowej klasyfikacji ekstremalnej charakteryzujących się podliniową złożonością obliczeniową względem liczby etykiet.

Poniżej opisany jest główny wkład przedstawiony w rozprawie.

Postaci klasyfikatorów bayesowskich

W rozprawie dokonujemy przeglądu miar oceny jakości klasyfikacji używanych w ekstremalnej klasyfikacji wieloetykietowej. Dowodzimy, że klasyfikatorem bayesowskim dla precyzji na k-tym miejscu jest wskazanie k etykiet z najwyższym prawdopodobieństwem warunkowym. Podobnie pokazujemy postaci klasyfikatora bayesowskiego dla miar DCG@k oraz NDCG@k. Dodatkowo na podstawie literatury omawiamy postaci klasyfikatora bayesowskiego dla uogólnionych miar oceny jakości klasyfikacji [Kotłowski and Dembczyński, 2017] oraz czułości na $@k$-tym miejscu [Menon et al., 2019]. W ten sposób pokazujemy, dla których miar oceny jakości klasyfikacji optymalne predykcje mogą być określone na podstawie warunkowych prawdopodobieństw etykiet.

Model PLT

Proponujemy i opisujemy model probabilistycznych drzew etykiet (en. *probabilistic label trees*, PLT). Probabilistyczne drzewa etykiet używają drzewa etykiet do rozkładu prawdopodobieństwa warunkowego etykiet poprzez zastosowanie reguły łańcuchowej wzdłuż ścieżki od korzenia drzewa do liścia odpowiadającego etykiecie. W ten sposób redukują one oryginalny problem wieloetykietowy do wielu problemów klasyfikacji (estymacji) binarnej. Z tego punktu widzenia PLT jest przedstawicielem *redukcji uczenia* [Beygelzimer et al., 2016]. PLT używa probabilistycznych klasyfikatorów binarnych we wszystkich wierzchołkach drzewa do estymacji odpowiednich czynników, będących prawdopodobieństwami warunkowymi. Iloczyn estymat prawdopodobieństwa na ścieżce od korzenia do liścia jest estymatą prawdopodobieństwa warunkowego etykiety odpowiadającej liściowi. Do efektywnej predykcji PLT używa odpowiednich procedur opartych na przeszukiwaniu drzewa.

Rozważamy dwa sposoby uczenia PLT: trening wsadowy lub przyrostowy przy danej strukturze drzewa etykiet, oraz trening w pełni przyrostowy, w którym drzewo jest konstruowane jednocześnie z treningiem klasyfikatorów. Dowodzimy specyficzną tożsamość klasyfikatora PLT nauczonego przyrostowo oraz w pełni przyrostowo. Analizujemy trzy sposoby przeszukiwania drzewa w celu predykcji. Pierwszy odnajduje wszystkie etykiety o estymowanym prawdopodobieństwie warunkowym przekraczającym wskazany próg. Drugi z nich, oparty na przeszukiwaniu ze strategią jednolitego kosztu odnajduje wskazaną liczbę etykiet o najwyższych estymatach prawdopodobieństwa warunkowego. Trzeci, oparty na przeszukiwaniu wiązkowym, odnajduje przybliżone etykiety z najwyższą estymatą.

Zgodność i ograniczenia na żal

Wyniki teoretyczne związane z PLT dotyczą jego zgodności ze względu na wspomniane wcześniej miary oceny jakości klasyfikacji. Zgodność wykazujemy zgodnie z metodyką redukcji uczenia [Beygelzimer et al., 2016], ograniczając błąd L_1 estymacji prawdopodobieństw warunkowych etykiet za pomocą funkcji żalu klasyfikatorów wierzchołkowych, wyrażonego ze względu na silnie właściwy złożony błąd zastępczy. W tym celu wpierw ograniczamy błąd L_1 estymacji prawdopodobieństwa warunkowego etykiety za pomocą błędów L_1 estymacji prawdopodobieństw warunkowych związanych z wierzchołkami na ścieżce od korzenia do liścia odpowiadającego etykiecie. Następnie wyrażamy błąd L_1 każdego wierzchołka za pomocą silnie właściwego złożonego błędu zastępczego [Agarwal, 2014]. To pozwala nam połączyć żal klasyfikatorów wierzchołkowych z błędem L_1 estymacji prawdopodobieństw warunkowych etykiet. Ten wynik stanowi podstawę kolejnych ograniczeń ze względu na różne miary oceny jakości klasyfikacji, dla których optymalne predykcje można określić za pomocą prawdopodobieństw warunkowych etykiet.

Za pomocą wyprowadzonego ograniczenia na błąd L_1, pokazujemy ograniczenia żalu ze względu na uogólnione miary oceny jakości klasyfikacji. Do tej klasy funkcji należy między innymi strata Hamminga oraz mikro- i makro- uśredniana miara F_1. Wyprowadzone ograniczenia bazują na wynikach z [Kotłowski and Dembczyński, 2017] dotyczących metody 1-VS-ALL. Następnie rozważamy precyzję na k-tym miejscu. Definiujemy żal oraz ograniczamy go za pomocą błędu L_1 estymacji prawdopodobieństw etykiet. W ten sposób pokazujemy, że PLT używające dokładnej metody predykcji k etykiet z najwyższym estymowanym prawdopodobieństwem warunkowym, jest dostosowane do optymalizacji precyzji na k-tym miejscu. W podobny sposób analizujemy miarę DCG@k, pokazując analogiczne wyniki dla tej miary.

Analizujemy także związek pomiędzy PLT i hierarchicznym softmaksem. Pokazujemy, że PLT jest poprawnym uogólnieniem hierarchicznego softmaksu do problemów wieloetykietowych. Oznacza to, że dla danych wieloklasowych model PLT redukuje się do modelu hierarchicznego softmaksu. Ponadto pokazujemy, że inna popularna metoda uogólnienia hierarchicznego softmaksu, stosująca heurystykę wyboru jednej etykiety, stosowana przykładowo w FASTTEXT [Joulin et al., 2017] i LEARNED TREE [Jernite et al., 2017], nie jest zgodna ze względu na estymację prawdopodobieństw warunkowych etykiet przy błędzie L_1 i ze względu na precyzję na k-tym miejscu.

Złożoność obliczeniowa algorytmów uczących i predykcyjnych

Analizujemy złożoność obliczeniową algorytmów PLT służących do uczenia i predykcji. Pokazujemy, że uczenie klasyfikatorów PLT przy określonej strukturze drzewa, przy pewnych dodatkowych założeniach dotyczących struktury drzewa oraz maksymalnej liczby pozytywnych etykiet na obserwację, może być wykonany w czasie logarytmicznym ze względu na liczbę etykiet. Ponadto, pokazujemy, że przy dodatkowych założeniach dotyczących estymat prawdopodobieństw, również czas predykcji jest logarytmiczny, lub podliniowy, we względu na liczbę etykiet. Wyniki te nie są trywialne, ponieważ metody przeszukiwania drzewa na których bazuje predykcja w najgorszym przypadku mogą przeszukać całe drzewo, co prowadziłoby do złożoności liniowej.

Ocena empiryczna

Poza wynikami teoretycznymi analizujemy istniejące implementacje ogólnego schematu PLT, takie jak XMLC-PLT [Jasinska et al., 2016], PLT-VW[1], PARA-

[1]`https://github.com/VowpalWabbit/vowpal_wabbit`

BEL [Prabhu et al., 2018], BONSAI TREE [Khandagale et al., 2019], EXTREME-TEXT [Wydmuch et al., 2018], ATTENTIONXML [You et al., 2019], oraz NAPKINXC [Jasinska-Kobus et al., 2020a]. W analizie koncentrujemy się na możliwych sposobach implementacji ze względu na reprezentację cech i modeli oraz metody treningu i predykcji.

W części eksperymentalnej przede wszystkim koncentrujemy się na implementacjach NAPKINXC oraz PARABEL. Za ich pomocą analizujemy różne instancje modelu PLT i porównujemy uzyskiwane przez nie wyniki do powszechnie uznanych metod bazujących na drzewach decyzyjnych oraz metodach 1-VS-ALL. Empirycznie wykazujemy, że PLT jest konkurencyjne wobec najlepszych metod, i uzyskuje najwyższe wartości precyzji na pierwszym miejscu na większości zbiorów porównawczych, będąc jednocześnie trzy rzędy wielkości szybsze od metod 1-VS-ALL.

Przegląd prac stanowiących podstawę rozprawy

Poniżej dokonujemy przeglądu prac dotyczących PLT stanowiących podstawę rozprawy. Pierwsza praca dotycząca PLT [Jasinska and Dembczyński, 2015] została przedstawiona na warsztacie *Extreme Classification Workshop* przy konferencji ICML 2015. Ten artykuł wprowadzał model PLT oraz inny model, BRT, również będący drzewem etykiet. PLT wraz z prostymi metodami treningu i predykcji zostało następnie opublikowane w [Jasinska et al., 2016]. Ta praca dotyczy użycia PLT dla optymalizacji precyzji na k-tym miejscu oraz makro-uśrednianej miary F_1. Następnie w [Jasinska, 2018] zaproponowano wsadowy wariant predykcji używającej przeszukiwania ze strategią jednolitego kosztu. Następnie w [Wydmuch et al., 2018] przeanalizowano błąd L_1 estymacji oraz ograniczono żal ze względu na precyzję na k-tym miejscu. Złożoność obliczeniowa PLT została przeanalizowana w [Busa-Fekete et al., 2019]. Niniejsza rozprawa zawiera część wyników z tej pracy, dotyczących ograniczeń kosztów treningu i predykcji. W pełni przyrostowe PLT zostało zaproponowane w roku 2016, a opublikowane w [Jasinska-Kobus et al., 2020c,d]. Większość wyników teoretycznych przedstawionych w niniejszej rozprawie znajduje się w [Jasinska-Kobus et al., 2020a]. Niniejsza rozprawa zawiera również rezultaty niezależne od PLT. W [Jasinska and Karampatziakis, 2016] przedstawiono inny algorytm klasyfikacji ekstremalnej nazwany LTLS. Wyniki dotyczące NDCG@k zostały przedstawione w [Jasinska and Dembczyński, 2018].