CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning

Wissam Siblini, Pascale Kuntz, Frank Meyer

CRAFTML, an Efficient Clustering-based Random Forest for Extreme Multi-label Learning

Wissam Siblini [1 2] **Pascale Kuntz** [1] **Frank Meyer** [2]

Abstract

Extreme Multi-label Learning (XML) considers large sets of items described by a number of labels that can exceed one million. Tree-based methods, which hierarchically partition the problem into small scale sub-problems, are particularly promising in this context to reduce the learning/prediction complexity and to open the way to parallelization. However, the current best approaches do not exploit tree randomization which has shown its efficiency in random forests and they resort to complex partitioning strategies. To overcome these limits, we here introduce a new random forest based algorithm with a very fast partitioning approach called CRAFTML. Experimental comparisons on nine datasets from the XML literature show that it outperforms the other tree-based approaches. Moreover with a parallelized implementation reduced to five cores, it is competitive with the best state-of-the-art methods which run on one hundred-core machines.

1. Introduction

Multi-label classification has received a tremendous attention in the last decade and recently, stimulated by real-life applications involving large datasets (e.g. image (Partalas et al., 2015) and text (Deng et al., 2009) annotation and product recommendation (McAuley et al., 2015)), it has been extended to problems where the number of labels can exceed one million (Agrawal et al., 2013). In this new context, called eXtreme Multi-label Learning (XML), most of the classical algorithms face scalability issues (Weston et al., 2013) or performance degradation. In an attempt to overcome these challenges, researchers have recently explored three directions: (i) using optimization tricks such as primal/dual conversion or sparsification (Yen et al., 2016) and parallelization on supercomputers (Babbar & Schölkopf, 2017; Yen et al., 2017), (ii) reducing the data dimensionality for solving a smaller size problem with a classical approach (Yu et al., 2014; Bhatia et al., 2015) or (iii) hierarchically partitioning the initial problem into small scale sub-problems (Jasinska et al., 2016; Jain et al., 2016). The tree-based decomposition of the third approach has several advantages. It can exploit a hierarchical structuration of the problem either hidden in the mass of data or available for users like the hierarchy of the Wikipedia Article Categories (Partalas et al., 2015) or the ImageNet tags (Deng et al., 2009). Moreover, by decomposing learning into subtasks, it reduces the learning/prediction complexity and opens the way to parallelization. Finally, its sequence of successive decisions allows a great expressivity.

Motivated by those properties, we here present a novel fast and accurate tree-based approach called CRAFTML (Clustering-based RAndom Forest of predictive Trees for extreme Multi-label Learning). Similarly to PFastReXML (Jain et al., 2016) which is among the best tree-based approaches for XML, CRAFTML is a forest of decision trees trained with the supervision of the labels where the splitting conditions are based on all the features. But CRAFTML has two major differences with PFastReXML: (i) it exploits a random forest strategy which not only randomly reduces both the feature and the label spaces to obtain diversity but also replaces random selections with random projections to preserve more information; (ii) it uses a novel low-complexity splitting strategy which avoids the resolution of a multi-objective optimization problem at each node. Numerical experiments on nine datasets from the XML literature show that CRAFTML outperforms the existing XML tree-based approaches with a lower training time and a smaller memory size. Moreover, it has relevant advantages over the most accurate methods today (DISMEC (Babbar & Schölkopf, 2017) and PPDSparse (Yen et al., 2017)): its model size is smaller and its training/prediction complexities are much lower. Experiments confirm that CRAFTML remains competitive with these approaches which both run on one hundred-core machines. In addition, with a parallelized implementation on a five-core machine, the training time of CRAFTML becomes lower than the training time

[1]Computer Science Laboratory of Nantes (LS2N), France [2]Orange Labs Lannion, France. Correspondence to: Wissam Siblini <wissam.siblini@univ-nantes.fr>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

of DISMEC for all datasets and close to that of PPDSparse for the large dataset Amazon-670K with 490k instances and 670k labels. By taking advantage of the data sparsity and the hash-based dimensionality reduction, and by implementing a very fast partitioning strategy, CRAFTML leads to a high scalability and a very good tradeoff between accuracy, computing resources and execution speed.

The rest of the paper is organized as follows. Section 2 briefly recalls previous recent works in extreme multi-label learning. Section 3 describes our new proposal CRAFTML. Section 4 analyzes its temporal and spatial complexities and discusses the choice of the hyperparameter values. Section 5 compares the performances of CRAFTML with the best performers from the recent literature.

2. Related Works

Due to the increasing interest of multi-label classification in the last decade, many algorithms have been proposed (Tsoumakas & Katakis, 2006; Zhang & Zhou, 2014). Several numerical experimentations have highlighted the performances of the multi-label k-nearest neighbors (ML-kNN) (Zhang & Zhou, 2007) and of the multi-label random forest (RF-PCT) (Kocev et al., 2007). However, they cannot scale up anymore with the new dimensionality orders (10^5 to 10^7) of the eXtreme Multi-label Learning. Three different strategies are today developed to tackle the scaling issue: optimization tricks and parallelization, dimensionality reduction and hierarchical decomposition.

Optimization Tricks and Parallelization (PDSparse (Yen et al., 2016), PPDSparse (Yen et al., 2017), DISMEC (Babbar & Schölkopf, 2017)). PDSparse resorts to a sparse linear model regularized with an elastic net to minimize the ranking loss measure. It cleverly exploits a primal-dual conversion of the optimization problem and the sparsity to scale-up to XML data. It has been recently extended to a parallel model called PPDSparse. DISMEC is based on a regularized one-vs-rest large margin linear model. By exploiting the independence between the submodels associated to each label, computations are accelerated with a parallelization of the training stage. And the memory consumption is reduced with a parameter thresholding which leads to a sparse model.

Dimensionality Reduction (WSABIE (Weston et al., 2011), LEML (Yu et al., 2014), SLEEC (Bhatia et al., 2015), AnnexML (Tagami, 2017)). Generally speaking, dimensionality reduction offers a synthetized representation of the data with a lower number of variables (features, labels, or both). LEML and WSABIE were among the first dimensionality reduction approaches applied to XML. They create a low-rank reduced version of the data, but they miss the information brought by the long tail distribution of the labels, and the performances of the classifiers are debased (Bhatia et al., 2015). To overcome this difficulty, SLEEC and AnnexML train a set of local low-rank embeddings on a partition of the feature space to cover a global high-rank embedding of good quality. In SLEEC, the partition is deduced from an unsupervised clustering of the items. In AnnexML, the partition aims at gathering in a same class items which share tail labels. Only AnnexML and SLEEC are retained in our numerical experiments as they significantly outperform WSABIE and LEML.

Tree-based Methods (LPSR (Weston et al., 2013), FastXML (Prabhu & Varma, 2014), its extension PFastReXML (Jain et al., 2016), PLT (Jasinska et al., 2016)). Roughly speaking, the tree-based methods transform the initial large-scale problem into a series of small-scale subproblems by hierarchically partitioning the instance set or the label set. These different subsets are associated to the nodes of a tree. The initial whole set associated to the root is partitioned into a fixed number k of subsets which are themselves associated to k child nodes of the root. The decomposition process is repeated until a stop condition is checked on the subsets. In each node, two optimization problems are raised: (i) computing a partition for a given criterion, and (ii) defining a condition or building a classifier on the feature space to decide to which subset(s) of the partition an instance is assigned. In the prediction phase, the input instance follows a path from the root to a leaf (instance tree) or several leaves (label tree) determined by the successive local decisions. For a label tree, the labels associated to the reached leaves are those predicted with a non-zero probability. For an instance tree, the prediction is given by a local classifier trained on the leaf instances.

Two pioneering approaches RF-PCT and HOMER (Tsoumakas et al., 2008) presented in the last decade have highlighted the interest of the tree-based strategy for multi-label learning. In the XML literature, three recent tree-based approaches have been proposed: LPSR and FastXML respectively based on a single k-ary instance tree and a forest of binary instance trees, and PLT based on a label tree (Bengio et al., 2010; Deng et al., 2011). The partitioning criterion of LPSR aims at regrouping in a same subset the instances which share common features and labels. And the assignation of an instance to a subset depends on its proximity to the subset feature centroid. The partitioning criterion of FastXML is a nDCG-based label ranking loss which tends to regroup instances with common labels. And instances are assigned to one of the $k = 2$ subsets with a sparse splitting hyperplane on the feature space. FastXML has been very recently extended to a new version refered to as PFastReXML in order to better take into account the tail labels. PLT recursively builds label subsets which regroup labels

that occur in the same instances and stops its decomposition when the subsets contain a single label. A multi-label classifier is trained to estimate the probabilities of following the different root-to-leaf paths in the tree conditionally to the input instance features. Path ending on leaves associated to relevant labels are expected to have a high probability.

These approaches have obtained competitive results in numerical experiments. But we believe that there is still room for improvement by exploring two directions: using very fast partitioning strategies and exploiting tree feature/label randomization. Indeed, to build the tree nodes, the current XML approaches resort to complex optimization processes which can be replaced with simpler operations with lower complexities. Moreover, tree diversity, implemented with random feature selection, contributed to the success of random forests (Breiman, 2001) and RF-PCT in multi-label learning. In the XML literature, this has been tested with feature random selection in the MLRF approach (Agrawal et al., 2013) but the obtained predictive performances are limited compared to FastXML for one main reason: its partitioning strategy (Prabhu & Varma, 2014). A projection is able to preserve more information than a selection for a same ratio of compression and a joint random projection of features and labels is more promising to deal with the extreme number of labels. To address these shortcomings, we have introduced a novel tree-based approach called CRAFTML.

3. CRAFTML

In the following, we consider a training set of n instances, each described by a vector $x \in \mathbb{R}^{d_x}$ of d_x features and a vector $y \in \{0,1\}^{d_y}$ of d_y labels. The feature (resp. label) matrix X (resp. Y) of the set is the $n \times d_x$ (resp. $n \times d_y$) matrix where each row corresponds to the feature (resp. label) vector of an instance.

3.1. The Method Main Steps

CRAFTML computes a forest F of m_F k-ary instance trees whose construction follows the common scheme of the instance tree-based methods (see Algorithm 1) recalled in Section 2. The stop condition of the recursive partitioning of a tree is classical: (i) the cardinality of the node's instance subset is lower than a given threshold n_{leaf}, (ii) all the instances have the same features, or (iii) all the instances have the same labels. Once a tree has been trained, its leaves store the average label vector of their associated instances. Similarly to FastXML the nodes partitioning objective of CRAFTML is to regroup instances with common labels in a same subset but its computation is very different. Our goal was to develop a strategy driven by two constraints: partition computation must be based on randomly projected instances to ensure diversity and must perform low complexity operations for scalability. Therefore, the node training

Algorithm 1 trainTree

Input: Training set with a feature matrix X and a label matrix Y.
Initialize node v
v.isLeaf $\leftarrow$testStopCondition(X, Y)
if v.isLeaf $=$ false **then**
  v.classif $\leftarrow$ trainNodeClassifier(X, Y)
  $(X_{child_i}, Y_{child_i})_{i=0,..,k-1} \leftarrow$ split$(v$.classif$, X, Y)$
  for i **from** 0 **to** $k-1$ **do**
    $v.child_i \leftarrow$trainTree$(X_{child_i}, Y_{child_i})$
  end for
else
  $v.\widehat{y} \leftarrow$computeMeanLabelVector(Y)
end if
Output: node v

Algorithm 2 trainNodeClassifier

Input: feature matrix (X_v) and label matrix (Y_v) of the instance set of the node v.
$X_s, Y_s \leftarrow$ sampleRows(X_v, Y_v, n_s)
$X'_s \leftarrow X_s P_x$ # random feature projection
$Y'_s \leftarrow Y_s P_y$ # random label projection
$\boldsymbol{c} \leftarrow k$-means$(Y'_s, k)$ # $\boldsymbol{c} \in \{0, ..., k-1\}^{\min(n_v, n_s)}$
for i **from** 0 **to** $k-1$ **do**
  (classif)$_{i,.} \leftarrow$ computeCentroid$(\{(X'_s)_{j,.} | c_j = i\})$
end for
Output: Classifier classif $(\in \mathbb{R}^{k \times d'_x})$.

$\boldsymbol{c}$ is a vector where the j^{th} component c_j denotes the cluster index of the j^{th} instance associated to $(X'_s)_{j,.}$ and $(Y'_s)_{j,.}$.

stage in CRAFTML is decomposed into three consecutive steps (see Algorithm 2):

1. a random projection into lower dimensional spaces of the label and feature vectors of the node's instances.
2. a k-means based partitioning of the instances into k temporary subsets from their projected labels.
3. The training of a simple multi-class classifier to assign each instance to its relevant temporary subset (i.e. cluster index computed at step 2) from its feature vector. The instances are partitioned into k final subsets (child nodes) by the classifier ("split" in Algorithm 1).

In the prediction phase, for each tree, the input instance follows a root-to-leaf path determined by the successive decisions of the classifier and the provided prediction is the average label vector stored in the leaf reached. The forest aggregates the tree predictions with the average operator.

Let us specify that contrary to classical random forests which use bootstraps, each tree of CRAFTML is trained

on the full initial dataset. In XML, instance samples can miss a large number of labels and we want each tree to consider every label. The main similarities and differences between CRAFTML and the other state-of-the-art tree-based approaches are summarized in Table 1.

Table 1. Comparison of the characteristics of the tree-based methods. **depends on the memory size, **random feature selection.*

	Label trees		Instance Trees			
	HOMER	PLT	RF-PCT	LPSR	FastXML	CRAFTML
Adapted for XML	No	Yes	No	Yes	Yes	Yes
Feature random projections	No	Yes*	Yes**	No	No	Yes
Label random projections	No	No	No	No	No	Yes
Several trees	No	No	Yes	No	Yes	Yes
Binary trees	No	No	Yes	No	Yes	No
Partitioning condition is defined on multiple features	Yes	Yes	No	Yes	Yes	Yes

3.2. Node Training in Detail

We here detail the three steps of the instance partitioning process in each node v of a tree T of a forest F.

Step 1: Random Projection of the Instances of v The feature and label vectors x and y of each instance of v are projected into a space with a lower dimensionality: $x' = xP_x$ and $y' = yP_y$ where P_x (resp P_y) is a random projection matrix of $\mathbb{R}^{d_x \times d'_x}$ (resp. $\mathbb{R}^{d_y \times d'_y}$) and d'_x (resp. d'_y) is the dimension of the reduced feature (resp. label) space. The projection matrices are different from one tree to another. To optimize memory, the coefficients of the projection matrices are not stored but generated with a seed when needed. We have tested two random projections: a projection generated from a standard Gaussian distribution and a sparse orthogonal projection hereafter referred to as the hashing trick (Weinberger et al., 2009) which has only one non-zero parameter with a value of -1 or $+1$ on each row. Comparisons have led us to favor the hashing trick. It led to slightly better performances. Moreover, through the sparsity of its projection, it is much faster, and as the resulting projected vector has at the most as many non-zero elements as the original one, other components of the forest can be accelerated. In addition, we have explored the impact of the diversity of the projection matrices between the nodes of a same tree T. We have considered four combinations: SxSy, SxDy, DxSy and DxDy where Sx (resp. Sy) is the case where the feature (resp. label) projections are the Same in each node of T and Dx (resp. Dy) is the case where the feature (resp. label) projections are Different from one node to another. Comparisons presented in Section 4.2 have led us to favor SxSy.

Step 2: Instance Partitioning into k Temporary Subsets Let Y_s be the label matrix of a sample drawn without replacement of size at most n_s of the instance set associated to v. The sample is partitioned with a spherical k-means (Loyd's algorithm) applied on Y_sP_y. The cosine metric of the spherical k-means is fast to compute and is well-adapted to sparse data. The cluster centroids are initialized with the k-means++ strategy (Arthur & Vassilvitskii, 2007) which improves cluster stability and algorithm performances against a random initialization.

Step 3: Assigning a Subset from the Projected Features The multi-class classifier is very simple: in each temporary subset, we compute the centroid of the instance projected feature vectors. In the prediction phase, the classifier assigns the subset whose centroid is the closest to the input instance projected feature vector. We have also tested a one-vs-rest linear model but the resulting forest was less accurate and much slower. Two metrics (cosine and classical Euclidean) have been compared too and, for similar reasons as in the k-means, the cosine is better.

4. Algorithm Analysis and Parameter Setting

In this section, we provide bounds for the time and memory complexities of CRAFTML. Then, we analyze the impact of the hyperparameters which govern its performances, and we recommend a parameter setting adapted to XML data.

4.1. Time and Memory Complexities

In the following, we denote by s_x (resp. s_y) the average number of non-zero elements in the feature (resp. label) vectors of the instances. In practice, thanks to the hashing trick, the projected feature and label vectors have less than respectively s_x and s_y non-zero elements on average. For a node v of a tree T, n_v denotes the number of instances of the subset associated to v. The number i of iterations of the spherical k-means is fixed a priori.

Lemma 1. *For a node v of a tree T, the time complexity C_v is bounded by $O\big(n_v \times C\big)$ where $C = k \times (i \times s_y + s_x)$ is the complexity per instance.*

Proof. The time complexity of a node v is the sum of the complexities of the spherical k-means initialized with a k-means++, of the training of the multi-class classifier and of its predictions on the instances of the subset associated to v. The k-means algorithm is applied on the projected labels and its complexity is bounded by the sum of the complexity $O(i \times n_v \times s_y \times k)$ of the Loyd's algorithm and the complexity $O(n_v \times s_y \times k)$ of the k-means++. The classifier training is based on the computation of the feature centroid for each cluster which leads to a total complexity $O(n_v \times s_x)$. For its predictions, the classifier computes the cosine distance to the k centroids for each instance, which requires $O(n_v \times k \times s_x)$ operations. □

In practice, the complexity of the node training is lower than the bound given in Lemma 1. Indeed, the k-means and the classifier training are applied on a sample of the node instance set with a size equal to $\min(n_v, n_s) \leq n_v$. In addition, in our experiments, CRAFTML already reaches its best performances with only $i = 2$ iterations in the k-means.

Let us now consider a strictly k-ary tree T and denote by l_T its number of leaves, by $m_T = \frac{l_T - 1}{k-1}$ its number of nodes and by $\overline{n}_T = \frac{\sum_{v \in T} n_v}{m_T}$ the average number of instances in its nodes.

Proposition 1. *If the tree T is balanced, its training time complexity C_T is bounded by $O\big(log_k(\frac{n}{n_{leaf}}) \times n \times C\big)$. Otherwise, C_T is equal to $O\big(\frac{l_T - 1}{k-1} \times \overline{n}_T \times C\big)$.*

Proof. If T is a balanced tree, the j^{th} level T_j of T has k^j nodes and for each node $v \in T_j$, $n_v = \frac{n}{k^j}$. From Lemma 1, the complexity of the j^{th} level is therefore $O\big(\frac{n}{k^j} \times k^j \times C\big)$ which is independent of j. Due to the stop condition, the number of levels is bounded by $O(log_k(\frac{n}{n_{leaf}}))$. Finally the product of a level complexity with the number of levels gives the complexity C_T for the tree.

If T is an unbalanced tree, the training complexity of T which is the sum of the complexities of its nodes equals to $O\big(\sum_{v \in T} n_v \times C\big)$. Substituting $\sum_{v \in T} n_v$ by $m_T \times \overline{n}_T$ in the last formula ends the proof. □

With the ratio $\frac{C}{k-1} = \frac{k \times (i \times s_y + s_x)}{k-1}$, the contribution of k in the time complexity vanishes for the unbalanced tree. Furthermore, this time depends on the number of leaves l_T and the average number of instance in the nodes $\overline{n}_T$. In practice, l_T is between $\frac{n}{n_{leaf}}$ for the best case and n for the worst case. In our experiments we have observed on the XML datasets that, when fixing $n_{leaf} = 10$ and $k = 2$, we obtain $l_T = \frac{n}{2.83 \pm 0.41}$ and $\overline{n}_T = 24.32 \pm 2.61$.

We can note that by exploiting the data sparsity (s_x and s_y) the time complexity is independent of the projection dimensions d'_x and d'_y. In XML datasets where instances are very sparse, s_x and s_y are much smaller than d_x and d_y. In addition, the complexity of the random projection which is negligible compared to the other operations is not considered here. For instance, in a tree T, the complexity of the fastest combination SxSy chosen in our experiments is equal to $O\big(n \times (s_x + s_y) \times C_{gen}\big)$ where C_{gen} is the complexity of generating a coefficient of the projection matrices[1]. It is also important to stress that, in practice, the bound C_T is above reality due to the instance sampling in each node v which reduces the complexities of the spherical k-means and of the classifier training.

[1] In our experiments, the coefficients are generated with MurmurHash3 (Appleby, 2008) which is among the fastest hash function and which produces good quality random distributions.

Proposition 2. *The memory complexity of a tree T is bounded by $O\big(n \times s_y + m_T \times k \times d'_x\big)$.*

Proof. Each leaf of T stores the mean label vector of its associated instances. In the best case all the $\frac{n}{l_T}$ instances have the same labels and the vector contains s_y non-zero elements on average. In the worst case the $\frac{n}{l_T}$ instances have different labels and the vector approximately contains $\frac{s_y \times n}{l_T}$ non-zero elements on average. Consequently, the memory required for the leaves is bounded by $O\big(l_T \times \frac{s_y \times n}{l_T}\big)$. Each node of T stores a multi-class classifier represented by the k feature vectors of the centroids of dimension d'_x; the required memory is $m_T \times k \times d'_x$. □

In practice, the bound determined in Proposition 2 is significantly greater than the required memory for two reasons. Firstly, the instances regrouped in a same leaf with the clustering share many common labels and the worst case does not occur. Secondly, the centroids stored in the nodes are sparse especially for nodes distant from the root because they are computed on a small subset of similar vectors.

4.2. Performance Analysis

In this section, we explore the respective effect of five hyperparameters (d'_x, d'_y, m_F, n_{leaf}, n_s) on the predictive performances of CRAFTML for five real world datasets from the XML repository[2]: four common datasets from the multi-label learning literature (Bibtex, Mediamill, Delicious, EURLex-4K) with a maximum of 5000 features and 3993 labels and a large one (Wiki10-31K) with 101938 features and 30938 labels (see Table 2 for details). The quality of the results is measured with the precision at 1 (P@1), 3 (P@3) and 5 (P@5) which are defined in the repository and classically used in XML numerical experiments. To avoid test set overfitting here, we restrict ourselves to the training part of the datasets: a validation set with twenty percent of the instances is used for evaluation.

Impact of the Projection Figures 1 and 2 show the impact of the projection dimensions and of the diversity strategies on the performances of CRAFTML with $m_F = 50$. Performances significantly increase with d'_x, less with d'_y and both evolutions reach a plateau which varies with the dataset characteristics. When the two dimensions d'_x and d'_y are large enough (> 500), the effect of the random projections is substancial (comparison with vs without projection in Figure 1). Moreover, in this case the four combinations of presence/absence of node diversity SxSy, SxDy, DxSy and DxDy lead to close behaviors (Figure 2).

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

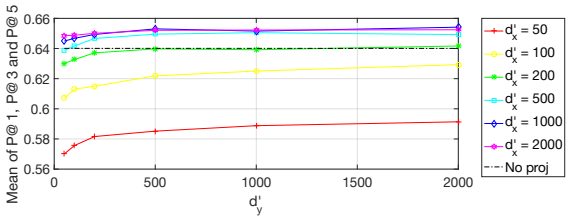

Figure 1. Evolution of the performances of a fifty tree forest (example of EURLex-4K and "SxDy" variation) with respect to d'_y, for six values of d'_x. The charts for the other variations (SxSy,DxSy,DxDy) and other datasets have the same shape. The performance of the forest without random projection is also displayed in a dotted line.

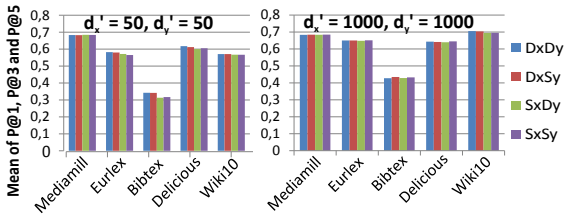

Figure 2. Comparison of the four combinations for node diversity for two values of (d'_x,d'_y) for a fifty tree forest.

Impact of the Number of Trees The contribution of the nodes to the forest memory complexity given by Proposition 2 linearly depends on the product $m_F \times d'_x$. This raises a question: are the best performances obtained with a few trees with a large dimension d'_x or with a large set of trees with a small dimension? Intuitively several trees are required to benefit from the aggregation effect but the dimension must be reasonable to preserve the accuracy of each tree. Figure 3 confirms this intuition. It shows that the optimum is reached for a correct balance between m_F and d'_x whose value varies with the dataset.

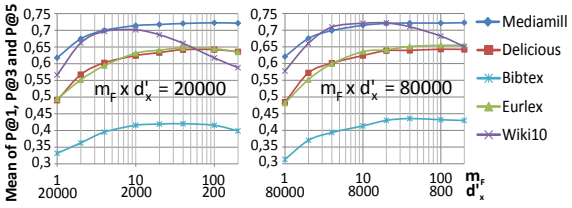

Figure 3. Evolution of the performances with respect to the number of tree at iso-($m_F \times d'_x$).

Impact of the Tree Depth The tree depth is controlled by the stop condition and especially by the minimal number of instances per leaf n_{leaf}. For each tree a large n_{leaf} is required to avoid overfitting the training data (left chart on Figure 4), but for a forest, a limited set of instances per leaf creates more diversity between the trees (right chart on Figure 4). This phenomenon has already been described in the random forest literature (Breiman, 2001).

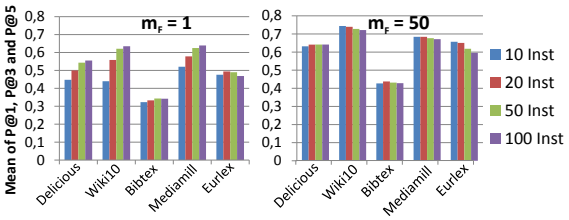

Figure 4. Performances of a tree ($m_F = 1$) and a fifty tree forest ($m_F = 50$) for 4 different values of n_{leaf}. The performances for the single tree are averaged on 30 runs.

Impact of the Sample Size in the Spherical k-means For the dataset with the largest number of instance $n = 1.7M$ (WikiLSHTC-325K) the performances improve with the sample size until they reach a plateau for $n_s = 20000$. The impacts on $P@1$, on $P@3$ and on $P@5$ are almost identical.

4.3. Parameter Setting

Experiments have shown that CRAFTML requires a dozen deep trees and large dimensions ($d'_x > 5000$ and $d'_y > 5000$) for the projected spaces. To limit size effects in the experimental comparisons, the chosen number of trees and stop condition are the same as for FastXML: $m_F = 50$ and $n_{leaf} = 10$ (Prabhu & Varma, 2014). As shown in Section 4.1, the label projection dimension d'_y does not impact the time and memory complexities and it has consequently been fixed to an arbitrary high value: $d'_y = min(d_y, 10000)$. The feature projection dimension d'_x has also no effect on time and a very limited one on memory in practice. CRAFTML reaches the plateau of performances for each dataset for a sample size $n_s = 20000$ and a dimension $d'_x = min(d_x, 10000)$. And its measured model size is low (Table 3). Moreover, for the chosen values of m_F, n_{leaf}, d'_x and d'_y, CRAFTML obtains close performances with the four variations SxSy, SxDy, DxSy and DxDy. We use SxSy which is the fastest combination.

5. Experimental Comparisons

In this section we compare CRAFTML with the nine best methods of the state-of-the-art in extreme multi-label learning presented in Section 2: four tree-based methods (FastXML, PFastReXML, LPSR, PLT) and five others with a distinction between those adapted to single-core

Table 2. Comparison between CRAFTML and the state-of-the-art methods on the test set of classical eXtreme Multi-label datasets. The label dimension d_y, the feature dimension d_x, the number of instances of the training set n and the number of instances in the test set n_{test} are explicited. The best result among tree-based methods is in bold font. The best result among all methods is underlined.

		Tree-based					Other				
		CRAFTML	PFastReXML	FastXML	LPSR	PLT	SLEEC	AnnexML	PDSparse	DISMEC	PPDSparse
Mediamill	P@1	**85.86**	83.98	84.22	83.57		87.82	-	83.64	84.83	84.42
$d_x = 120$, $d_y = 101$	P@3	**69.01**	67.37	67.33	65.78	-	73.45	-	66.13	67.17	67.26
$n = 30993$, $n_{test} = 12914$	P@5	**54.65**	53.02	53.04	49.97	-	59.17	-	50.90	52.80	52.78
Bibtex	P@1	**65.15**	63.46	63.42	62.11	-	65.08	-	62.36	63.69	63.69
$d_x = 1836$, $d_y = 159$	P@3	**39.83**	39.22	39.23	36.65	-	39.64	-	36.50	38.80	39.43
$n = 4880$, $n_{test} = 2515$	P@5	28.99	**29.14**	28.86	26.53	-	28.87	-	26.50	28.30	28.67
Delicious	P@1	**70.26**	67.13	69.61	65.01	-	67.59	-	-	-	-
$d_x = 500$, $d_y = 983$	P@3	63.98	62.33	**64.12**	58.96	-	61.38	-	-	-	-
$n = 12920$, $n_{test} = 3185$	P@5	59.00	58.62	**59.27**	53.49	-	56.56	-	-	-	-
EURLex-4K	P@1	**78.81**	75.45	71.36	76.37	-	79.26	-	75.90	82.40	74.61
$d_x = 5000$, $d_y = 3993$	P@3	**65.21**	62.70	59.90	63.36	-	64.30	-	61.16	68.50	59.56
$n = 15539$, $n_{test} = 3809$	P@5	**53.71**	52.51	50.39	52.03	-	52.33	-	50.83	57.70	48.43
Wiki10-31K	P@1	**85.19**	83.57	83.03	72.72	84.34	85.88	86.50	-	85.20	-
$d_x = 101938$, $d_y = 30938$	P@3	**73.17**	68.61	67.47	58.51	72.34	72.98	74.28	-	74.60	-
$n = 14146$, $n_{test} = 6616$	P@5	**63.27**	59.10	57.76	49.50	62.72	62.70	64.19	-	65.90	-
WikiLSHTC-325K	P@1	**56.57**	56.05	49.75	27.44	45.67	54.83	63.36	60.70	64.40	64.13
$d_x = 1617899$, $d_y = 325056$	P@3	34.73	**36.79**	33.10	16.23	29.13	33.42	40.66	39.62	42.50	42.10
$n = 1778351$, $n_{test} = 587084$	P@5	25.03	**27.09**	24.45	11.77	21.95	23.85	29.79	29.20	31.50	31.14
Delicious-200K	P@1	**47.87**	41.72	43.07	18.59	45.37	47.85	46.66	37.69	45.50	45.05
$d_x = 782585$, $d_y = 205443$	P@3	**41.28**	37.83	38.66	15.43	38.94	42.21	40.79	30.16	38.70	38.34
$n = 196606$, $n_{test} = 100095$	P@5	**38.01**	35.58	36.19	14.07	35.88	39.43	37.64	27.01	35.50	34.9
Amazon-670K	P@1	37.35	**39.46**	36.99	28.65	36.65	35.05	42.08	-	44.70	43.04
$d_x = 135909$, $d_y = 670091$	P@3	33.31	**35.81**	33.28	24.88	32.12	31.25	36.65	-	39.70	38.24
$n = 490449$, $n_{test} = 153025$	P@5	30.62	**33.05**	30.53	22.37	28.85	28.56	32.76	-	36.10	34.94
AmazonCat-13K	P@1	92.78	91.75	**93.11**	-	91.47	90.53	93.55	87.43	93.40	92.72
$d_x = 203882$, $d_y = 13330$	P@3	**78.48**	77.97	78.20	-	75.84	76.33	78.38	70.48	79.10	78.14
$n = 1186239$, $n_{test} = 306782$	P@5	63.58	**63.68**	63.41	-	61.02	61.52	63.32	56.70	64.10	63.41

machines (SLEEC, AnnexML, PDSparse) and those especially designed for parallel implementations (DISMEC, PPDSparse). The numerical experiments are carried on nine XML datasets from different application domains: Bibtex, Mediamill, Delicious, EURLex-4K, Wiki10-31K, Delicious-200K, AmazonCat-13K, WikiLSHTC-325K, Amazon-670K. The instance, feature and label cardinalities are reported in the first column of Table 2 and additional details are available in the XML repository. Results considered for the comparisons are extracted from the last published results [3]. The hyperparameters chosen for CRAFTML are those described in the previous section. Table 2 presents the results of the predictive performances and Table 3 the time consumption and the model size.

5.1. Comparison with Tree-based Methods

CRAFTML outperforms the best tree-based methods in most cases. For the datasets WikiLSHTC-325K and Amazon-670K, the domination of PFastreXML may be partly explained by the fact that it is trained with label propensities that are computed with additional external information (label hierarchy of Wikipedia and Amazon) (Jain et al., 2016). Comparisons of computing times are sensitive as the approaches have been developed with different languages (Java for CRAFTML) and times have been measured on different machines. Nevertheless Table 3 confirms that CRAFTML is very competitive. Compared with the other tree-based methods its observed training time is lower on average and its model size is smaller. These measures are consistent with the theoretical results: due to the sampling strategy and the dimensionality reduction resulting from the random projections, the training time and memory complexities of CRAFTML are the lowest ones. Its predictive time is higher than the others but its complexity is equivalent.

5.2. Comparison with other Single-core Implementations (SLEEC, PDSparse, AnnexML)

The performances of CRAFTML are better than those of PDSparse except for WikiLSHTC-325K. They are substantially equivalent to those of SLEEC but there is a slight domination of CRAFTML on the four largest datasets. Moreover, CRAFTML is faster than SLEEC and than PDSparse except for the dataset AmazonCat-13K. The specificities of this dataset -i.e. a small number of labels and a large number of instances- favor PDSparse. The CRAFTML model size is lower than that of SLEEC -except for WikiLSHTC-325K-, but it is greater than that of PDSparse: 1.2 times for EURLex-4K, 1.98 times for WikiLSHTC-325K, 93 times

[3] from the XML repository for PFastReXML, FastXML, LPSR-NB, SLEEC, and DISMEC, from (Jasinska et al., 2016) for PLT, from (Yen et al., 2017) for PDSparse, PPDSparse and the remaining results of DISMEC, and from (Tagami, 2017) for AnnexML

Table 3. Training time, prediction time and model sizes of the compared algorithms on large datasets. Values for FastXML, PFastReXML, SLEEC, PDSparse, DISMEC and PPDSparse are extracted from (Yen et al., 2017) and the same conditions (RAM, CPU) are fixed for our measurements. For CRAFTML's forest, the training time in parenthesis is measured on a five-core parallelization.

Machine		1 core						100 cores	
Language		Java		C++				C++	
Algorithm		CRAFTML Forest	CRAFTML Tree (Forest/50)	FastXML	PFastReXML	SLEEC	PDSparse	DISMEC	PPDSparse
EURLex-4K	Train (s)	196 (47.75)	3.92	315.9	324.4	4543.4	773.2	76.07	9.95
	Test (ms)	5.39	0.1078	3.65	5.43	3.67	0.73	2.26	1.5
	Model size (Mb)	30	0.6	384	455	121	25	15	9.5
WikiLSHTC-325K	Train (s)	18508 (5092)	370.16	19160	20070	39000	94343	271407	353
	Test (ms)	7.67	0.1534	1.02	1.47	4.85	3.89	65	290
	Model size (Gb)	1.06	0.021	14	16	0.635	0.534	8.1	4.9
Delicious-200K	Train (s)	4754 (1174)	95.08	8832.46	8807.51	4838.7	5137.4	38814	2869
	Test (ms)	8.6	0.172	1.28	7.4	2.685	0.432	311.4	275
	Model size (Gb)	0.346	0.007	1.3	20	2.1	0.004	18	9.4
Amazon-670K	Train (s)	5653 (1487)	113.06	5624	6559	20904	-	174135	921.9
	Test (ms)	5.02	0.1004	1.41	1.98	6.94	-	148	20
	Model size (Gb)	0.494	0.010	4.0	6.3	6.6	-	8.1	5.3
AmazonCat-13K	Train (s)	10606 (2876)	212.12	11535	13985	119840	2789	11828	122.8
	Test (ms)	5.12	0.1024	1.21	1.34	13.36	0.87	0.2	1.82
	Model size (Gb)	0.659	0.013	9.7	11	12	0.015	2.1	0.347

for Delicious-200K and 45 times for Amazon-13K. With a threshold applied on its parameter after training, the final PDSparse model size is very low, but PDSparse requires a large amount of memory during training; e.g. it cannot be trained on the dataset Amazon-670K with 100Gb of memory (Yen et al., 2017). The comparison with AnnexML is more sensitive as its training time and model size are not published for a single-core implementation. The published performances show that CRAFTML is close to AnnexML except for Amazon-670K and WikiLSHTC-325K. But for the latter, the training time of AnnexML (4 hours) only mentioned for a 24-core implementation suggests that CRAFTML is faster (1.5 hour on a 5-core machine).

5.3. Comparison with Parallel Implementations (DISMEC, PPDSparse)

The results of the linear models DISMEC and PPDSparse reported in Tables 2 and 3 have been obtained on a one hundred-core machine. Let us recall that the DISMEC model has been specifically designed for parallelization and is inapplicable on a single-core machine. When comparing performances of these two approaches with those obtained by CRAFTML on a single-core machine the conclusions are mixed and depend on the dataset. The datasets WikiLSHTC-325K and Amazon-670K seem to favor the two linear model based approaches over every tree-based methods. For most datasets, the size of CRAFTML model is lower than that of DISMEC and PPDSparse. Surprisingly, the prediction time of CRAFTML obtained on a single-core machine is often lower than those of DISMEC and PPDSparse. Its training time is also lower than that of DISMEC for the large datasets and similar for the smallest ones but higher than that of PPDSparse. In addition, we have measured the time gains of CRAFTML with a five-core machine. In this case, the training time of CRAFTML is lower than that of DISMEC for all the datasets and that of PPDSparse for Delicious-200K, and it becomes close to the training time of PPDSparse for Amazon-670K. Consequently, even with five cores only, CRAFTML is competitive with the best parallelized approaches. Most importantly, its acceleration factor of about four between a single-core and a five-core implementation and its lower training/prediction time complexity allows us to hope to be as fast or even faster than PPDSparse on a comparable supercomputer.

6. Conclusion

Our new XML multi-label learning method CRAFTML outperforms the other tree-based methods with a single core computer and it is competitive with the state-of-the-art PPDSparse even with a restricted parallel implementation. Contrary to most of the current XML methods, CRAFTML does not rely on a complex optimization scheme. It combines simple and fast learning blocks (e.g. k-mean clustering, basic multi-class classifier) which leaves room for extensions to reach performances required by current societal and technical challenges (Kambatla et al., 2014). With the growing dimensionality of data, machine learning increasingly resorts to supercomputers. But this access is far from being available everywhere today and its cost will set limits in the future. Consequently, cheap and scalable machine learning algorithms are required to favor the democratization of the numerous real-world applications which still rely on standard computation. However, cloud computing (Hashem et al., 2015) and the increasing development of supercomputers (Dean et al., 2018) also need methods that fully exploit the available computation resources by being, in particular, easily parallelizable.

Acknowledgements

We thank the reviewers whose comments helped us to improve and clarify this manuscript.

References

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. 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*, pp. 13–24. ACM, 2013.

Appleby, A. Murmurhash 2.0 - https://sites.google.com/site/murmurhash/, 2008.

Arthur, D. and Vassilvitskii, S. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027–1035. Society for Industrial and Applied Mathematics, 2007.

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

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

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. Sparse local embeddings for extreme multi-label classification. In *Advances in neural information processing systems*, pp. 730–738, 2015.

Breiman, L. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Dean, J., Patterson, D., and Young, C. A new golden age in computer architecture: Empowering the machine learning revolution. *IEEE Micro*, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition.*, pp. 248–255. IEEE, 2009.

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

Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Khan, S. U. The rise of big data on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

Jain, H., Prabhu, Y., and Varma, M. 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*, pp. 935–944. ACM, 2016.

Jasinska, K., Dembczynski, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., and Hullermeier, E. Extreme f-measure maximization using sparse probability estimates. In *International Conference on Machine Learning*, pp. 1435–1444, 2016.

Kambatla, K., Kollias, G., Kumar, V., and Grama, A. Trends in big data analytics. *Journal of Parallel and Distributed Computing*, 74(7):2561–2573, 2014.

Kocev, D., Vens, C., Struyf, J., and Džeroski, S. Ensembles of multi-objective decision trees. *Machine Learning: ECML 2007*, pp. 624–631, 2007.

McAuley, J., Pandey, R., and Leskovec, J. Inferring networks of substitutable and complementary products. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794. ACM, 2015.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R., and Galinari, P. Lshtc: A benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*, 2015.

Prabhu, Y. and Varma, M. 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*, pp. 263–272. ACM, 2014.

Tagami, Y. 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*, pp. 455–464. ACM, 2017.

Tsoumakas, G. and Katakis, I. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3), 2006.

Tsoumakas, G., Katakis, I., and Vlahavas, I. Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD08)*, pp. 30–44, 2008.

Weinberger, K., Dasgupta, A., Langford, J., Smola, A., and Attenberg, J. Feature hashing for large scale multitask learning. In *International Conference on Machine Learning*, pp. 1113–1120. ACM, 2009.

Weston, J., Bengio, S., and Usunier, N. Wsabie: Scaling up to large vocabulary image annotation. In *International Joint Conference on Artificial Intelligence*, volume 11, pp. 2764–2770, 2011.

Weston, J., Makadia, A., and Yee, H. Label partitioning for sublinear ranking. In *International Conference on Machine Learning*, pp. 181–189, 2013.

Yen, I. E., Huang, X., Dai, W., Ravikumar, P., Dhillon, I., and Xing, E. 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*, pp. 545–553. ACM, 2017.

Yen, I. E.-H., Huang, X., Ravikumar, P., Zhong, K., and Dhillon, I. Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pp. 3069–3077, 2016.

Yu, H.-F., Jain, P., Kar, P., and Dhillon, I. Large-scale multi-label learning with missing labels. In *International Conference on Machine Learning*, pp. 593–601, 2014.

Zhang, M.-L. and Zhou, Z.-H. Ml-knn: A lazy learning approach to multi-label learning. *Pattern recognition*, 40 (7):2038–2048, 2007.

Zhang, M.-L. and Zhou, Z.-H. A review on multi-label learning algorithms. *IEEE transactions on knowledge and data engineering*, 26(8):1819–1837, 2014.