A. Probabilistic label trees

An example of a probabilistic label tree (PLT) for 4 labels (y_1, y_2, y_3, y_4) is given in Fig. 2. To estimate posterior probability $\eta(\boldsymbol{x}, j) = \mathbf{P}(y_j = 1 \,|\, \boldsymbol{x})$, PLT uses a path from a root to the j-th leaf. In each node t, we associate with a training instance $(\boldsymbol{x}, \boldsymbol{y})$ a label z_t such that:

$$z_t = [\![\textstyle\sum_{j \in L(t)} y_j \geq 1]\!] \quad \text{(or equivalently } z_t = \textstyle\bigvee_{j \in L(t)} y_j \text{)}$$

Recall that $L(t)$ is a set of all leaves of a subtree with the root in the t-th node. In leaf nodes the labels z_j, $j \in L$, correspond to original labels y_j.

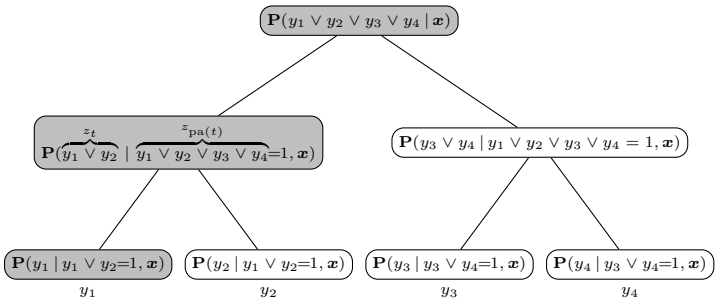

Figure 2. An example of a probabilistic label tree for 4 labels (y_1, y_2, y_3, y_4).

Consider the leaf node j and the path from the root to this leaf node. Using the chain rule of probability, we can express $\eta(\boldsymbol{x}, j)$ in the following way:

$$\eta(\boldsymbol{x}, j) = \prod_{t \in \mathrm{Path}(j)} \eta_T(\boldsymbol{x}, t)\,, \tag{10}$$

where $\eta_T(\boldsymbol{x}, t) = \mathbf{P}(z_t = 1 \,|\, z_{\mathrm{pa}(t)} = 1, \boldsymbol{x})$, for all non-root nodes t, and $\eta(\boldsymbol{x}, t) = \mathbf{P}(z_t = 1 \,|\, \boldsymbol{x})$, if t is the root node (denoted by $r(T)$).

To see the correctness of (10) notice that $z_t = 1$ implies $z_{\mathrm{pa}(t)} = 1$. So, for non-root nodes t and $\mathrm{pa}(t)$ we have:

$$\begin{aligned}
\eta_T(\boldsymbol{x}, t)\eta_T(\boldsymbol{x}, \mathrm{pa}(t)) &= \mathbf{P}(z_t = 1 \,|\, z_{\mathrm{pa}(t)} = 1, \boldsymbol{x}) \mathbf{P}(z_{\mathrm{pa}(t)} = 1 \,|\, z_{\mathrm{pa}(\mathrm{pa}(t))} = 1, \boldsymbol{x}) \\
&= \frac{\mathbf{P}(z_t = 1, z_{\mathrm{pa}(t)} = 1, \boldsymbol{x})}{\mathbf{P}(z_{\mathrm{pa}(t)} = 1, \boldsymbol{x})} \frac{\mathbf{P}(z_{\mathrm{pa}(t)} = 1, z_{\mathrm{pa}(\mathrm{pa}(t))} = 1, \boldsymbol{x})}{\mathbf{P}(z_{\mathrm{pa}(\mathrm{pa}(t))} = 1, \boldsymbol{x})} \\
&= \frac{\mathbf{P}(z_t = 1, \boldsymbol{x})}{\mathbf{P}(z_{\mathrm{pa}(t)} = 1, \boldsymbol{x})} \frac{\mathbf{P}(z_{\mathrm{pa}(t)} = 1, \boldsymbol{x})}{\mathbf{P}(z_{\mathrm{pa}(\mathrm{pa}(t))} = 1, \boldsymbol{x})} \\
&= \frac{\mathbf{P}(z_t = 1, \boldsymbol{x})}{\mathbf{P}(z_{\mathrm{pa}(\mathrm{pa}(t))} = 1, \boldsymbol{x})}\,.
\end{aligned}$$

In words, the probability associated with the parent node $\mathrm{pa}(t)$ cancels out and we can express $\eta_T(\boldsymbol{x}, t)\eta_T(\boldsymbol{x}, \mathrm{pa}(t))$ as the product of probabilities associated only with node t and its grandparent node $\mathrm{pa}(\mathrm{pa}(t))$. Applying the above rule consecutively to $\prod_{t \in \mathrm{Path}(j)} \eta_T(\boldsymbol{x}, t)$ and recalling that for the root note $\eta_T(\boldsymbol{x}, r(T)) = \mathbf{P}(z_{r(T)} = 1 \,|\, \boldsymbol{x})$, we finally get $\eta(\boldsymbol{x}, j)$.

Below we show that PLTs posses strong theoretical guarantees. We derive a relation between estimation error minimized by the node classifiers and estimation error of posterior probabilities $\eta(\boldsymbol{x}, j)$. This relation states that we can upperbound the latter error by the former. This also implies that for optimal node classifiers we get optimal multi-label classifier in terms of estimation of posterior probabilities.

We are interested in bounding the estimation error of posterior probabilities of labels at point $\boldsymbol{x}$

$$\ell(\eta(\boldsymbol{x}), \hat{\eta}(\boldsymbol{x})) = \frac{1}{m} \sum_{j=1}^{m} |\eta(\boldsymbol{x}, j) - \hat{\eta}(\boldsymbol{x}, j)|\,,$$

in terms of an estimation error of node classifiers

$$\ell(\eta_T(\boldsymbol{x}, t), \hat{\eta}_T(\boldsymbol{x}, t)) = |\eta_T(\boldsymbol{x}, t) - \hat{\eta}_T(\boldsymbol{x}, t)|\,.$$

Expressing $\eta(\boldsymbol{x}, j)$ and $\hat{\eta}(\boldsymbol{x}, j)$ by (10) and applying Lemma 2 from (Beygelzimer et al., 2009a), we get:

$$|\eta(\boldsymbol{x}, j) - \hat{\eta}(\boldsymbol{x}, j)| \leq \sum_{t \in \text{Path}(j)} |\eta_T(\boldsymbol{x}, t) - \hat{\eta}_T(\boldsymbol{x}, t)| \,. \tag{11}$$

Equation (11) already shows that minimization of the estimation error by node classifiers improves the overall performance of PLTs. We can show, however, even a more general result concerning surrogate regret bounds by referring to the theory of strongly proper composite losses (Agarwal, 2014).

Assume that a node classifier has a form of a real-valued function f_t. Moreover, there exists a strictly increasing (and therefore invertible) link function $\psi : [0, 1] \to \mathbb{R}$ such that $f_t(\boldsymbol{x}) = \psi(\hat{\eta}_T(\boldsymbol{x}, t))$. Recall that the regret of f_t in terms of a loss function ℓ at point $\boldsymbol{x}$ is defined as:

$$\text{reg}_\ell(f_t \,|\, \boldsymbol{x}) = L_\ell(f_t \,|\, \boldsymbol{x}) - L_\ell^*(\boldsymbol{x}) \,,$$

where $L_\ell(f_t \,|\, \boldsymbol{x})$ is the expected loss at point $\boldsymbol{x}$:

$$L_\ell(f \,|\, \boldsymbol{x}) = \mathbb{E}_{y_j \,|\, \boldsymbol{x}} \left[\ell(y_j, f_t(\boldsymbol{x}))\right] \,,$$

and $L_\ell^*(\boldsymbol{x})$ is the minimum expected loss at point $\boldsymbol{x}$.

If a node classifier is trained by a learning algorithm that minimizes a strongly proper composite loss, e.g., squared, exponential, or logistic loss, like in our implementation (see in Appendix B), then the bound (11) can be expressed in terms of the regret of this loss function:

$$\left|\eta_T(\boldsymbol{x}, t) - \psi^{-1}(f_t)\right| \leq \sqrt{\frac{2}{\lambda}} \sqrt{\text{reg}_\ell(f_t \,|\, \boldsymbol{x})}$$

where λ is a strong properness constant specific for a given loss function (for more detail, see (Agarwal, 2014)). By putting the above inequality into (11), we get

$$|\eta(\boldsymbol{x}, j) - \hat{\eta}(\boldsymbol{x}, j)| \leq \sum_{t \in \text{Path}(j)} |\eta_T(\boldsymbol{x}, t) - \hat{\eta}_T(\boldsymbol{x}, t)| = \sum_{t \in \text{Path}(j)} \left|\eta_T(\boldsymbol{x}, t) - \psi^{-1}(f_t)\right| \leq \sum_{t \in \text{Path}(j)} \sqrt{\frac{2}{\lambda}} \sqrt{\text{reg}_\ell(f_t \,|\, \boldsymbol{x})}$$

B. Training of node classifiers

In each node t we trained a linear classifier $f_t(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$, where $\boldsymbol{x} = (1, x_1, \ldots, x_p)$. To this end we used a variant of stochastic gradient descent to minimize logistic loss. Albeit successfully used in large scale learning, the optimization of empirical loss using stochastic gradient descent is a particularly challenging task when the number of features and labels is large. The step function should ensure a quick convergence in order to reduce the number of required training epochs. Furthermore, it should support sparse updates of the weights (i.e., only weights for non-zero features should be updated to ensure fast training time). Duchi & Singer (2009) propose a two phase gradient step:

$$\begin{aligned} \boldsymbol{w}_{t+\frac{1}{2}} &= \boldsymbol{w}_t - \gamma_t \boldsymbol{g}_t \\ \boldsymbol{w}_{t+1} &= \arg\min_{\boldsymbol{w}} \left\{ \frac{1}{2} \left\| \boldsymbol{w} - \boldsymbol{w}_{t+\frac{1}{2}} \right\|^2 + \lambda \gamma_t r(\boldsymbol{w}) \right\} \end{aligned}$$

where $\boldsymbol{w}_t$ is the weight vector at time step t, $r(\boldsymbol{w})$ is a regularization function, λ is regularization parameter, and γ_t is an adaptive step size, and $\boldsymbol{g}_t$ is the gradient vector at $\boldsymbol{x}_t$ of logistic loss applied to the linear model f_t.

For stochastic gradient descent with L_2^2 regularization, the step function reduces to

$$\boldsymbol{w}_{t+1} = \frac{\boldsymbol{w}_t - \gamma_t \boldsymbol{g}_t}{1 + \lambda \gamma_t}$$

By using

$$\Pi_t = \prod_{i=1}^{t} (1 + \lambda \gamma_t) \quad \text{and} \quad \tilde{\boldsymbol{w}}_t = \Pi_t \boldsymbol{w}_t \,,$$

we can rewrite the step function to the following form:

$$\tilde{\boldsymbol{w}}_{t+1} = \tilde{\boldsymbol{w}}_t - \Pi_t \gamma_t \boldsymbol{g}_t$$

Thanks to this transformation, we are able to make sparse updates by storing only the current value of Π_t (one value for each node classifier). This is because the i-th component of $\tilde{\boldsymbol{w}}$ does not change when x_i is zero. More formally,

$$\tilde{w}_{i,t+1} = \tilde{w}_{i,t}, \text{if } g_{i,t} = 0\,.$$

During prediction or computation of gradient $\boldsymbol{g}_t$, we use:

$$\boldsymbol{w}_t = \frac{\tilde{\boldsymbol{w}}_t}{\Pi_t}\,.$$

In our implementation we adapt the step size γ_t as suggested in (Bottou, 2012):

$$\gamma_t = \frac{\gamma}{1 + \lambda\gamma t}\,,$$

where γ is an initial parameter.

C. Tuning of hyperparameters

A PLT has only one global hyperparameter which is the degree of the tree denoted by b. The other hyperparameters are associated with the node classifiers. To tune the stochastic gradient descent described above we varied values of γ, λ, and the number of epochs. All hyperparameters were tuned by using the open-source hyperparameter optimizer SMAC (Hutter et al., 2011) with a wide range of parameters, which is reported in Table 3. The validation process was carried out by using a $80/20$ split of the training data for every dataset we used.

Table 3. The hyperparameters of the PLT method and their ranges used in hyperparameter optimization,

Hyperparameter	Validation range
b	$\{2, \ldots, 256\}$
λ	$[10 - 0.000001]$
γ	$[10 - 0.000001]$
Num. of epochs	$\{5, \ldots, 30\}$

D. F-scores by tuning the input parameters $\mathbf{a}$ and $\mathbf{b}$ of OFO algorithms

In our experimental study described in Section 6, we did not tune the input parameter $\mathbf{a}$ of the OFO algorithm but set all of its components to 1. We carried out experiments for assessing the impact of the input parameter $\mathbf{a}$ on the performance of OFO. Its optimal value was selected from the set $C = \{10000, 1000, 200, 100, 50, 20, 10, 7, 5, 4, 3, 2\}$ based on the same validation process like in case of $\mathbf{b}$ and we took into account the fact that a_i/b_i should be in range $(\hat{\pi}_j/(\hat{\pi}_j + 1), 0.5]$, as it was pointed out in (5). The macro F-scores computed for the test and validation set are shown in Table 4 along with the validated values of a_i and b_i. For sake of readability, we repeat here the scores achieved by STO and FTA reported earlier in Table 2. The macro F-scores achieved by OFO are slightly better thanks to the additional degree of freedom, and thus, the OFO algorithm outperforms FTA and STO algorithm on almost every datasets except the Amazon dataset in which case the OFO and FTA algorithms are tied.

Table 4. The test macro F-scores obtained by validating both input parameters a and b. The numbers in bold indicate the best score achieved on each dataset.

Algorithm	Dataset	a_i	b_i	OFO Valid. F-score	OFO Test F-score	FTA Test F-score	STO Test F-score
PLT	RCV1	300	20000	22.20	**22.00**	20.41	21.16
PLT	AmazonCat	700	5000	33.37	35.30	34.83	31.64
PLT	wiki10	100	200	55.27	**30.28**	29.98	24.02
PLT	Delicious-200K	100	200	34.88	**11.20**	11.12	10.96
PLT	WikiLSHTC	100	200	39.94	14.00	12.31	16.22
PLT	Amazon	100	200	54.84	51.28	51.77	46.94
FASTXML	RCV1	1000	100000	19.84	19.28	17.04	19.58
FASTXML	AmazonCat	10000	100000	50.21	**41.48**	41.07	37.28
FASTXML	wiki10	5000	100000	54.72	29.91	29.88	28.26
FASTXML	Delicious-200K	100	500	35.02	**11.20**	11.18	10.83
FASTXML	WikiLSHTC	5000	100000	45.78	**21.38**	21.24	20.41
FASTXML	Amazon	10000	100000	53.91	**52.86**	**52.86**	47.53