# The Weight Function in the Subtree Kernel is Decisive

Romain Azaïs, Florian Ingels

# THE WEIGHT FUNCTION IN THE SUBTREE KERNEL IS DECISIVE

Romain Azaïs
romain.azais@inria.fr

Florian Ingels
florian.ingels@inria.fr

Laboratoire Reproduction et Développement des Plantes, Univ Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRA, Inria, F-69342, Lyon, France

**Abstract**

Tree data are ubiquitous because they model a large variety of situations, e.g., the architecture of plants, the secondary structure of RNA, or the hierarchy of XML files. Nevertheless, the analysis of these non-Euclidean data is difficult per se. In this paper, we focus on the subtree kernel that is a convolution kernel for tree data introduced by Vishwanathan and Smola in the early 2000's. More precisely, we investigate the influence of the weight function from a theoretical perspective and in real data applications. We establish on a 2-classes stochastic model that the performance of the subtree kernel is improved when the weight of leaves vanishes, which motivates the definition of a new weight function, learned from the data and not fixed by the user as usually done. To this end, we define a unified framework for computing the subtree kernel from ordered or unordered trees, that is particularly suitable for tuning parameters. We show through eight real data classification problems the great efficiency of our approach, in particular for small datasets, which also states the high importance of the weight function. Finally, a visualization tool of the significant features is derived.

**keywords:** classification of tree data; kernel methods; subtree kernel; weight function; tree compression

## 1 Introduction

### 1.1 Analysis of tree data

Tree data naturally appear in a wide range of scientific fields, from RNA secondary structures in biology [22] to `XML` files [8] in computer science through dendrimers [23] in chemistry and physics. Consequently, the statistical analysis of tree data is of great interest. Nevertheless, investigating these data is difficult due to the intrinsic non-Euclidean nature of trees.

Several approaches have been considered in the literature to deal with this kind of data: edit distances between unordered or ordered trees (see [6] and the references therein), coding processes for ordered trees [27], with a special focus on conditioned Galton-Watson trees [3, 5]. One can also mention the approach developed in [31]. In the present paper, we focus on kernel methods, a complementary family of techniques that are well-adapted to non-Euclidean data.

Kernel methods consists in mapping the original data into a (inner product) feature space. Choosing the proper feature space and finding out the mapping might be very difficult. Furthermore, the curse of dimensionality takes place and the feature space may be extremely big, therefore impossible to use. Fortunately, a wide range of prediction algorithms do not need to access that

feature space, but only the inner product between elements of the feature space. Building a function, called a kernel, that simulates an inner product in an implicit feature space, frees us from constructing a mapping. Indeed, $K : \mathcal{X}^2 \to \mathbb{R}$ is said to be a kernel function on $\mathcal{X}$ if, for any $(x_1, \dots, x_n) \in \mathcal{X}^n$, the Gram matrix $[K(x_i, x_j)]_{1 \leq i,j \leq n}$ is positive semidefinite. By virtue of Mercer's theorem [24], there exists a (inner product) feature space $\mathcal{Y}$ and a mapping $\varphi : \mathcal{X} \to \mathcal{Y}$ such that, for any $(x, y) \in \mathcal{X}^2$, $K(x, y) = \langle \varphi(x), \varphi(y) \rangle_{\mathcal{Y}}$. This technique is known as the kernel trick. Algorithms that can use kernels include Support Vector Machines (SVM), Principal Components Analyses (PCA) and many others. We refer the reader to the books [9, 25, 26] and the references therein for more detailed explanations of theory and applications of kernels.

To use kernel-based algorithms with tree data, one needs to design kernel functions adapted to trees. Convolution kernels, introduced by Haussler [18], measure the similarity between two complex combinatorial objects based on the similarity of their substructures. Based on this strategy, many authors have developed convolution kernels for trees, among them the subset tree kernel [7], the subtree kernel [30] and the subpath kernel [21]. A recent state-of-the-art on kernels for trees can be found in the thesis of Da San Martino [10], as well as original contributions on related topics. In this article, we focus on the subtree kernel as defined in [30]. In this introduction, we develop some concepts on trees in Subsection 1.2. They are required to deal with the precise definition of the subtree kernel in Subsection 1.3 as well as the aim of the paper presented in Subsection 1.4.

### 1.2 Unordered and ordered rooted trees

**Rooted trees** A rooted tree $T$ is a connected graph with no cycle such that there exists a unique vertex $\mathcal{R}(T)$, called the root, which has no parent, and any vertex different from the root has exactly one parent. The leaves $\mathcal{L}(T)$ are all the vertices without children. The height of a vertex $v$ of a tree $T$ can be recursively defined as $\mathcal{H}(v) = 0$ if $v$ is a leaf of $T$ and

$$\mathcal{H}(v) = 1 + \max_{w \in \mathcal{C}(v)} \mathcal{H}(w)$$

otherwise. The height $\mathcal{H}(T)$ of the tree $T$ is defined as the height of its root, i.e., $\mathcal{H}(T) = \mathcal{H}(\mathcal{R}(T))$. The outdegree of $T$ is the maximal branching factor that can be found in $T$, that is

$$\deg(T) = \max_{v \in T} \#\mathcal{C}(v),$$

where $\mathcal{C}(v)$ denotes the set of children of $v$. For any vertex $v$ of $T$, the subtree $T[v]$ rooted in $v$ is the tree composed of $v$ and all its descendants $\mathcal{D}(v)$. $\mathcal{S}(T)$ denotes the set of subtrees of $T$.

**Unordered trees** Rooted trees are said unordered if the order between the sibling vertices of any vertex is not significant. The precise definition of unordered rooted trees, or simply unordered trees, is obtained from the following equivalence relation: two trees $T_1$ and $T_2$ are isomorphic (as unordered trees) if there exists a one-to-one correspondence $\Phi$ from the set of vertices of $T_1$ into the set of vertices of $T_2$ such that, if $w$ is a child of $v$ in $T_1$, then $\Phi(w)$ is a child of $\Phi(v)$ in $T_2$. The set of unordered trees is the quotient set of rooted trees by this equivalence relation.

**Ordered trees** In ordered rooted trees, or simply ordered trees, the set of children of any vertex is ordered. As before, ordered trees can be defined as a quotient set if one adds the concept of order to the equivalence relation: two trees $T_1$ and $T_2$ are isomorphic (as ordered trees) if there exists a one-to-one correspondence $\Phi$ from the set of vertices of $T_1$ into the set of vertices of $T_2$ such that, if $w$ is the $r^{\text{th}}$ child of $v$ in $T_1$, then $\Phi(w)$ is the $r^{\text{th}}$ child of $\Phi(v)$ in $T_2$.

In the whole paper, $\mathcal{T}^*$ denotes the set of $*$-trees with $* \in \{\text{ordered}, \text{unordered}\}$.

### 1.3 Subtree kernel

The subtree kernel has been introduced in [30] as a convolution kernel on trees for which the similarity between two trees is measured through the similarity of their subtrees. A subtree kernel $K$ on $*$-trees is defined as,

$$\forall\, T_1, T_2 \in \mathcal{T}^*,\ K(T_1, T_2) = \sum_{\tau \in \mathcal{T}^*} w_\tau\, \kappa\left(N_\tau(T_1), N_\tau(T_2)\right), \tag{1}$$

where $w_\tau$ is the weight associated to $\tau$, $N_\tau(T)$ counts the number of subtrees of $T$ that are isomorphic (as $*$-trees) to $\tau$ and $\kappa$ is a kernel function on $\mathbb{N}$, $\mathbb{Z}$ or $\mathbb{R}$ (see [25, Section 2.3] for some classic examples). Assuming $\kappa(0, n) = \kappa(n, 0) = 0$, the formula (1) of $K$ becomes

$$K(T_1, T_2) = \sum_{\tau \in \mathcal{S}(T_1) \cap \mathcal{S}(T_2)} w_\tau\, \kappa\left(N_\tau(T_1), N_\tau(T_2)\right),$$

making the sum finite. Indeed, all the subtrees $\tau \in \mathcal{T}^* \setminus \mathcal{S}(T_1) \cap \mathcal{S}(T_2)$ do not count in the sum (1). In this paper, as in [30], we assume that $\kappa(n, m) = nm$, then

$$K(T_1, T_2) = \sum_{\tau \in \mathcal{S}(T_1) \cap \mathcal{S}(T_2)} w_\tau\, N_\tau(T_1) N_\tau(T_2). \tag{2}$$

which is the subtree kernel as introduced in [30].

The weight function $\tau \mapsto w_\tau$ is the only parameter to be tuned. In the literature, the weight is always assumed to be a function of a quantity measuring the "size" of $\tau$, in particular its height $\mathcal{H}(\tau)$. Then $w_\tau$ is taken as an exponential decay of this quantity, $w_\tau = \lambda^{\mathcal{H}(\tau)}$ for some $\lambda \in [0, 1]$ [1, 7, 10, 21, 30]. This choice can be justified in the following manner. If a subtree $\tau$ is counted in the kernel, then all its subtrees are also counted. Then an exponential decay counterbalances the exponential growth of the number of subtrees.

In the literature, two algorithms have been proposed to compute the subtree kernel for ordered trees. The approach of [30] is based on string representations of trees, while the authors of [1, 10] extensively use DAG reduction of tree data, an algorithm that achieves lossly compression of trees. To the best of our knowledge, the case of unordered trees has only been considered through the arbitrary choice of a sibling order.

### 1.4 Aim of the paper

The aim of the present paper is threefold:

1. We investigate the theoretical properties of the subtree kernel on a 2-classes model of random trees in Section 2. More precisely, we provide a lower-bound for the contrast of the kernel in Proposition 2.2. Indeed, the higher the contrast, the less data are required to achieve a given performance in prediction (see [4] for general similarity functions and Corollary 2.3 for the subtree kernel). We exploit this result to show in Subsection 2.4 that the contrast of the subtree kernel is improved if the weight of leaves vanishes. The relevance of the model is discussed in Remark 2.1.

2. We rely on [1, 10] on ordered trees to develop in Section 3 a unified framework based on DAG reduction for computing the subtree kernel from ordered or unordered trees, with or without labels on their vertices. Subsection 3.1 is devoted to DAG reduction of unordered then ordered trees. DAG reduction of a forest is introduced in Subsection 3.2. Then, the subtree kernel is computed from the annotated DAG reduction of the dataset is Subsection 3.3. We notice in Remark 3.8 that DAG reduction of the dataset is costly but makes possible superfast repeated computations of the kernel, which is particularly adapted for tuning parameters. This is the main advantage of the DAG computation of the subtree kernel compared to the algorithm based on string representations [30]. Our method allows the implementation of any weighting function, while the recursive computation of the subtree kernel proposed in [10, Chapter 6] also uses DAG reduction of tree data but makes an extensive use of the exponential form of the weight (combining equations (3.12) and (6.2) from [10]). We also investigate the theoretical complexities of the different steps of the DAG computation for both ordered and unordered trees (see Proposition 3.2 and Remark 3.8). This kind of question has been tackled in the literature only for ordered trees and from a numerical perspective [1, Section 4].

3. As aforementioned, we show in the context of a stochastic model that the performance of the subtree kernel is improved when the weight of leaves is 0 (see Section 2). Relying on this (see also Remark 2.5 on the possible generalization of this result), we define in Section 4 a new weight function, called discriminance, that is not a function of the size of the argument as in the literature, but is learned from the data. The learning step of the discriminance weight function strongly relies on the DAG computation of the subtree kernel presented above because it allows the enumeration of all the subtrees composing the dataset without redundancies. We explore in Section 5 the relevance of this new weighting scheme across several datasets, notably on the difficult prediction problem of the language of a Wikipedia article from its structure in Subsection 5.2. Beyond very good classification results, we show that the methodology developed in the paper can be used to extract the significant features of the problem and provide a visualization at a glance of the dataset. In addition, we remark that the average discriminance weight decreases exponentially as a function of the height (except for leaves). Thus, the discriminance weight can be interpreted as the second order of the exponential weight introduced in the literature. Application to real-world datasets in Subsections 5.3, 5.4 and 5.5 shows that the discriminance weight is particularly relevant for small databases when the classification problem is rather difficult, as depicted in Fig. 18.

Finally, concluding remarks are presented in Section 6. Technical proofs have been deferred into Appendices A and B.

## 2 Theoretical study

In this section, we define a stochastic model of 2-classes tree data. From this ideal dataset, we prove the efficiency of the subtree kernel and derive the sufficient size of the training dataset to get a classifier with a given prediction error. We also state on this simple model that the weight of leaves should always be 0. We emphasize that this study is valid for both ordered and unordered trees.

### 2.1 Two trees as different as possible

Our goal is to build a 2-classes dataset of random trees. To this end, we first define two typical trees $T_0$ and $T_1$ that are as different as possible in terms of subtree kernel.

Let $T_0$ and $T_1$ be two trees that fulfill the following conditions:

1. $\forall i \in \{0,1\}$, $\forall u, v \in T_i \setminus \mathcal{L}(T_i)$, if $u \neq v$ then $T_i[u] \neq T_i[v]$, i.e, two subtrees of $T_i$ are not isomorphic (except leaves).
2. $\forall u \in T_0 \setminus \mathcal{L}(T_0)$, $\forall v \in T_1 \setminus \mathcal{L}(T_1)$, $T_0[u] \neq T_1[v]$, i.e., any subtree of $T_0$ is not isomorphic to a subtree of $T_1$ (except leaves).

These two assumptions ensure that the trees $T_0$ and $T_1$ are as different as possible. Indeed, it is easy to see that

$$K(T_0, T_1) = w_\bullet \# \mathcal{L}(T_0) \# \mathcal{L}(T_1),$$

which is the minimal value of the kernel and where $\omega_\bullet$ is the weight of leaves. We refer to Fig. 1 for an example of trees that satisfy these conditions.

Figure 1: Two trees $T_0$ and $T_1$ that fulfill conditions 1 and 2.

Trees of class $i$ will be obtained as random editions of $T_i$. In the sequel, $T_i(v \mapsto \tau)$ denotes the tree $T_i$ in which the subtree rooted at $u$ has been replaced by $\tau$. These random edits will tend to make trees of class 0 closer to trees of class 1. To this end, we introduce the following additional assumption. Let $(\tau_h)$ a sequence of trees such that $\mathcal{H}(\tau_h) = h$.

3. Let $u \in T_0$ and $v \in T_1$. We consider the edited trees $T_0' = T_0(u \mapsto \tau_{\mathcal{H}(u)})$ and $T_1' = T_1(v \mapsto \tau_{\mathcal{H}(v)})$. Then, $\forall u' \in T_0' \setminus \left(\tau_{\mathcal{H}(u)} \cup \mathcal{L}(T_0')\right)$, $\forall v' \in T_1' \setminus \left(\tau_{\mathcal{H}(v)} \cup \mathcal{L}(T_1')\right)$, $T_0'[u'] \neq T_1'[v']$.

In other words, if one replaces subtrees of $T_0$ and $T_1$ by subtrees of the same height, then any subtree of $T_0$ is not isomorphic to a subtree of $T_1$ (except the new subtrees and leaves). This means that the similarity between random edits of $T_0$ and $T_1$ will come only from the new subtrees and not from collateral modifications. We refer to Fig. 2 for an example of trees that satisfy these conditions.

Figure 2: Two trees $T_0$ and $T_1$ that fulfill conditions 1, 2 and 3.

### 2.2 A stochastic model of 2-classes tree data

From now on, we assume that, for any $h > 0$, $\tau_h$ is not a subtree of $T_0$ nor $T_1$. For the sake of simplicity, $T_0$ and $T_1$ have the same height $H$. In addition, if $u \in T_i$ then $T_i^u$ denotes $T_i(u \mapsto \tau_{\mathcal{H}(u)})$.

The stochastic model of 2-classes tree data that we consider is defined from the binomial distribution $P_\rho = \mathcal{B}(H, \rho/H)$ on support $\{0, \dots, H\}$ with mean $\overline{P}_\rho = \rho$. The parameter $\rho \in [0, H]$ is fixed. In the dataset, class $i$ is composed of random trees $T_i^u$, where the vertex $u$ has been picked uniformly at random among vertices of height $h$ in $T_i$, where $h$ follows $P_\rho$. Furthermore, the considered training dataset is well-balanced in the sense that it contains the same number of data of each class.

Intuitively, when $\rho$ increases, the trees are more degraded and thus two trees of different class are closer. $\rho$ somehow measures the similarity between the two classes. In other words, the larger $\rho$, the more difficult is the supervised classification problem.

**Remark 2.1.** *The structure of a markup document such as an* `HTML` *page can be described by a tree (see Subsection 5.1 and Fig. 6 for more details). In this context, the tree $T_i$, $i \in \{0, 1\}$, can be seen as a model of the structure of a webpage template. By assumption, the two templates of interest are as different as possible. However, they are completed in a similar manner, for example to present the same content in two different layouts. Edition of the templates is modeled by random edit operations. They tend to bring trees from different templates closer.*

### 2.3 Theoretical guarantees on the subtree kernel

The authors of [4] have introduced a theory that describes the effectiveness of a given kernel in terms of similarity-based properties. A similarity function over $\mathcal{X}$ is a pairwise function $K : \mathcal{X}^2 \to [-1, 1]$ [4, Definition 1]. It is said $(\epsilon, \gamma)$-strongly good [4, Definition 4] if, with probability at most $1 - \epsilon$,

$$\mathbb{E}_{x',y}[K(x, x') - K(x, y)] \geq \gamma,$$

where $\text{label}(x) = \text{label}(x') \neq \text{label}(y)$. From this definition, the authors derive the following simple classifier: the class of a new data $x$ is predicted by 1 if $x$ is more similar on average to points in class 1 than to points in class 0, and 0 otherwise. In addition, they prove [4, Theorem 1] that a well-balanced training dataset of size $32/\gamma^2 \log(2/\delta)$ is sufficient so that, with probability at least $1 - \delta$, the above algorithm applied to an $(\epsilon, \gamma)$-strongly good similarity function produces a classifier with error at most $\epsilon + \delta$.

We aim to prove comparable results for the subtree kernel that is not a similarity function. To this end, we focus for $i \in \{0, 1\}$ on

$$\Delta_x^i = \mathbb{E}_{u,v}[K(T_i^x, T_i^u) - K(T_i^x, T_{1-i}^v)]. \tag{3}$$

We emphasize that the two following results (Proposition 2.2 and Corollary 2.3) assume that the weight of leaves $\omega_\bullet$ is 0. For the sake of readability, we introduce the following notations, for any $0 \leq h \leq H$ and $i \in \{0, 1\}$,

$$\begin{aligned} K_{i,h} &= \max_{\{u \in T_i \,:\, \mathcal{H}(u)=h\}} K(T_i[u], T_i[u]), \\ C_{i,h} &= \frac{K(T_i, T_i) - K_{i,h}}{\#\mathcal{L}(T_i)}, \\ G_\rho(h) &= 1 - \sum_{k=h+1}^{H} P_\rho(k). \end{aligned}$$

The following results are expressed in terms of a parameter $0 \leq h < H$. The statement is then true with probability $G_\rho(h)$. This is equivalent to state a result that is true with probability $1 - \epsilon$, for any $\epsilon > 0$.

**Proposition 2.2.** *If $w_{T_i} > 0$ then $\Delta_x^i = 0$ if and only if $x = \mathcal{R}(T_i)$. In addition, if $\rho > H/2$, for any $0 \leq h < H$, with probability $G_\rho(h)$, one has*

$$\Delta_x^i \geq P_\rho(0) C_{i,h}. \tag{4}$$

**Proof.** The proof lies in Appendix A. ❧

This result shows that the two classes can be well-separated by the subtree kernel. The only data that can not be separated are the trees completely edited. In addition, the lower-bound in (4) is of order $H \exp(-\rho)$ (up to a multiplicative constant).

**Corollary 2.3.** *For any $0 \leq h \leq H$, a well-balanced training dataset of size*

$$\frac{2 \max_i K(T_i, T_i)^2}{\min_i C_{i,h}^2} \frac{\exp(2\rho)}{H^2} \log\left(\frac{2}{\delta}\right)$$

*is sufficient so that, with probability at least $1 - \delta$, the aforementioned classification algorithm produces a classifier with error at most $1 - G_\rho(h) + \delta$.*

**Proof.** The proof is based on the demonstration of [4, Theorem 1]. However, in our setting, the kernel $K$ is bounded by $\max_i K(T_i, T_i)$ and not by 1. Consequently, by Hoeffding bounds, the sufficient size of the training dataset if of order

$$2 \log\left(\frac{2}{\delta}\right) \frac{\max_i K(T_i, T_i)^2}{\gamma^2}, \tag{5}$$

where $\gamma$ can be read in Proposition 2.2, $\gamma = P_\rho(0) C_{i,h} \geq P_\rho(0) \min_i C_{i,h}$. The coefficient 2 lies because we consider here the total size of the dataset and not only the number of examples of each class. Together with $P_\rho(0) \sim H \exp(-\rho)$, we obtain the expected result. ❧

### 2.4 Weight of leaves

Here $K^+$ is the subtree kernel obtained from the weights used in the computation of $K$ together with a positive weight on leaves, $w_\bullet > 0$. We aim to show that $K^+$ separates the two classes less than $K$. $\Delta_x^{+,i}$ denotes the conditional expectation (3) computed from $K^+$.

**Proposition 2.4.** *For any* $x \in T_i$,

$$\Delta_x^{+,i} = \Delta_x^i + w_\bullet \# \mathcal{L}(T_i[x]) D_{i,1-i},$$

*where* $D_{i,1-i} = \mathbb{E}_{u,v}[\# \mathcal{L}(T_i^u) - \# \mathcal{L}(T_{1-i}^v)]$.

**Proof.** We have the following decomposition, for any trees $T_1$ and $T_2$,

$$K^+(T_1, T_2) = K(T_1, T_2) + w_\bullet \# \mathcal{L}(T_1) \# \mathcal{L}(T_2),$$

in light of the formula (2) of $K$. Thus, with (3),

$$\begin{aligned} \Delta_x^{+,i} &= \mathbb{E}_{u,v}\left[K(T_i^x, T_i^u) + w_\bullet \# \mathcal{L}(T_i^x) \# \mathcal{L}(T_i^u) - K(T_i^x, T_{1-i}^v) - w_\bullet \# \mathcal{L}(T_i^x) \# \mathcal{L}(T_{1-i}^v)\right] \\ &= \Delta_x^i + \mathbb{E}_{u,v}\left[w_\bullet \# \mathcal{L}(T_i^x)(\# \mathcal{L}(T_i^u) - \# \mathcal{L}(T_i^v))\right], \end{aligned}$$

which ends the proof. ∎

The sufficient number of data provided in Corollary 2.3 is obtained (5) through the square ratio of $\max_i K(T_i, T_i)$ over $\min_i \Delta_x^i$. First, it should be noticed that $K^+(T_i, T_i) > K(T_i, T_i)$. In addition, by virtue of Proposition 2.4, either $\Delta_x^{+,0} \leq \Delta_x^0$ or $\Delta_x^{+,1} \leq \Delta_x^1$ (and the inequality is strict if trees of classes 0 and 1 have not the same number of leaves on average). Consequently,

$$\min_i \Delta_x^{+,i} \leq \min_i \Delta_x^i,$$

and thus the sufficient number of data mentioned above is minimum for $\omega_\bullet = 0$.

**Remark 2.5.** *The results stated in this section establish that the subtree kernel is more efficient when the weight of leaves is 0. It should be placed in perspective with the exponential weighting scheme of the literature [1, 7, 10, 21, 30] for which the weight of leaves is maximal. We conjecture that the accuracy of the subtree kernel should be in general improved by imposing a null weight to any subtree present in two different classes. This can not be established from the model for which the only such subtrees are the leaves. Relying on this, one of the objectives of the sequel of the paper is to develop a learning method for the weight function that improves in practice the classification results (see Sections 4 and 5).*

## 3 DAG computation of the subtree kernel

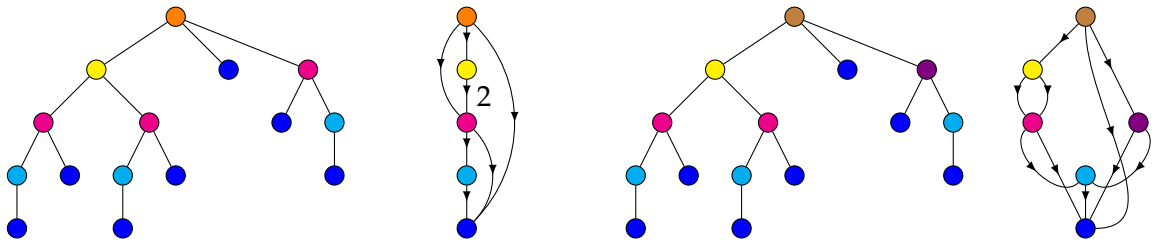
In this section, we define DAG reduction, an algorithm that achieves both compression of data and enumeration of all subtrees of a tree without redundancies. DAG reduction of a tree is presented in Subsection 3.1, while Subsection 3.2 is devoted to the compression of a forest. In Subsection 3.3, we state that the subtree kernel can be computed from the DAG reduction of dataset of trees.

### 3.1 DAG reduction of a tree

Trees can present internal repetitions in their structure. Eliminating these structural redundancies defines a reduction of the initial data that can result in a Directed Acyclic Graph (DAG). In particular, beginning with [29], DAG representations of trees are also much used in computer graphics where the process of condensing a tree into a graph is called object instancing [17]. DAG reduction can be computed upon unordered or ordered trees. We begin with the case of unordered trees.

**Unordered trees** We consider the equivalence relation "existence of an unordered tree isomorphism" on the set of the subtrees of a tree $T$: $Q(T) = (V, E)$ denotes the quotient graph obtained from $T$ using this equivalence relation. $V$ is the set of equivalence classes on the subtrees of $T$, while $E$ is a set of pairs of equivalence classes $(C_1, C_2)$ such that $\mathcal{R}(C_2) \in \mathcal{C}(\mathcal{R}(C_1))$ up to an isomorphism. The graph $Q(T)$ is a DAG [15, Proposition 1] that is a connected directed graph without path from any vertex $v$ to itself. Let $(C_1, C_2)$ be an edge of the DAG $Q(T)$. We define $L(C_1, C_2)$ as the number of occurrences of a tree of $C_2$ just below the root of any tree of $C_1$. The tree reduction of $T$ is defined as the quotient graph $Q(T)$ augmented with labels $L(C_1, C_2)$ on its edges. We refer to Fig. 3a for an example of DAG reduction of an unordered tree. Two different algorithms that allow the computation of the DAG reduction of an unordered tree but that share the same time-complexity in $\mathcal{O}(\#T^2 \deg(T) \log(\deg(T)))$ are presented in [15].

**Ordered trees** In the case of ordered trees, it is required to preserve the order of the children in the DAG reduction. As for unordered trees, we consider the quotient graph $Q(T) = (V, E)$ obtained from $T$ using the equivalence relation between ordered trees. $V$ is the set of equivalence classes on the subtrees of $T$. Here, the edges of the graph are ordered as follows. $(C_1, C_2)$ is the $r^{\text{th}}$ edge between $C_1$ and $C_2$ if $\mathcal{R}(C_2)$ is the $r^{\text{th}}$ child of $\mathcal{R}(C_1)$ up to an isomorphism. We obtain a DAG with ordered edges that compresses the initial tree $T$. An example of DAG reduction of an ordered tree is presented in Fig. 3b. Polynomial algorithms have been developed to allow the computation of a DAG, with complexities ranging in $\mathcal{O}(\#T^2)$ to $\mathcal{O}(\#T)$ for ordered trees [12].




(a) Unordered (b) Ordered

Figure 3: A tree (left) and its DAG reduction (right) seen (a) as an unordered tree and (b) as an ordered tree. In each figure, roots of isomorphic subtrees are displayed with the same color, which is reproduced on the corresponding vertex of the DAG. Note that the subtree on the left is colored differently in the two cases, whether the order of its children is relevant or not. If no label is specified on an edge (in the unordered case), it is equal to 1.

In this paper, $\mathfrak{R}^*(T)$ denotes the DAG reduction of $T$ as $*$-tree, $* \in \{\text{ordered}, \text{unordered}\}$. It is crucial to notice that the function $\mathfrak{R}^*$ is a one-to-one correspondence, which means that DAG

reduction is a lossless compression algorithm. In other words, $T$ can be reconstructed from $\mathfrak{R}^*(T)$ and $(\mathfrak{R}^*)^{-1}$ stands for the inverse function.

The DAG structure inherits of some properties of trees. For a vertex $v$ in a DAG $D$, we will denote by $\mathcal{C}(v)$ ($\mathcal{P}(v)$, respectively) the set of children (parents, respectively) of $v$. $\mathcal{H}(v)$ and $\deg(v)$ are inherited as well. Similarly to trees, we denote by $D[v]$ the subDAG rooted in $v$ composed of $v$ and all its descendants in $D$.

### 3.2 DAG reduction of a forest

Let $\mathbb{T}_{\mathcal{F}_\mathcal{T}}$ be the super-tree obtained from a forest of $*$-trees $\mathcal{F}_\mathcal{T} = (T_1, \dots, T_N)$ by placing in this order each $T_i$ as a subtree of an artificial root. We define the DAG reduction of the forest $\mathcal{F}_\mathcal{T}$ as $\mathfrak{R}^*(\mathcal{F}_\mathcal{T}) = \mathfrak{R}^*(\mathbb{T}_{\mathcal{F}_\mathcal{T}})$.

However, if the forest $\mathcal{F}_\mathcal{T}$ is stocked as a forest of compressed DAGs, that is, $\mathcal{F}_\mathcal{D} = (D_1, \dots, D_N)$ (with $D_i = \mathfrak{R}^*(T_i)$), it would be superfluous to decompress all trees before reducing the super-tree. So, one would rather compute $\mathfrak{R}^*(\mathcal{F}_\mathcal{T})$ directly from $\mathcal{F}_\mathcal{D}$. From now on, we consider only forests of DAGs that we will denote unambiguously $\mathcal{F}$. In this context, $\mathfrak{R}^*(\mathcal{F})$ stands for the DAG reduction of the forest of trees $((\mathfrak{R}^*)^{-1}(D_1), \dots, (\mathfrak{R}^*)^{-1}(D_N))$. We define the degree of the forest as $\deg(\mathcal{F}) = \max_{i=1}^N \deg(D_i)$.

Computing $\mathfrak{R}^*(\mathcal{F})$ from $(D_1, \dots, D_N)$ is in two steps: (i) we construct a super-DAG $\mathbb{D}_\mathcal{F}$ from $\mathcal{F} = (D_1, \dots, D_N)$ by placing in this order each $D_i$ as a subDAG of an artificial root (with time-complexity $\mathcal{O}(\deg(\mathcal{F}) \sum_{i=1}^N \#D_i)$), and (ii) we recompress $\mathbb{D}_\mathcal{F}$ using Algorithm 1. Fig. 4 illustrates step by step Algorithm 1 on a forest of two trees seen as unordered then ordered trees.

**Proposition 3.1.** *Algorithm 1 correctly computes $\mathfrak{R}^*(\mathcal{F})$.*

**Proof.** Starting from the leaves, we examine all vertices of same height in $\mathbb{D}_\mathcal{F}$. Those with same children (with respect to $*$) are merged into a single vertex. The algorithm stops when at some height $h$, we cannot find any vertices to be merged. Vertices that are merged in the algorithm represents isomorphic subtrees, so it suffices to prove that the algorithm stops at the right time. Let $h$ be the first height for which $\sigma(h) = \emptyset$.

Suppose by contradiction that some vertices were to be merged at some height $h' > h$. They represents isomorphic subtrees, so that their respective children should also be merged together, and all of their descendants by induction. As any vertex of height $h'' + 1$ admits at least one child of height $h''$, $\sigma(h)$ would not be empty, which is absurd. ∎

**Proposition 3.2.** *Algorithm 1 has time-complexity:*

1. $\mathcal{O}(\#\mathbb{D}_\mathcal{F} \deg(\mathcal{F})(\log \deg(\mathcal{F}) + \mathcal{H}(\mathbb{D}_\mathcal{F})))$ *for unordered trees;*
2. $\mathcal{O}(\#\mathbb{D}_\mathcal{F} \deg(\mathcal{F})\, \mathcal{H}(\mathbb{D}_\mathcal{F}))$ *for ordered trees.*

**Proof.** The proof lies in Appendix B. ∎

**Remark 3.3.** *One might also want to treat online data, but without recompressing the whole dataset when adding a single entry in the forest. Let $\mathfrak{R}^*(\mathcal{F})$ be the already recompressed forest and $D$ a new DAG to be introduced in the data. It suffices to place $D$ has the rightmost child of the artificial root of $\mathfrak{R}^*(\mathcal{F})$ to get $\mathbb{D}_{\mathcal{F}\cup D}$, then run Algorithm 1 to obtain $\mathfrak{R}^*(\mathcal{F} \cup D)$.*

**Algorithm 1:** DAGRECOMPRESSION

**Data:** $\mathbb{D}_{\mathcal{F}}$ the superdag obtained from a forest of DAG reductions of $*$-trees, $* \in \{\text{ordered}, \text{unordered}\}$

**Result:** $\mathfrak{R}^*(\mathcal{F})$

1 Construct, within one exploration of $\mathbb{D}_{\mathcal{F}}$, the mapping $h \mapsto \mathbb{D}_{\mathcal{F}}{}^h$ where $\mathbb{D}_{\mathcal{F}}{}^h$ is the set of vertices of $\mathbb{D}_{\mathcal{F}}$ at height $h$

2 **for** $h$ **from** $0$ **to** $\mathcal{H}(\mathbb{D}_{\mathcal{F}}) - 1$ **do**

3 Let $\sigma(h) = \left\{f^{-1}(\{S\}) : S \in \operatorname{Im} f, \#f^{-1}(\{S\}) \geq 2\right\}$ be the set of vertices to be merged at height $h$, where $f : \nu \in \mathbb{D}_{\mathcal{F}}{}^h \mapsto \mathcal{C}(\nu)$

4 **if** $\sigma(h) = \emptyset$ **then**

5 Exit algorithm;

6 **else**

7 **for** $M$ **in** $\sigma(h)$ **do**

8 Choose one element $\nu_M$ in $M$ to remain in $\mathbb{D}_{\mathcal{F}}$

9 Denote by $\delta_M$ the other elements of $M$

10 **for** $\nu$ **in** $\mathbb{D}_{\mathcal{F}}$ **such that** $\mathcal{H}(\nu) > h$ **do**

11 **for** $\mu$ **in** $\mathcal{C}(\nu)$ **such that** $\exists M \in \sigma(h)$, $\delta_M \ni \mu$ **do**

12 Delete $\mu$ from $\mathcal{C}(\nu)$

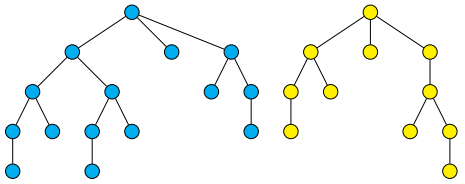
13 Add $\nu_M$ in $\mathcal{C}(\nu)$

14 **for** $M \in \sigma(h)$ **do**

15 Delete $\delta_M$ from $\mathbb{D}_{\mathcal{F}}$

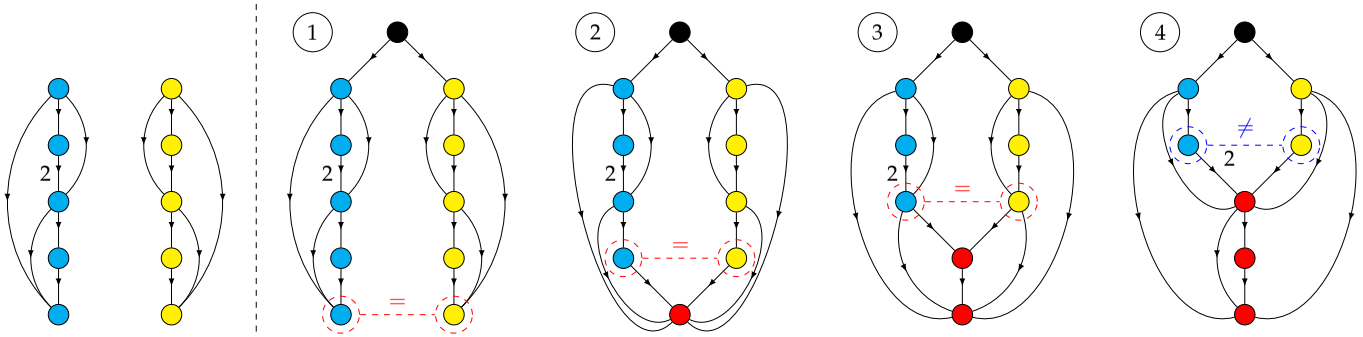
16 **return** $\mathbb{D}_{\mathcal{F}}$

It should be noticed that $\operatorname{Im} f$ (that appears line 3) depends on $*$. Indeed, if $* =$ ordered, $\operatorname{Im} f$ is the set of all *lists* of children; otherwise, $\operatorname{Im} f$ is the set of all *multisets* of children.

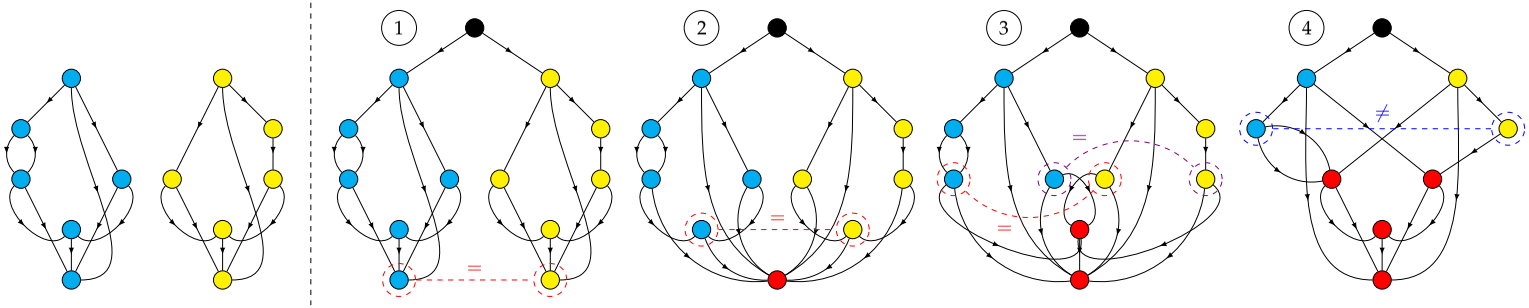


(a)

Figure 4: An illustration step by step of the Algorithm 1 with (a) two trees $T_1$ (in cyan) and $T_2$ (in yellow), seen as (b) unordered or (c) ordered trees. One can observe the DAGs (left) and the execution of the algorithm (right). At each step 1, 2 and 3, we examine vertices at height (0,1,2) and merge those which have same children. At step 4, we can not find any vertex to merge and we stop. Note that in (c) at step 3, we find two pairs of vertices to be merged : we are not restricted to one pair per height. Merged vertices are colored in red. The artificial root is colored in **black**.




(b)




(c)

### 3.3 DAG annotation and kernel computation

We consider a dataset composed of two parts: the train dataset $\mathcal{X}_{\text{train}} = (T_1, \dots, T_n)$ and the dataset to predict $\mathcal{X}_{\text{pred}} = (T_{n+1}, \dots, T_N)$. In the train dataset, the classes of the data are assumed to be known. Our aim is to compute two Gram matrices $G = [K(T_i, T_j)]_{i,j}$, where:

- $(i, j) \in \mathcal{X}_{\text{train}} \times \mathcal{X}_{\text{train}}$ for the training matrix $G_{\text{train}}$;
- $(i, j) \in \mathcal{X}_{\text{pred}} \times \mathcal{X}_{\text{train}}$ for the prediction matrix $G_{\text{pred}}$.

SVM algorithms will use $G_{\text{train}}$ to learn their classifying rule, and $G_{\text{pred}}$ to make predictions [9, Section 6.1]. Other algorithms, such as kernel PCA, would also require to compute a Gram matrix before processing [25, Section 14.2]. We denote by $\Delta = \mathfrak{R}^*(\mathcal{X}_{\text{train}} \cup \mathcal{X}_{\text{pred}})$ the DAG reduction of the dataset and, for any $1 \leq i \leq N$, $D_i = \mathfrak{R}^*(T_i)$. DAG computation of the subtree kernel requires to annotate the DAG with different pieces of information.

**Origins** In order to compute the subtree kernel, it will be necessary to retrieve from the vertices of $\Delta$ their origin in the dataset, that is, from which tree they come from. For any vertex $v$ in $\Delta \setminus \mathcal{R}(\Delta)$, the origin of $v$ is defined as

$$o(v) = \big\{ i \in \{1, \dots, n, n+1, \dots, N\} : D_i \ni v \big\}.$$

Assuming that $(D_1, \dots, D_N)$ are children of the root of $\Delta$ in this order (which is achieved if $\Delta$ had been constructed following the ideas developed in Subsection 3.2) leads to the following proposition.

**Proposition 3.4.** *Origins can be calculated using the recursive formula,*

$$\forall v \in \Delta \setminus \mathcal{R}(\Delta),\ o(v) = \begin{cases} \{i\} & \text{if } v \text{ is the } i^{th} \text{ child of } \mathcal{R}(\Delta), \\ \bigcup\limits_{p \in \mathcal{P}(v)} o(p) & \text{otherwise.} \end{cases}$$

**Proof.** Using the assumption, origins are correct for the children of $\mathcal{R}(\Delta)$. If $D_i \ni v$ for some $i \in \{1, \dots, N\}$ and $v \in \Delta$, then $D_i \supseteq \mathcal{D}(v)$. The statement follows by induction. ∎

**Frequency vectors** Remember that in (2) $N_\tau(T)$ counts the number of subtrees of a tree $T$ that are $*$-isomorphic to the tree $\tau$. To compute the kernel, we need to know this value, and we claim that we can compute it using only $\Delta$. We associate to each vertex $v \in \Delta \setminus \mathcal{R}(\Delta)$ a frequency vector $\varphi_v$ where, for any $1 \leq i \leq N$, $\varphi_v(i) = N_{(\mathfrak{R}^*)^{-1}(\Delta[v])}(T_i)$.

**Proposition 3.5.** *Frequency vectors can be calculated using the recursive formula,*

$$\forall v \in \Delta \setminus \mathcal{R}(\Delta),\ \varphi_v = \begin{cases} (\mathbb{1}_{\{i \in o(v)\}})_{i \in \{1, \dots, N\}} & \text{if } v \in \mathcal{C}(\mathcal{R}(\Delta)), \\ \sum\limits_{p \in \mathcal{P}(v)} L(p, v)\, \varphi_p & \text{otherwise,} \end{cases}$$

*where either* $L(p, v) = 1$ *if* $* =$ ordered, *or* $L(p, v)$ *is the label on the edge between* $p$ *and* $v$ *in* $\Delta$ *if* $* =$ unordered.

**Proof.** Let $v$ be in $\Delta \setminus \mathcal{R}(\Delta)$. If $v \in \mathcal{C}(\mathcal{R}(\Delta))$, then $v$ represents the root of a tree $T_i$ (possibly several trees if there are repetitions in the dataset), and therefore $\varphi_v(i) = N_{T_i}(T_i) = 1$. Otherwise, suppose by induction that $\varphi_p(i)$ is correct for all $p \in \mathcal{P}(v)$, and any $i$. We fix $p \in \mathcal{P}(v)$. $v$ appears $L(p, v)$ times as a child of $p$, so if $(\mathfrak{R}^*)^{-1}(\Delta[p])$ appears $\varphi_p(i)$ times in $T_i$, then the number of occurrences of $(\mathfrak{R}^*)^{-1}(\Delta[v])$ in $T_i$ as a child of $(\mathfrak{R}^*)^{-1}(\Delta[p])$ is $L(p, v)\, \varphi_p(i)$. Summing over all $p \in \mathcal{P}(v)$ leads $\varphi_v(i)$ to be correct as well. ∎

**DAG weighting** The last thing that we lack to compute the kernel is the weight function. Remember that it is defined for trees as a function $w : \mathcal{T} \to \mathbb{R}^+$. As we only need to know the weights of the subtrees associated to vertices of $\Delta$, we define the weight function for DAG as, for any $v \in \Delta$, $\omega_v = w_{(\mathfrak{R}^*)^{-1}(\Delta[v])}$.

**Remark 3.6.** *In light of Propositions 3.4 and 3.5, it should be noted that both $o$ and $\varphi$ can be calculated in one exploration of $\Delta$. By definition, this is also true for $\omega$.*

**DAG computation of the subtree kernel** We introduce the matching subtrees function $\mathcal{M}$ as

$$\begin{aligned} \mathcal{M} \colon \{1, \dots, N\}^2 &\to 2^{\Delta} \\ (i, j) &\mapsto \{v \in \Delta : \{i, j\} \subseteq o(v)\} \end{aligned}$$

where $2^{\Delta}$ is the powerset of the vertices of $\Delta$. Note that $\mathcal{M}$ is symmetric. This leads us to the following proposition.

**Proposition 3.7.** *For any $T_i, T_j \in \mathcal{X}_{\text{train}} \cup \mathcal{X}_{\text{pred}}$, we have*

$$K(T_i, T_j) = \sum_{v \in \mathcal{M}(i,j)} \omega_v\, \varphi_v(i)\, \varphi_v(j).$$

**Proof.** By construction, it suffices to show that $\mathfrak{R}^*(\mathcal{S}(T_i) \cap \mathcal{S}(T_j)) = \mathcal{M}(i, j)$. Let $\tau \in \mathcal{S}(T_i) \cap \mathcal{S}(T_j)$. Then $\mathfrak{R}^*(\tau) \in \mathfrak{R}^*(T_i)$ and $\mathfrak{R}^*(\tau) \in \mathfrak{R}^*(T_j)$. Necessarily, $\mathfrak{R}^*(\tau) \in \Delta$ and $\{i, j\} \subseteq o(\mathfrak{R}^*(\tau))$. So $\mathfrak{R}^*(\tau) \in \mathcal{M}(i, j)$. Reciprocally, let $v \in \mathcal{M}(i, j)$. We denote $\tau = (\mathfrak{R}^*)^{-1}(v)$. As $\{i, j\} \subseteq o(v)$, then $\tau \in \mathcal{S}(T_i) \cap \mathcal{S}(T_j)$. ∎

**Remark 3.8.** *$\mathcal{M}$ can be created in $\mathcal{O}(N^2 \#\Delta)$ within one exploration of $\Delta$ and allows afterward computations of the subtree kernel $K(T_i, T_j)$ in $\mathcal{O}(\#\mathcal{M}(i, j)) = \mathcal{O}(\min(\#D_i, \#D_j))$, which is more efficient than the $O(\#T_i + \#T_j)$ algorithm proposed in [30] (the time-complexity is announced in [21, Section 1]). However, since the whole process through Algorithm 1 is costly, the global method that we propose in this paper is not faster than existing algorithms. Nonetheless, our algorithm is particularly adapted to repeated computations from the same data, e.g., for tuning parameters. Indeed, once $\mathcal{M}$ and $\Delta$ have been created, they can be stored and are ready to use. An illustration of this property is provided from experimental data in Fig. 19.*

**Remark 3.9.** *The DAG computation of the subtree kernel investigated in this section relies on the references [1, 10]. Our work and the aforementioned papers are different and complementary. First, our framework is valid for both ordered and unordered trees, while these papers focus only on ordered trees. In addition, the method developed in [1, 10] is only adapted to exponential weights (see equations (3.12) and (6.2) from [10]). Thus, even if this algorithm is also based on DAG reduction of trees, it is less general than ours since the weight function is not constrained (see in particular Section 4 where the weight function is learned from the data). Finally, in [1, Section 4], the time-complexities are studied only from a numerical point of view, while we state theoretical results.*

# 4 Discriminance weight function

For a given probability level and a given classification error, and under the stochastic model of Subsection 2.2, we state in Subsection 2.4 that the sufficient size of the training dataset is minimum when the weight of leaves is 0. In other words, counting the leaves, which are the only subtrees that appear in both classes, does not provide a relevant information to the classification problem associated to this model. As mentioned in Remark 2.5, we conjecture that, in a more general model, this result would be true for any subtree present in both classes. In this section, we propose to rely on this idea by defining a new weight function, learned from the data and called discriminance weight that assigns a large weight to subtrees, that help to discriminate the classes, i.e., that are present or absent in exactly one class, and a low weight otherwise.

The training dataset is divided into two parts: $\mathcal{X}_{\text{weight}} = (T_1, \dots, T_m)$ to learn the weight function, and $\mathcal{X}_{\text{class}} = (T_{m+1}, \dots, T_n)$ to estimate the Gram matrix. For the sake of readability, $\Delta$ denotes the DAG reduction of the whole dataset, including $\mathcal{X}_{\text{weight}}$, $\mathcal{X}_{\text{class}}$ and $\mathcal{X}_{\text{pred}}$. In addition, we assume that the data are divided into $K$ classes numbered from 1 to $K$.

For any vertex $v \in \Delta \setminus \mathcal{R}(\Delta)$, we define the vector $\rho_v$ of length $K$ as,

$$\forall\, 1 \leq k \leq K,\ \rho_v(k) = \frac{1}{\#C_k} \sum_{T_i \in C_k} \mathbb{1}_{\{i \in o(v)\}},$$

where $(C_k)_{1 \leq k \leq K}$ forms a partition of $\mathcal{X}_{\text{weight}}$ such that $T_i \in C_k$ if and only if $T_i$ is in class $k$. In other words, $\rho_v(k)$ is the proportion of data in class $k$ that contain the subtree $(\mathfrak{R}^*)^{-1}(\Delta[v])$. Therefore, $\rho_v$ belongs to the $K$-dimensional hypercube. It should be noticed that $\rho_v$ is a vector of zeros as soon as $(\mathfrak{R}^*)^{-1}(\Delta[v])$ is not a subtree of a tree of $\mathcal{X}_{\text{weight}}$.

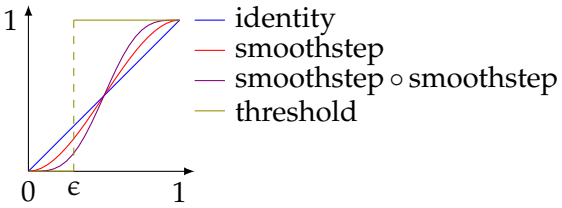
For any $1 \leq k \leq K$, let $e_k$ ($\bar{e}_k$, respectively) be the vector of zeros with a unique 1 in position $k$ (vector of ones with a unique 0 in position $k$, respectively). If $\rho_v = e_k$, the vertex $v$ corresponds to the subtree $(\mathfrak{R}^*)^{-1}(\Delta[v])$, which only appears in class $k$: $v$ is thus a good discriminator of this class. Otherwise, if $\rho_v = \bar{e}_k$, the vertex $v$ appears in all the classes except class $k$ and is still a good discriminator of the class. For any vertex $v$, $\delta_v$ measures the distance between $\rho_v$ and its nearest point of interest $e_k$ or $\bar{e}_k$,

$$\delta_v = \min_{k=1}^{K} \min(|\rho_v - e_k|, |\rho_v - \bar{e}_k|).$$

It should be noted that the maximum value of $\delta_v$ depends on the number of classes and can be larger than 1. If $\delta_v$ is small, then $\rho_v$ is close to a point of interest. Consequently, since $v$ tends to discriminate a class, its weight should be large. In light of this remark, the discriminance

weight of a vertex $v$ is defined as $\omega_v = f(1-\delta_v)$, where $f : (-\infty, 1] \to [0,1]$ is increasing with $f(x) = 0$ for $x \leq 0$ and $f(1) = 1$. Fig. 5 illustrates some usual choices for $f$. In the sequel, we chose $\omega_v = f^*(1-\delta_v)$ with the smoothstep function $f^* : x \mapsto 3x^2 - 2x^3$. We borrowed the smoothstep function from computer graphics [13, p. 30], where it is mostly used to have smooth transition in a threshold function.

Figure 5: The discriminance weight is defined by $\omega_\tau = f(1-\delta_\tau)$ where $f : (-\infty, 1] \to [0,1]$ is increasing with $f(0) = 0$ and $f(1) = 1$. This figure presents some usual choices for $f$.




Since leaves appear in all the trees of the training dataset, $\rho_\bullet$ is a vector of ones and thus $\delta_\bullet = 1$, which implies $\omega_\bullet = 0$. This is consistent with the result developed in Subsection 2.4 on the stochastic model. As aforementioned, the discriminance weight is inspired from the theoretical results established in Subsection 2.4 and the conjecture presented in Remark 2.5. The relevance in practice of this weight function will be investigated in the sequel of the paper through two applications.

**Remark 4.1.** *The discriminance weight is defined from the proportion of data in each class that contain a given subtree, for all the subtrees appearing in the dataset. It is thus required to enumerate all these subtrees. This is done, without redundancy, via the DAG reduction $\Delta$ of the dataset defined and investigated in Section 3. As the $m$ trees of the training dataset dedicated to learning the discriminance weight are partitioned into $K$ classes, computing one $\rho_v$ vector is of complexity $\mathcal{O}(m)$. Therefore, computing all of them is in $\mathcal{O}(\#\Delta m)$. In addition, computing all values of $\delta_v$ is in $\mathcal{O}(\#\Delta K^2)$, as there are $2K$ Euclidean distances to be computed for each vector of length $K$. All gathered, computing the discriminance weight function has an overall complexity of $\mathcal{O}(\#\Delta(N + K^2))$.*

# 5 Real data analysis

This section is dedicated to the application of the methodology developed in the paper to eight real datasets with various characteristics in order to show its strengths and weaknesses. The related questions are supervised classification problems. As mentioned in Subsection 3.3, our approach consists in computing the Gram matrices of the subtree kernel via DAG reduction and with a new weight function called the discriminance (see Section 4). In particular, we aim to compare the usual exponential weight of the literature and the latter in terms of prediction capability. In all the sequel, the Gram matrices are used as inputs to SVM algorithms in order to tackle these classification problems. We emphasize that this approach is not restricted to SVM but can be applied with other prediction algorithms.

## 5.1 Preliminaries

In this subsection, we introduce (i) the protocol that we have followed to investigate several datasets, together with a description of (ii) the classification metrics that we use to assess the

quality of our results, (iii) an extension of DAG reduction to take into account discrete labels on vertices of trees, and (iv) the standard method to convert a markup document into a tree. It should be already noted that all the datasets presented in the sequel are composed of trees (that can be ordered or unordered, labeled or not) together with their class.

**Protocol** For each dataset, we have followed the same presentation and procedure. First, a description of the data is made notably via histograms describing the size, outdegree, height and class repartition of trees. Given the dispersion of some of these quantities, we have binned together the values that does not fit inside the interval $[Q_1 - 1.5 \cdot IQR; Q_3 + 1.5 \cdot IQR]$ where $IQR = Q_3 - Q_1$ is the interquartile range. Therefore, the flattened-large bins that appears in some histograms represents those outliers bins. The objective of this part is to show the wide range of datasets considered in the paper.

In a second time, we evaluated the performance of the subtree kernel on a classification task via two methods: (i) for exponential weights $\tau \mapsto \lambda^{\mathcal{H}(\tau)}$ we randomly split the data in half, one for training a SVM, and one for prediction; (ii) for discriminance weight, we randomly split the data in thirds, one for training the discriminance weight, one for training a SVM, and the last one for prediction. We repeated 50 times this random split for discriminance, and for different values of $\lambda$. The classification results are assessed by some metrics defined in the upcoming paragraph, and gathered in boxplots. The first application example, presented in Subsection 5.2, is slightly different since (i) we have worked with 50 distinct databases, and (ii) the results have been completed with a deeper analysis of the discriminance weights, in relation with the usual weighting scheme of the literature.

**Classification metrics** To quantify the quality of a prediction, we use four standard metrics that are accuracy, precision, recall and F-score. For a class $k$, one can have true positives $TP_k$, false positives $FP_k$, true negatives $TN_k$ and false negatives $FN_k$. In a binary classification problem, those metrics are defined as,

$$\begin{aligned}
\text{Accuracy}(k) &= \frac{TP_k + TN_k}{TP_k + FP_k + FN_k + TN_k}, \\
\text{Precision}(k) &= \frac{TP_k}{TP_k + FP_k}, \\
\text{Recall}(k) &= \frac{TP_k}{TP_k + FN_k}, \\
\text{F-score}(k) &= \frac{2\,\text{Precision}(k)\,\text{Recall}(k)}{\text{Precision}(k) + \text{Recall}(k)}.
\end{aligned}$$

For a problem with $K > 2$ classes, we adopt the macro-average approach, that is,

$$\text{Metric} = \frac{1}{K}\sum_{k=1}^{K} \text{Metric}(k).$$

We used the implementation available in the `scikit-learn` Python library, via the two functions `accuracy_score` and `precision_recall_fscore_support`.

**DAG reduction with labels** In the sequel, some of the presented datasets are composed of labeled trees, that are trees which each vertex possesses a label. Labels are supposed to take only a finite number of different values. Two labeled $*$-trees are said isomorphic if (i) they are $*$-isomorphic, and (ii) the underlying one-to-one correspondence mapping vertices of $T_1$ into vertices of $T_2$ is such that $\forall v \in T_1$, $v$ and $\Phi(v)$ have the same label. The set of labeled $*$-trees is the quotient set of rooted trees by this equivalence relation. It should be noticed that the subtree kernel as well as DAG reduction are defined through only the concept of isomorphic subtrees. As a consequence, they can be straightforwardly extended to labeled $*$-trees. This formalization is an extension of the definition introduced by the authors of [1, 10], as they consider only ordered labeled trees, whereas we can consider unordered labeled trees as well.

**From a markup document to a tree** Some of the datasets come from markup documents (`XML` or `HTML` files). From such a document, one can extract a tree structure, identifying each couple of opening and closing tags as a vertex, which children are the inner tags. It should be noticed that, during this transcription, semantic data is forgotten: the tree only describes the topology of the document. Fig. 6 illustrates the conversion from `HTML` to tree on a small example. Such a tree is ordered but can be considered as unordered. Finally, a tag can also be chosen as a label for the corresponding vertex in the tree.

```
<html>
    <body>
        <h1>
            Lorem ipsum dolor sit amet, consectetur adipiscing elit.
        </h1>
        <p>
            Sed non risus.
        </p>
        <ul>
            <li>
                Suspendisse lectus tortor, dignissim sit amet,
                adipiscing nec, ultricies sed, dolor.
            </li>
            <li>
                Cras elementum ultrices diam.
                <ol>
                    <li>
                        Maecenas ligula massa, varius a,
                        semper congue, euismod non, mi.
                    </li>
                    <li>
                        Proin porttitor, orci nec nonummy
                        molestie, enim est eleifend mi,
                        non fermentum diam nisl sit amet erat.
                    </li>
                </ol>
            </li>
            <li>
                Duis semper. Duis arcu massa, scelerisque vitae,
                consequat in, pretium a, enim.
            </li>
        </ul>
        <p>
            Pellentesque congue. Ut in risus volutpat libero
            pharetra tempor.
        </p>
    </body>
</html>
```

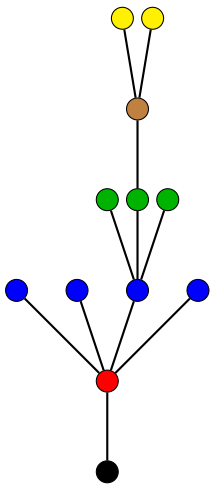


Figure 6: Underlying ordered tree structure (right) present in a `HTML` document (left). Each level in the tree is colored in the same way as the corresponding tags in the document. Natural order from top to bottom in the `HTML` document corresponds to left-to-right order in the tree.

## 5.2 Prediction of the language of a Wikipedia article from its topology

**Classification problem and results** Wikipedia pages are encoded in `HTML` and, as aforementioned, can therefore be converted into trees. In this context, we are interested in the following question: does the (ordered or unordered) topology of a Wikipedia article (as an `HTML` page) contain the information of the language in which it has been written? This can be formulated as a supervised classification problem: given a training dataset composed of the tree structures

of Wikipedia articles labeled with their language, is a prediction algorithm able to predict the language of a new data only from its topology? The interest of this question is discussed in Remark 5.1.

In order to tackle this problem, we have built 50 databases of 480 trees each, converted from Wikipedia articles as follows. Each of the databases is composed of 4 datasets:

- a dataset to predict $\mathcal{X}_{\text{pred}}$ made of 120 trees;
- a small train dataset $\mathcal{X}_{\text{train}}^{\text{small}}$ made of 40 trees;
- a medium train dataset $\mathcal{X}_{\text{train}}^{\text{medium}}$ made of 120 trees;
- and a large train dataset $\mathcal{X}_{\text{train}}^{\text{large}}$ made of 200 trees.

For each dataset, and each language, we picked Wikipedia articles at random using the Wikipedia API[1], and converted them into unlabeled trees. It should be noted that the probability to have the same article in at least two different languages is extremely low. For each database, we aim at predicting the language of the trees in $\mathcal{X}_{\text{pred}}$ using a SVM algorithm based on the subtree kernel for ordered and unordered trees, and trained with $\mathcal{X}_{\text{train}}^{\text{size}}$ where size $\in \{\text{small}, \text{medium}, \text{large}\}$. Fig. 8 provides the description of one typical database. All trees seem to share common characteristics, regardless of their class.

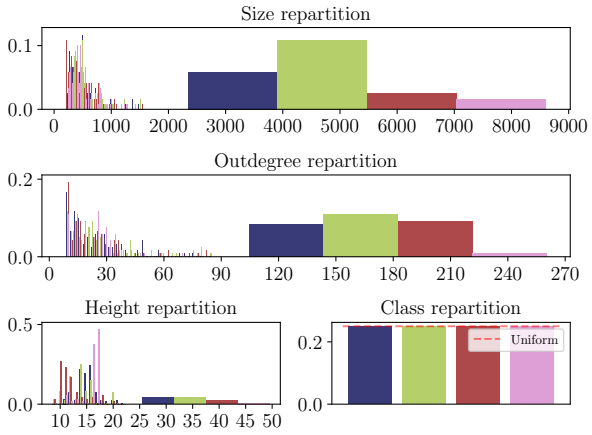



Figure 7: Description of a Wikipedia dataset (480 trees).

Classification results over the 50 databases are displayed in Fig. 8. Discriminance weighting achieves highly better results than exponential weighting, with all metrics greater than 90% on average from only 200 training data. This points out that the language information exists in the structure of Wikipedia pages, whether they are considered as ordered or unordered trees, unlike what intuition as well as subtree kernel with exponential weighting suggest. It should be added that the variance of all metrics seem to decrease with the size of the training dataset when using discriminance.

[1] https://www.mediawiki.org/wiki/API:Random (last accessed in October 2019)

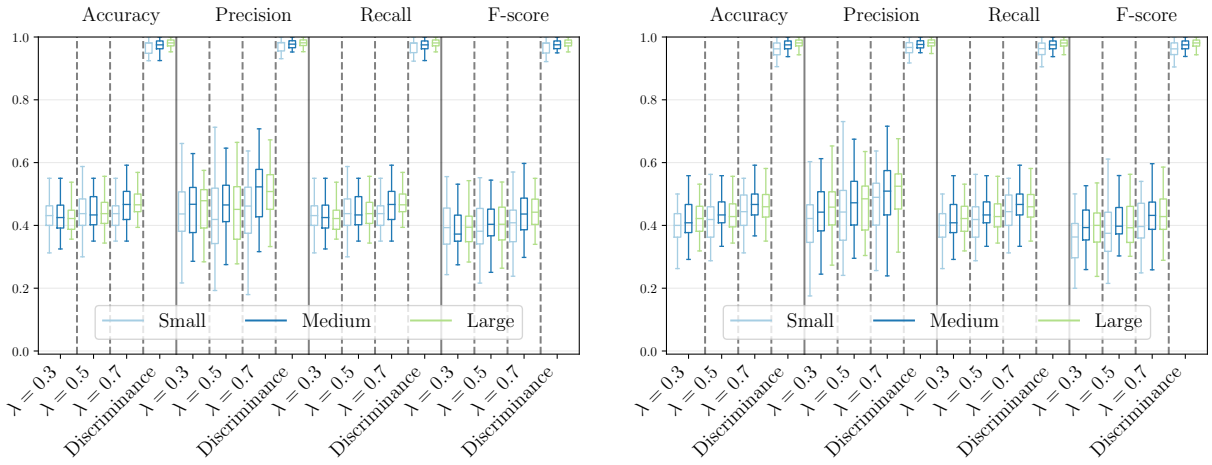



Figure 8: Classification results for the 50 Wikipedia databases as ordered (left) and unordered (right) trees. λ values stands for exponential decay weight of the form $\tau \mapsto \lambda^{\mathcal{H}(\tau)}$. The colors of the boxplot indicates, for each size $\in$ {small, medium, large}, the results obtained for the classification of $\mathcal{X}_{\text{pred}}$ from $\mathcal{X}^{\text{size}}_{\text{train}}$.

These numerical results show the great interest of the discriminance weight, in particular with respect to an exponential weight decay. Nevertheless, it should be compelling in this context to understand the classification rule learned by the algorithm. Indeed, this could lead to explain how the information of the language is present in the topology of the article.

**Comprehensive learning and data visualization** When a learning algorithm is efficient for a given prediction problem, it is interesting to understand which features are significant. In the subtree kernel, the features are the subtrees appearing in all the trees of all the classes. Looking at (2), the contribution of any subtree $\tau$ to the subtree kernel with discriminance weighting is the product of two terms: the discriminance weight $w_\tau$ quantifies the ability of $\tau$ to discriminate a class, while $\kappa(N_\tau(T_1), N_\tau(T_2))$ evaluates the similarity between $T_1$ and $T_2$ with respect to $\tau$ through the kernel $\kappa$. As explained in Section 4, if $w_\tau$ is close to 1, $\tau$ is an important feature in the prediction problem.

As shown in Section 3, DAG reduction provides a tool to compress a dataset without loss. We recall that each vertex of the DAG represents a subtree appearing in the data. Consequently, we propose to visualize the important features on the DAG of the dataset where the radius of the vertices is an increasing function of the discriminance weight. In addition, each vertex of the DAG can be colored as the class that it helps to discrimine, either positively (the vertex of the DAG corresponds to a subtree that is present almost only in the trees of this class), or negatively. This provides a visualization at a glance of the whole dataset that highlights the significant features for the underlying classification problem. We refer the reader to Fig. 9 for an application to one of our datasets. Thanks to this tool, we have remarked that the subtree corresponding to the License at the bottom of any article highly depends on the language, and thus helps to predict the class.

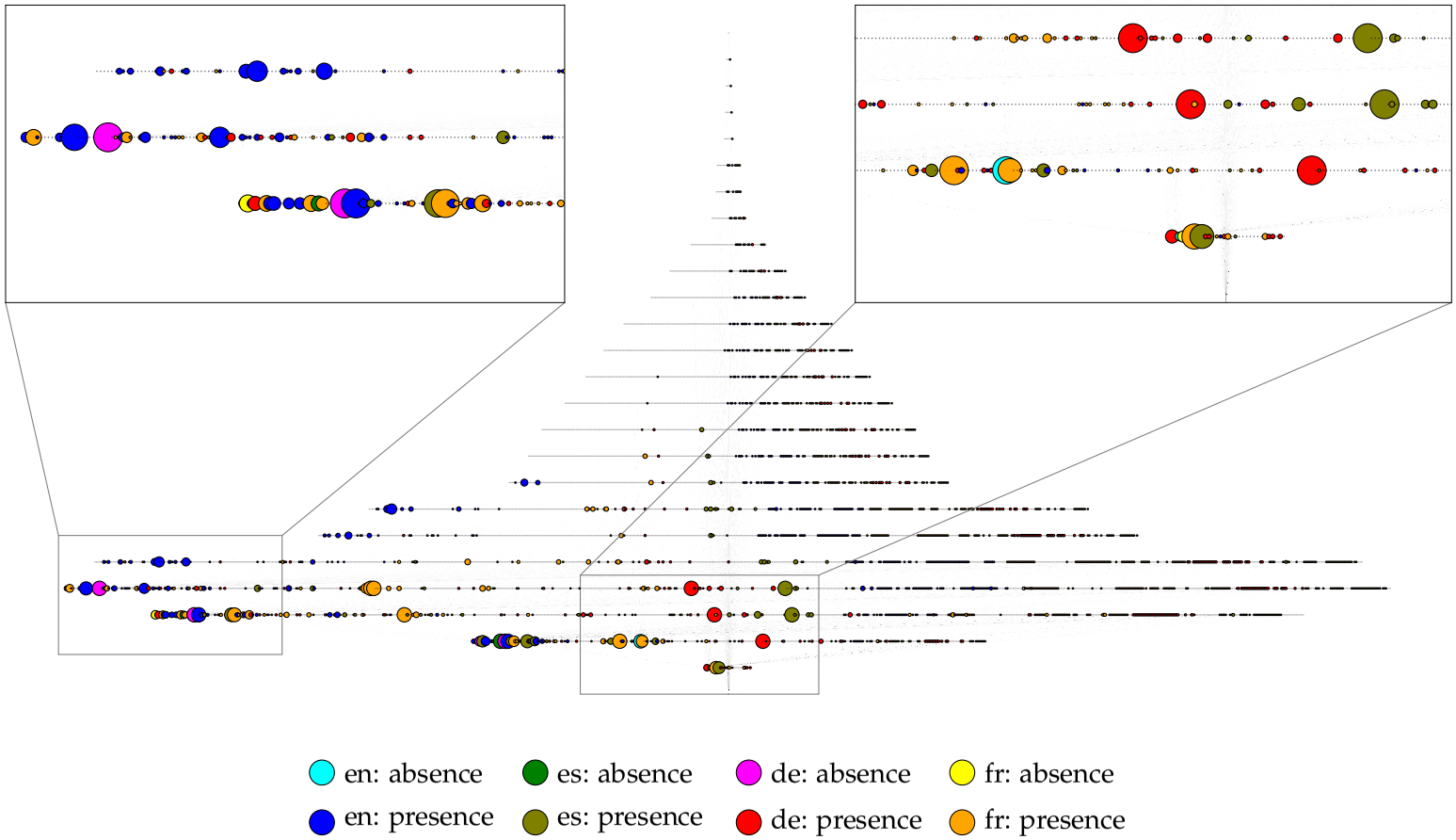



Figure 9: Visualization of one dataset $\mathcal{X} = \mathcal{X}_{\text{train}}^{\text{medium}} \cup \mathcal{X}_{\text{pred}}$ of unordered trees among the 30 Wikipedia databases. Each vertex $v \in \mathfrak{R}^*(\mathcal{X})$ is scaled according to $f^*(1 - \delta_v)$ so that the largest vertices are those that best discriminate the different classes. For each $v$, we find the class $k$ such that $\rho_v$ has minimal distance to either $e_k$ or $\overline{e}_k$. If it is $e_k$, we say that $v$ discriminates by its presence, and if it is $\overline{e}_k$, $v$ discriminates by its absence. We color $v$ following this distinction according to the legend, where "en" is for English language, "de" for German, "fr" for French, and "es" for Spanish.

**Distribution of discriminance weights** To provide a better understanding of our results, we have analyzed in Fig. 10 the distribution of discriminance weights of one of our large training datasets. It shows that the discriminance weight behaves on average as a shifted exponential. Considering the great performance achieved by the discriminance weight, this illustrates that exponential weighting presented in the literature is indeed a good idea, when setting $w_\bullet = 0$ as shown in Subsection 2.4 or suggested in [30, 6 Experimental results]. However, a closer look to the distribution in Fig. 10 (left) reveals that important features in the kernel are actually outliers: relevant information is both far from the average behavior and scarce. To a certain extent and regarding these results, discriminance weight is the second order of the exponential weight.

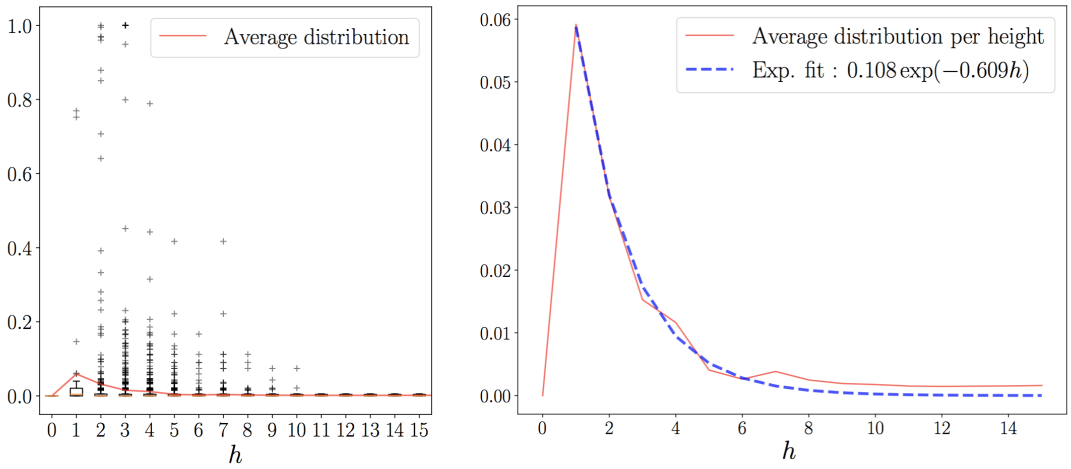



Figure 10: Estimation of the distribution of the discriminance weight function $h \mapsto \{w_\nu : \mathcal{H}(\nu) = h, \nu \in \mathfrak{R}^*(\mathcal{X})\}$ from one large training Wikipedia dataset of unordered trees (left) and fit of its average behavior (in red) to an exponential function (in blue). All ordered and unordered datasets show a similar behavior.

**Remark 5.1.** *The classification problem considered in this subsection may seem unrealistic as ignoring the text information is obviously counterproductive in the prediction of the language of an article. Nevertheless, this application example is of interest for two main reasons. First, this prediction problem is difficult as shown by the bad results obtained from the subtree kernel with exponential weights (see Fig. 8). As highlighted in Fig. 9 and 10 (left), the subtrees that can discriminate the classes are very unfrequent and diverse (in terms of size and structure), so difficult to be identified. On a different level, as Wikipedia has a very large corpus of pages, it provides a practical tool to test our algorithms and investigate the properties of our approach. Indeed, we can virtually create as many different datasets as we want by randomly picking articles, ensuring that we avoid overfitting.*

### 5.3 Markup documents datasets

We present and analyze in this subsection three datasets obtained from markup documents.

**INEX 2005 and 2006** These datasets originate from the INEX competition [11]. There are `XML` documents, that we have been considering as ordered and unordered in our experiments. INEX

2005 is made of 9 630 documents arranged in 11 classes, whereas INEX 2006 has 18 classes for 12 107 documents. For INEX 2005, classes can be split into two groups of trees with similar characteristics, as shown in Fig. 11 (left). However, inside each group, all trees are alike. In the case of INEX 2006, no special group seems to emerge from topological characteristics of the data, as pointed out in Fig. 11 (right).

The classification results are depicted in Fig. 12, for both datasets, and with trees considered successively as ordered and unordered. For INEX 2005, both exponential decay and discriminance achieve similar good performance. However, for INEX 2006, neither of them are able to achieve significant results. Actually, discriminance performs slightly worse than exponential decay. From these results we deduce that subtrees do not seem to form the appropriate substructure to capture the information needed to properly classify the data.

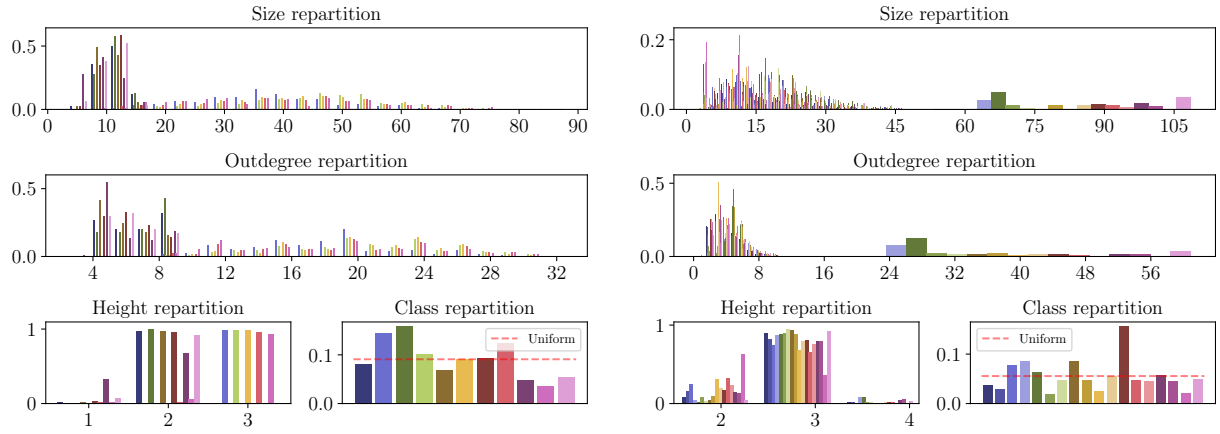



Figure 11: Description of INEX 2005 (9 630 trees, left) and INEX 2006 (12 107 trees, right) datasets.

**Videogame sellers** We manually collected, for two major websites selling videogames[2], the URLs of the top 100 best-selling games, and converted them into ordered labeled trees. As webpages might seem similar to some extent, the trees are actually very different, as highlighted in Fig. 13. We found that the subtree kernel retrieves this information as, for both exponential decay and discriminance weights, we achieved 100% of correct classifications in all our tests.

## 5.4 Biological datasets

In this subsection, three datasets from the literature are analyzed, all related to biological topics.

**Vascusynth** The Vascusynth dataset from [16, 20] is composed of 120 unordered trees that represent blood vasculatures with different bifurcations numbers. In a tree, each vertex has a continuous label describing the radius $r$ of the corresponding vessel. We have discretized these continuous labels in three categories: small if $r < 0.02$, medium if $0.02 \leq r < 0.04$ and large if $r \geq 0.04$ (all values are in arbitrary unit). We split up the trees into three classes, based on their bifurcation number. Based on Fig. 14 (left), we can distinguish between the three classes by looking only at

[2] steam.com and gog.com (last accessed in October 2019)

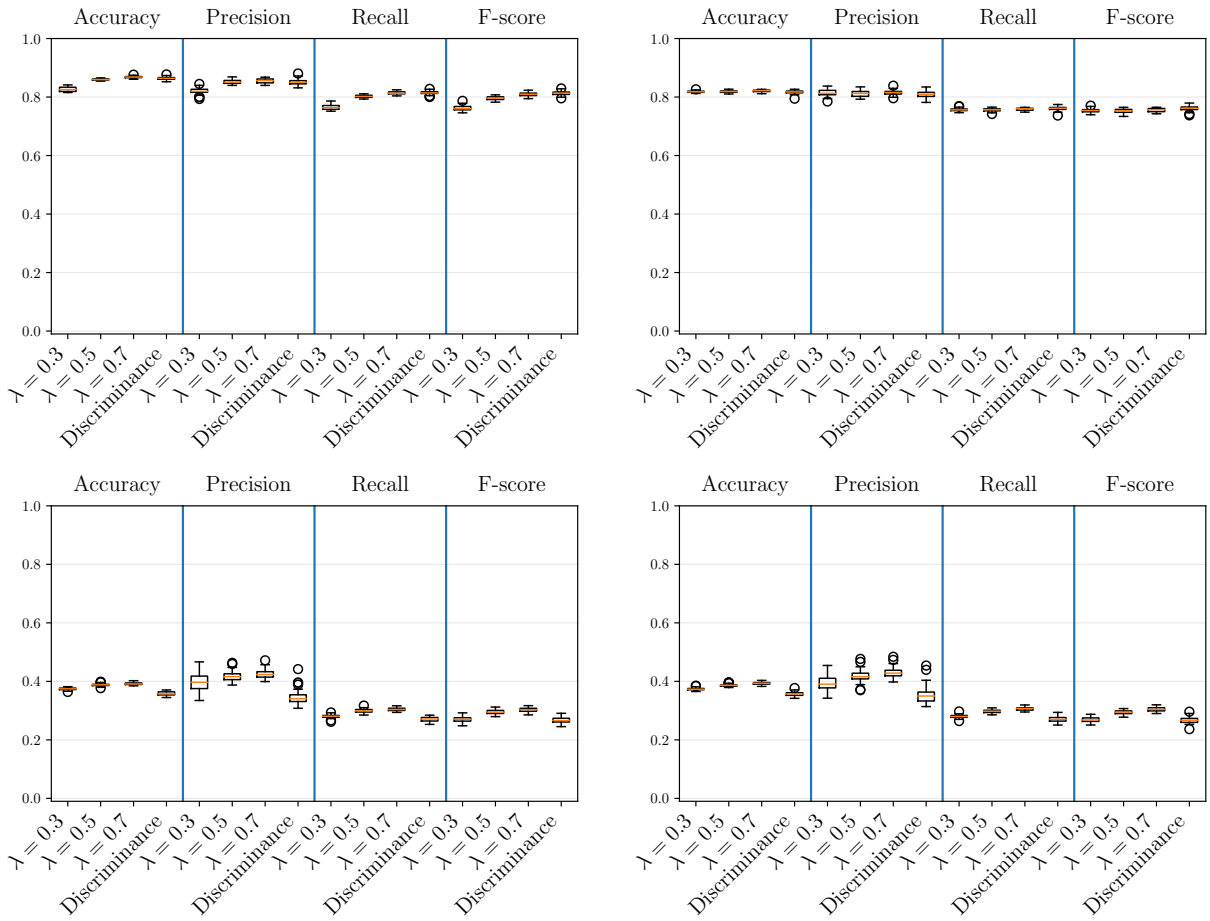



Figure 12: Classification results for INEX 2005 (top) and INEX 2006 (bottom) as ordered (left) and unordered (right) trees.

the size of trees. Contrary to the videogame sellers dataset that had the same property, the classification does not achieve 100% of good classification, as depicted in Fig. 14 (right). On average, discriminance performs better than the other weights, despite having a larger variance. This is probably due to the small size of the dataset, as the discriminance is learned only with around 13 trees per class.

**Hicks et al. cell lineage trees** Across cellular division, tracking the lineage of a single cell naturally defines a tree. In a recent article, Hicks et al. have been investigating the variability inside cell lineages trees of three different species [19]. From the encoding of the data that they have provided as a supplementary material[3], we have extracted ordered unlabeled trees that are presented in Fig. 15 (left). The dataset contains, for two classes, trees of outdegree 0 (i.e., isolated leaves) that can be considered as noise. With respect to the exponential weight, the value of the kernel between such trees will be identical, whether they belong to the same class or to two different classes. They therefore contribute to reducing the kernel's ability to effectively discriminate between these two

[3] https://doi.org/10.1101/267450 (last accessed in October 2019)

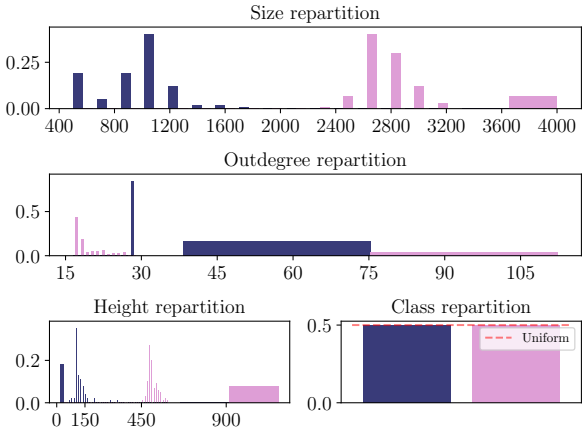



Figure 13: Description of the videogame sellers dataset (200 trees).

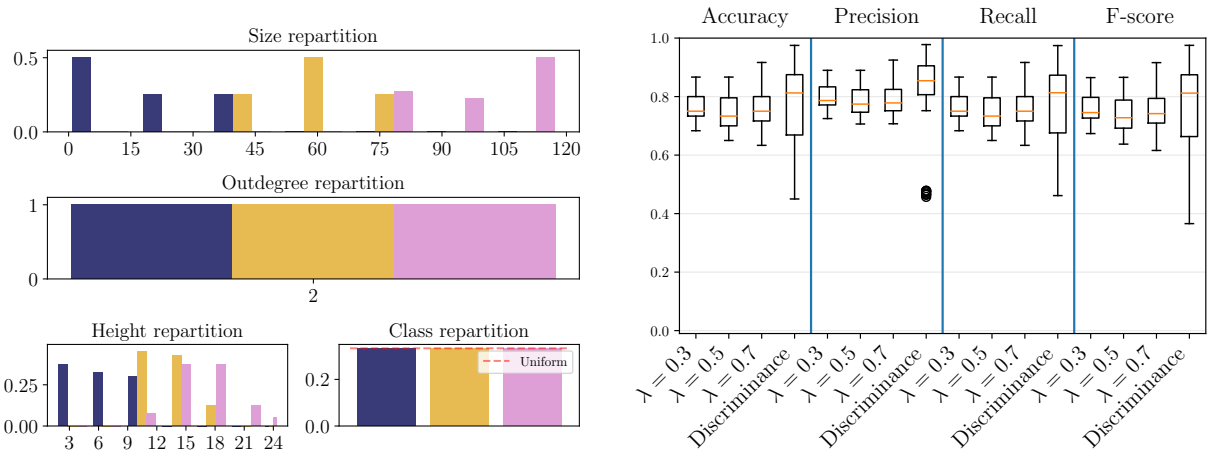



Figure 14: Description of the Vascusynth dataset (120 trees, left) and classification results (right).

classes. On the other hand, the discriminance weight will assign them a zero value, "de-noising", in a way, the data. This observation may explain why discriminance weight achieves better results than exponential weight.

**Faure et al. cell lineage trees** Faure et al. have developed a method to construct cell lineage trees from microscopy [14] and provided their data online[4]. We extracted 300 unordered and unlabeled trees, divided between three classes. It seems from Fig. 16 (left) that one class among the three can be distinguished from the two others. Classification results can be found in Fig. 16 (right): the discriminance weight performs better than the exponential weight, whatever the value of the parameter.

[4] https://bioemergences.eu/bioemergences/openworkflow-datasets.php (last accessed in October 2019)

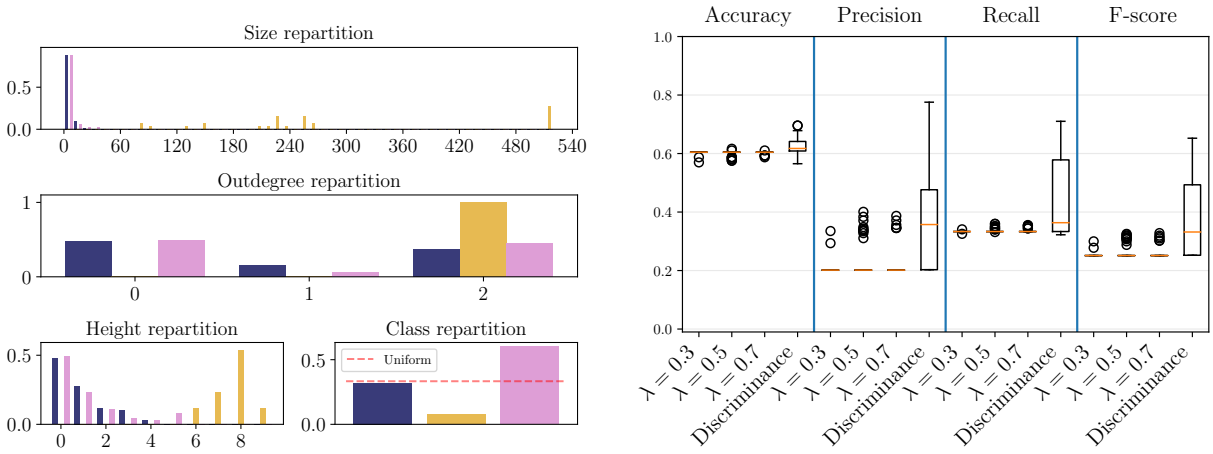



Figure 15: Description of the Hicks et al. dataset (345 trees, left) and classification results (right).

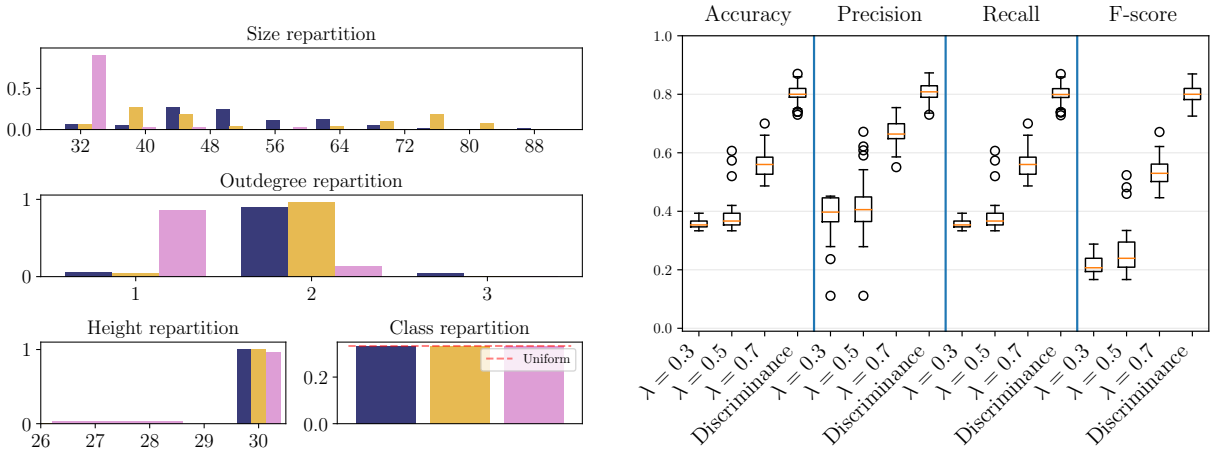



Figure 16: Description of the Faure et al. dataset (300 trees, left) and classification results (right).

## 5.5 LOGML

The LOGML dataset is made of user sessions on an academic website, namely the Rensselaer Polytechnic Institute Computer Science Department website[5], that registered the navigation of users across the website. 23 111 unordered labeled trees are present, divided into two classes. The trees are very alike, as shown in Fig. 17 (left), and the classification results of Fig. 17 (right) are very similar to INEX 2005, where all weight functions behave similarly, without any advantage for the discriminance weight in terms of prediction.

[5] https://science.rpi.edu/computer-science (last accessed in October 2019)

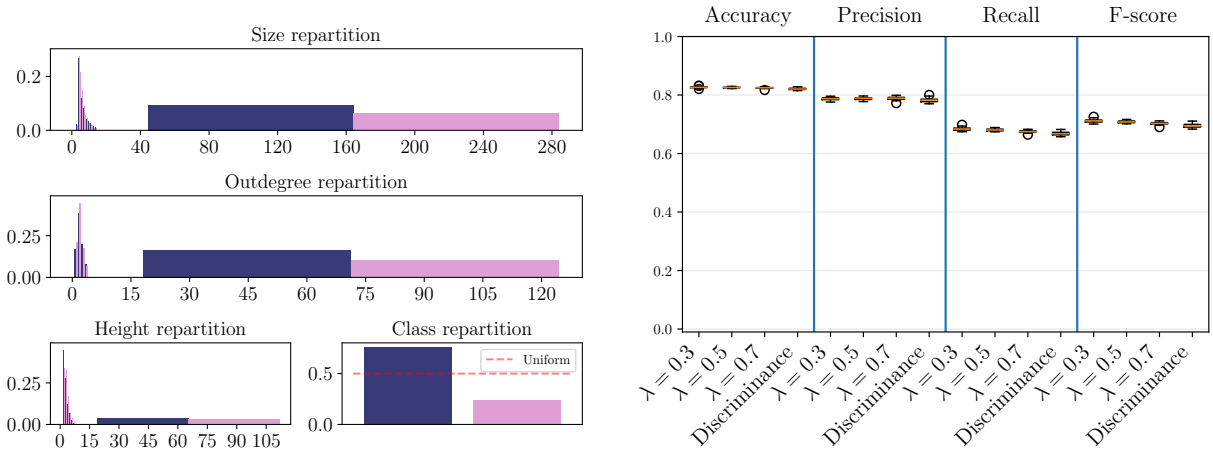



Figure 17: Description of the LOGML dataset (23 111 trees, left) and classification results (right).

# 6 Concluding remarks

## 6.1 Main interest of the DAG approach: learning the weight function

In Section 2, we have shown on a 2-classes stochastic model that the efficiency of the subtree kernel is improved by imposing that the weight of leaves is null. As explained in Remark 2.5, we conjecture that the weight of any subtree present in two different classes should be 0. The main interest of the DAG approach developed in Section 3 is that it allows to learn the weight function from the data, as developed in Section 4 with the discriminance weight function. Our method has been implemented and tested in Section 5 on eight real datasets with very different characteristics that are summed up in Table 1.

As a conclusion of our experiments, we have analyzed the relative improvement in prediction obtained with the discriminance weight against the best exponential weight in order to show both the importance of the weight function and the relevance of the method developed in this paper. More precisely, for each dataset and each classification metric, we have calculated

$$\mathrm{RI} = \frac{\mathrm{Metric}_{\mathrm{discr}} - \max(\mathrm{Metric}_\lambda)}{\max(\mathrm{Metric}_\lambda)},$$

from the average values of the different metrics. The results are presented in Fig. 18. We have found that, except in one case, discriminance behaves as good as exponential weight decay and even performs better in most of the datasets. Furthermore, one can observe a kind of trend, where the relative improvement decreases when the number of trees in the training dataset is increasing, which proves the great interest of the discriminance to handle small datasets, provided that (i) the problem is difficult enough that the exponential weights are not already high performing, as it is the case in the Videogames sellers dataset, and (ii) the dataset is not too small, as for Vascusynth. Indeed, as the discriminance is learned independently from the SVM, one must have enough training data to divide them efficiently. Nevertheless, it should be noted that, in the framework of the DAG approach, results from the discriminance weight can be obtained much faster due to the fact that the Gram matrices are estimated from one half of the training dataset, while learning

the discrimance is very fast as it can be done in one traversal of the DAG (see time-complexity presented in Remark 4.1). Finally, we have investigated on a single example some properties of the discriminance, discovering that it can be interpreted as a second-order exponential weight, as well as a method for visualizing the important features in the data.

| Dataset | Wikipedia | Videogames | INEX 2005 | INEX 2006 | Vascusynth | Hicks et al. | Faure et al. | LOGML |
|---|---|---|---|---|---|---|---|---|
| Ord. / Unord. | Both | Ord. | Both | Both | Unord. | Ord. | Unord. | Unord. |
| labeled | ✗ | ✓ | ✓ | ✓ | ✓ | ✗ | ✗ | ✓ |
| Number of trees | 160 – 320 | 200 | 9 630 | 12 107 | 120 | 345 | 300 | 23 111 |
| Number of classes | 4 | 2 | 11 | 18 | 3 | 3 | 3 | 2 |

Table 1: Summary of the 8 datasets.

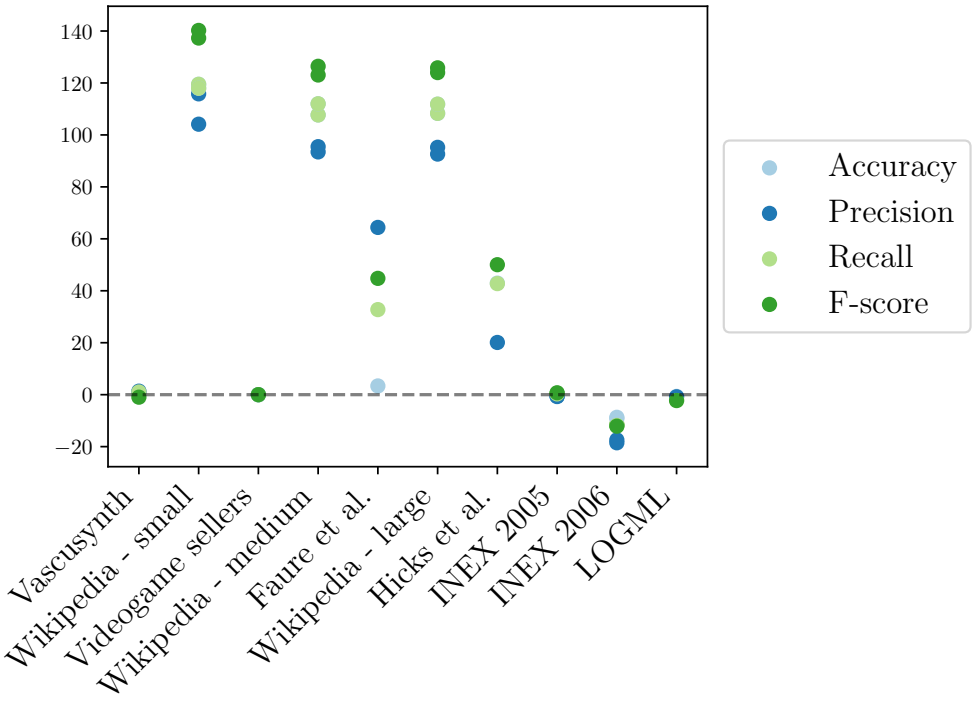



Figure 18: Relative improvement RI (in percentage) of the discriminance against the best value of λ for all datasets (sorted by increasing number of trees in the training dataset) and all metrics.

### 6.2 Interest of the DAG approach in terms of computation time

As shown in Fig. 16 (right), the exponential decay classification results for the Faure et al. dataset are very dependent on the value chosen for the parameter λ. In this case, it can be interesting to tune this parameter and estimate its best value with respect to a prediction score. This requires to

compute the Gram matrices from different weight functions. We present in Fig. 19 the computation time required to compute the Gram matrices from a given number of values of the parameter. As expected from the theoretical results, we observe a linear dependency: the intercept corresponds to the computation time required to compute and annotate the DAG reduction, while the slope is associated with the time required to compute the Gram matrices, which is proportional to the average of $\mathcal{O}(\min(\#T_i, \#T_j))$ (see Remark 3.8). This can be compared to the time-complexity of the algorithm developed in [30] which is the average of $\mathcal{O}(\#T_i + \#T_j)$. Consequently, the corresponding computation times should be proportional to at least twice the slope that we observe with the DAG approach. This shows another interest of our method that is not related to the discriminance weight function. It should be faster to compute several repetitions of the subtree kernel from the DAG approach than from the previous algorithm [30] provided that the number of repetitions is large enough.

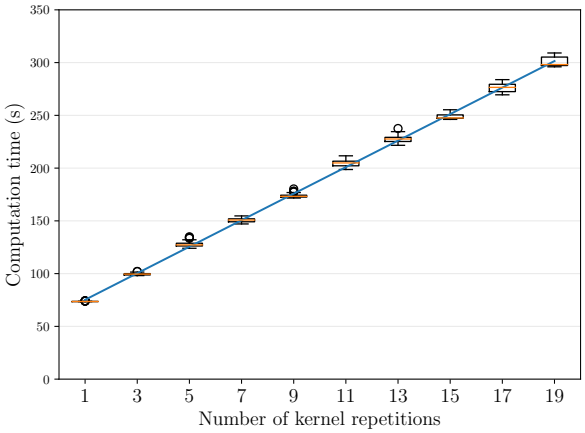



Figure 19: Computation time required to compute several repetitions of the kernel on the Faure et al. dataset. All those calculations have been repeated 50 times for each number of repetitions. The intercept corresponds to the DAG compression of the dataset, which is independent on the number of repetitions. The blue curve is a linear fitting of all the measurement points.

### 6.3 Implementation and reproducibility

The `treex` library for Python [2] is designed to manipulate rooted trees, with a lot of diversity (ordered or not, labeled or not). It offers options for random generation, visualization, edit operations, conversions to other formats, and various algorithms. We implemented the subtree kernel as a module of `treex` so that the interested reader can manipulate the concepts discussed in this paper in a ready-to-use manner.

Basically, the `subtree_kernel` module allows the computation of formula (2) with options for choosing (i) $\kappa$ among some classic choices of kernels [25, Section 2.3] and (ii) the weight function among exponential decay or discriminance. Resorting to dependencies to `scikit-learn`, tools for processing databases and compute SVM are also provided for the sake of self-containedness. Finally, visualization tools are also made available to perform the comprehensive learning approach discussed in Subsection 5.2.

Installing instructions and the documentation of `treex` can be found from [2]. For the sake of reproducibility, the databases used in Section 5, as well as the scripts that were designed to create them and process them, have been made available on the first author webpage[6].

[6] http://perso.ens-lyon.fr/romain.azais/subtree-kernel/

## Acknowledgments

This work has been supported by the European Union's H2020 project ROMI. The authors would like to thank Dr. Bruno Leggio as well as Dr. Giovanni Da San Martino for helping them to access some of the datasets presented in the paper: *grazie*. Last but not least, the authors are grateful to three anonymous reviewers for their valuable comments on a first version of the manuscript.

## References

[1] Fabio Aiolli, Giovanni Da San Martino, Alessandro Sperduti, and Alessandro Moschitti. Fast on-line kernel learning for trees. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 787–791. IEEE, 2006.

[2] Romain Azaïs, Guillaume Cerutti, Didier Gemmerlé, and Florian Ingels. treex: a python package for manipulating rooted trees. *The Journal of Open Source Software*, 4, 2019.

[3] Romain Azaïs, Alexandre Genadot, and Benoît Henry. Inference for conditioned Galton-Watson trees from their Harris path. *To appear in ALEA*, 2019.

[4] Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008.

[5] Karthik Bharath, Prabhanjan Kambadur, Dipak Dey, Rao Arvin, and Veerabhadran Baladandayuthapani. Statistical tests for large tree-structured data. *Journal of the American Statistical Association*, 2016.

[6] Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217 – 239, 2005.

[7] Michael Collins and Nigel Duffy. Convolution kernels for natural language. In *Advances in neural information processing systems*, pages 625–632, 2002.

[8] Gianni Costa, Giuseppe Manco, Riccardo Ortale, and Andrea Tagarelli. A tree-based approach to clustering xml documents by structure. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Knowledge Discovery in Databases: PKDD 2004*, pages 137–148, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[9] Nello Cristianini, John Shawe-Taylor, et al. *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[10] Giovanni Da San Martino. *Kernel methods for tree structured data*. PhD thesis, alma, 2009.

[11] Ludovic Denoyer and Patrick Gallinari. Report on the xml mining track at inex 2005 and inex 2006: categorization and clustering of xml documents. In *SIGIR Forum*, volume 41, pages 79–90, 2007.

[12] Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *J. ACM*, 27(4):758–771, October 1980.

[13] David S. Ebert and F. Kenton Musgrave. *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[14] E Faure, T Savy, B Rizzi, C Melani, M Remešíkova, R Špir, O Drblíková, R Čunderlík, G Recher, B Lombardot, et al. An algorithmic workflow for the automated processing of 3d+ time microscopy images of developing organisms and the reconstruction of their cell lineage. *Nat. Commun*, 2015.

[15] Christophe Godin and Pascal Ferraro. Quantifying the degree of self-nestedness of trees: application to the structural analysis of plants. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 7(4):688–703, 2010.

[16] Ghassan Hamarneh and Preet Jassi. Vascusynth: Simulating vascular trees for generating volumetric image data with ground truth segmentation and tree analysis. *Computerized Medical Imaging and Graphics*, 34(8):605–616, 2010.

[17] John C. Hart and Thomas A. DeFanti. Efficient antialiased rendering of 3-d linear fractals. *SIGGRAPH Comput. Graph.*, 25(4):91–100, July 1991.

[18] David Haussler. Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California, 1999.

[19] Damien G Hicks, Terence P Speed, Mohammed Yassin, and Sarah M Russell. Maps of variability in cell lineage trees. *PLoS computational biology*, 15(2):e1006745, 2019.

[20] Preet Jassi and Ghassan Hamarneh. Vascusynth: Vascular tree synthesis software. *Insight Journal*, January-June:1–12, 2011.

[21] Daisuke Kimura, Tetsuji Kuboyama, Tetsuo Shibuya, and Hisashi Kashima. A subpath kernel for rooted unordered trees. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 62–74. Springer, 2011.

[22] Shu-Yun Le, Ruth Nussinov, and Jacob V. Maizel. Tree graphs of RNA secondary structures and their comparisons. *Computers and Biomedical Research*, 22(5):461 – 473, 1989.

[23] M. A. Martín-Delgado, J. Rodriguez-Laguna, and G. Sierra. Density-matrix renormalization-group study of excitons in dendrimers. *Phys. Rev. B*, 65:155116, Apr 2002.

[24] James Mercer. Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446, 1909.

[25] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA, 2001.

[26] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[27] Dan Shen, Haipeng Shen, Shankar Bhamidi, Yolanda Muñoz Maldonado, Yongdai Kim, and J. Stephen Marron. Functional data analysis of tree data objects. *Journal of computational and graphical statistics : a joint publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America*, 23 2:418–438, 2014.

[28] Steven S Skiena. Sorting and searching. In *The Algorithm Design Manual*, pages 103–144. Springer, 2012.

[29] Ivan E. Sutherland. Sketchpad: A man-machine graphical communication system. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, AFIPS '63 (Spring), pages 329–346, New York, NY, USA, 1963. ACM.

[30] S.V.N. Vishwanathan and Alexander J Smola. Fast kernels on strings and trees. *Advances on Neural Information Proccessing Systems*, 14, 2002.

[31] Haonan Wang and J. S. Marron. Object oriented data analysis: Sets of trees. *Ann. Statist.*, 35(5):1849–1873, 10 2007.

# A Proof of Proposition 2.2

The proof is mainly based on the following technical lemma, which statement requires the following notation. If $u$ is a vertex of a tree $T$, $\mathcal{F}(u)$ denotes the family of $u$, i.e., the set composed of the ascendants of $u$, $u$, and the descendants of $u$ in $T$. We recall that $\mathcal{D}(u)$ stands for the set of descendants of $u$.

**Lemma A.1.** *Let $u, v \in T_i$, $i \in \{1, 2\}$. One has*

$$K(T_i^u, T_i^v) = K(T_i, T_i) - \sum_{x \in \mathcal{B}_{u,v}} \omega_{T_i[x]} + K(\tau_{\mathcal{H}(u)}, \tau_{\mathcal{H}(v)}),$$

*where*

$$\mathcal{B}_{u,v} = \begin{cases} \mathcal{D}(u) \cup \{u\} & \text{if } u = v, \\ \mathcal{F}(u) \cup \mathcal{F}(v) & \text{else.} \end{cases} \tag{6}$$

*Let $u \in T_1$ and $v \in T_2$. Then,*

$$K(T_1^u, T_2^v) = K(\tau_{\mathcal{H}(u)}, \tau_{\mathcal{H}(v)}).$$

**Proof.** We begin with the case $u \neq v$. The result relies on the following decomposition which is valid under the assumptions made on $T_i$ and the sequence $(\tau_h)$,

$$\mathcal{S}(T_i^u) \cap \mathcal{S}(T_i^v) \;=\; \left[\mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}\right] \;\cup\; \left[\mathcal{S}(\tau_{\mathcal{H}(u)}) \cap \mathcal{S}(\tau_{\mathcal{H}(v)})\right].$$

Together with (2),

$$\begin{aligned} K(T_i^u, T_i^v) \;&=\; \sum_{\theta \in \mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}} w_\theta N_\theta(T_i^u) N_\theta(T_i^v) \\ &+ \sum_{\theta \in \mathcal{S}(\tau_{\mathcal{H}(u)}) \cap \mathcal{S}(\tau_{\mathcal{H}(v)})} w_\theta N_\theta(T_i^u) N_\theta(T_i^v). \end{aligned}$$

If $\theta \in \mathcal{S}(\tau_{\mathcal{H}(u)}) \cap \mathcal{S}(\tau_{\mathcal{H}(v)})$, then $N_\theta(T_i^z) = N_\theta(\tau_{\mathcal{H}(z)})$, $z \in \{u, v\}$, because, for any $h > 0$, $\tau_h$ is not a subtree of $T_0$ nor $T_1$ by assumption. Thus,

$$\begin{aligned} \sum_{\theta \in \mathcal{S}(\tau_{\mathcal{H}(u)}) \cap \mathcal{S}(\tau_{\mathcal{H}(v)})} w_\theta N_\theta(T_i^u) N_\theta(T_i^v) \;&=\; \sum_{\theta \in \mathcal{S}(\tau_{\mathcal{H}(u)}) \cap \mathcal{S}(\tau_{\mathcal{H}(v)})} w_\theta N_\theta(\tau_{\mathcal{H}(u)}) N_\theta(\tau_{\mathcal{H}(v)}) \\ &=\; K(\tau_{\mathcal{H}(u)}, \tau_{\mathcal{H}(v)}), \end{aligned} \tag{7}$$

in light of (2) again. Furthermore, if $\theta \in \mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}$, then $N_\theta(T_i^z) = N_\theta(T_i)$, $z \in \{u, v\}$, and

$$\begin{aligned}\sum_{\theta \in \mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}} w_\theta N_\theta(T_i^u) N_\theta(T_i^v) &= \sum_{\theta \in \mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}} w_\theta N_\theta(T_i) N_\theta(T_i) \\ &= \sum_{\theta \in \mathcal{S}(T_i)} w_\theta N_\theta(T_i) N_\theta(T_i) \\ &\quad - \sum_{\theta \in \{T_i[u] \,:\, u \in \mathcal{F}(v) \cup \mathcal{F}(w)\}} w_\theta N_\theta(T_i) N_\theta(T_i) \\ &= K(T_i, T_i) - \sum_{\theta \in \{T_i[z] \,:\, z \in \mathcal{F}(u) \cup \mathcal{F}(v)\}} w_\theta, \end{aligned} \tag{8}$$

since $N_\theta(T_i) = 1$ because of the first assumption on $T_i$. (7) and (8) state the first result. When $u = v$, the decomposition is slightly different,

$$\mathcal{S}(T_i^u) \;=\; \left[\mathcal{S}(T_i) \setminus \{T_i[z] \,:\, z \in \{u\} \cup \mathcal{D}(u)\}\right] \;\cup\; \mathcal{S}(\tau_{\mathcal{H}(u)}),$$

but the rest of the proof is similar. Finally, the formula for $K(T_1^u, T_2^v)$ is a direct consequence of the third assumption on $T_1$, $T_2$ and the sequence $(\tau_h)$.

By virtue of the previous lemma, one can derive the following result on the quantity $\Delta_x^i$ defined by (3).

**Lemma A.2.** *Let $x \in T_i$, $i \in \{1, 2\}$. One has*

$$\Delta_x^i = K(T_i, T_i) - \mathbb{E}_u\left[\sum_{z \in \mathcal{B}_{x,u}} w_{T_i[z]}\right].$$

**Proof.** In light of Lemma A.1, one has

$$\Delta_x^i = K(T_i, T_i) - \mathbb{E}_u\left[\sum_{z \in \mathcal{B}_{x,u}} w_{T_i[z]}\right] + \mathbb{E}_u\left[K(\tau_{\mathcal{H}(x)}, \tau_{\mathcal{H}(u)})\right] - \mathbb{E}_v\left[K(\tau_{\mathcal{H}(x)}, \tau_{\mathcal{H}(v)})\right].$$

By assumption on the stochastic model of random trees, $\mathcal{H}(u)$ and $\mathcal{H}(v)$ have the same distribution and thus $\mathbb{E}_u[K(\tau_{\mathcal{H}(x)}, \tau_{\mathcal{H}(u)})] = \mathbb{E}_v[K(\tau_{\mathcal{H}(x)}, \tau_{\mathcal{H}(v)})]$, which states the expected result.

The next decomposition is useful to prove the result of interest. If $c_h^i$ denotes the number of subtrees of height $h$ appearing in $T_i$, $h \geq 0$, then the probability of picking a particular vertex $u$ is $P_\rho(\mathcal{H}(u))/c^i_{\mathcal{H}(u)}$ and thus

$$\mathbb{E}_u\left[\sum_{z \in \mathcal{B}_{x,u}} w_{T_i[z]}\right] = \frac{P_\rho(\mathcal{H}(x))}{c^i_{\mathcal{H}(x)}} \sum_{z \in \{x\} \cup \mathcal{D}(x)} w_{T_i[z]} + \sum_{u \in T_i \setminus \{x\}} \frac{P_\rho(\mathcal{H}(u))}{c^i_{\mathcal{H}(u)}} \sum_{z \in \mathcal{B}_{x,u}} w_{T_i[z]}.$$

In addition, for $u \in T_i \setminus \{x\}$,

$$\sum_{z \in \{x\} \cup \mathcal{D}(x)} \omega_{T_i[z]} \;=\; K(T_i[x], T_i[x]), \tag{9}$$

$$\sum_{z \in \mathcal{B}_{x,u}} \omega_{T_i[z]} \;=\; K(T_i, T_i) - \sum_{z \notin \mathcal{F}(x) \cup \mathcal{F}(u)} \omega_{T_i[z]}. \tag{10}$$

(9) and (10) together with Lemma A.2 show that

$$\Delta_x^i = \frac{P_\rho(\mathcal{H}(x))}{c^i_{\mathcal{H}(x)}} \left(K(T_i, T_i) - K(T_i[x], T_i[x]) + \sum_{u \in T_i \setminus \{x\}} \frac{P_\rho(\mathcal{H}(u))}{c^i_{\mathcal{H}(u)}} \sum_{z \notin \mathcal{F}(x) \cup \mathcal{F}(u)} \omega_{T_i[z]}.\right.$$

The left-hand term (and the right-hand term when $w_{T_i} > 0$) is null if and only if $x = \mathcal{R}(T_i)$, which shows the first result. In addition,

$$\Delta_x^i \geq \frac{P_\rho(\mathcal{H}(x))}{c^i_{\mathcal{H}(x)}} \left(K(T_i, T_i) - K(T_i[x], T_i[x])\right),$$

which states the expected formula (4) with $P_\rho(0) \leq P_\rho(\mathcal{H}(x))$ (true if $\rho > H/2$) and $c^i_{\mathcal{H}(x)} \leq \#\mathcal{L}(T_i)$. The conclusion comes from the fact that the probability of drawing a vertex $x$ of height greater than $h$ is $G_\rho(h)$.

# B Proof of Proposition 3.2

We denote by $D^h$ the set of vertices at height $h$ in any DAG $D$, and $* \in \{\text{ordered}, \text{unordered}\}$ the type of isomorphism considered. From the forest $(D_1, \ldots, D_N)$, we construct the DAG $\Delta$ such that (i) $D_i$ is a subDAG of $\Delta$ for all $i$, (ii) $\mathcal{H}(\Delta) = \max_i \mathcal{H}(D_i)$, (iii) all vertices in $\Delta$ have degree $\max_i \deg(D_i)$, and (iv) at each height except 0 and $\mathcal{H}(D)$, $\#\Delta^h = \max_i \#D_i^h$. If $\Delta$ is placed $N$ times under an artificial root, and then recompressed by the algorithm, indeed the output contains the recompression of the original forest. Therefore, this case is the worst possible for the algorithm, and we claim that it achieves the proposed complexity.

Let $\Delta$ be now a DAG with following properties : $\#\Delta = m$, $\mathcal{H}(\Delta) = H$, at each height $h \notin \{0, H\}$, $\#\Delta^h = n$ (so that $n(H-2)+2 = m$), and all vertices have degree $d$. $\mathbb{D}_\mathcal{F}$ is the super-DAG obtained after placing $N$ copies of $\Delta$ under an artificial root. We then have $\#\mathbb{D}_\mathcal{F} = 1 + Nm$ so that $\mathcal{O}(\#\mathbb{D}_\mathcal{F}) = \mathcal{O}(Nm) = \mathcal{O}(NHn)$ and $\deg(\mathcal{F}) = \deg(\Delta) = d$.

At the beginning of the algorithm, constructing the mapping $h \mapsto \mathbb{D}_\mathcal{F}{}^h$ in one exploration of $\mathbb{D}_\mathcal{F}$ has complexity $\mathcal{O}(\#\mathbb{D}_\mathcal{F})$. We will now examine the complexity of the further steps, with respect to $n, m, d, H$ and $N$. We introduce the following lemma :

**Lemma B.1.** *Constructing $\sigma(h)$ has time-complexity:*

1. $\mathcal{O}\big(\sum_{v \in \mathbb{D}_\mathcal{F}{}^h} \#\mathcal{C}(v) \log \#\mathcal{C}(v)\big)$ *for unordered trees;*
2. $\mathcal{O}\big(\sum_{v \in \mathbb{D}_\mathcal{F}{}^h} \#\mathcal{C}(v)\big)$ *for ordered trees.*

**Proof.** When sorting lists of size $L$, merge sort is known to have $\mathcal{O}(L \log L)$ complexity in the worst case [28]. Accordingly, we introduce

$$g^*(x) = \begin{cases} x & \text{if } * = \text{ordered;} \\ x(1 + \log x) & \text{if } * = \text{unordered.} \end{cases}$$

At height $h$, we construct $\sigma(h) = \{f^{-1}(S) \,:\, S \in \mathrm{Im}(f), \#f^{-1}(S) \geq 2\}$ where $f : v \in \mathbb{D}_\mathcal{F}{}^h \mapsto \mathcal{C}(v)$. Finding the preimage of $f$ requires first to construct $f$, by copying the children of each vertex in

$\mathbb{D}_{\mathcal{F}^h}$ (in the unordered case, we also need to sort them, so that we get rid of the order and can properly compare them). Then we only need to explore once the image and check whether an element has two or more antecedents. The global cost is then $\mathcal{O}(\sum_{v\in\mathbb{D}_{\mathcal{F}^h}} g^*(\#\mathcal{C}(v)))$.

We reuse the notation $g^*$ from the proof of Lemma B.1. With respect to $\Delta$, the complexity for constructing $\sigma(\cdot)$ is $\mathcal{O}(Nng^*(d))$. Exploring the elements of $\sigma(h)$ for (i) choosing a vertex $v_M$ to remain, and (ii) delete the other elements $\delta_M$ has complexity $\mathcal{O}(Nn)$. In addition, at height $h' > h$, exploring the children to replace them or not costs $\mathcal{O}(\sum_{v\in\mathbb{D}_{\mathcal{F}^{h'}}} \#\mathcal{C}(v)) = \mathcal{O}(Ndn)$.

The global complexity $C(\mathbb{D}_{\mathcal{F}})$ of the algorithm is then

$$C(\mathbb{D}_{\mathcal{F}}) = \mathcal{O}(\#\mathbb{D}_{\mathcal{F}}) + \sum_{h=0}^{\mathcal{H}(\Delta)} \mathcal{O}(Nng^*(d)) + \mathcal{O}(Nn) + \sum_{h'>h} \mathcal{O}(Ndn).$$

Remark that $\sum_{h=0}^{\mathcal{H}(D)} \mathcal{O}(Nn) = \mathcal{O}(Nm) = \mathcal{O}(\#\mathbb{D}_{\mathcal{F}})$, this leads to

$$C(\mathbb{D}_{\mathcal{F}}) = \mathcal{O}(\#\mathbb{D}_{\mathcal{F}}\, g^*(\deg(\mathcal{F}))) + \mathcal{O}\left(Ndn \sum_{h=0}^{\mathcal{H}(D)} \sum_{h'>h} 1\right).$$

The right-hand inner sum is in $\mathcal{O}(H^2)$. As

$$\mathcal{O}(NdnH^2) = \mathcal{O}(\#\mathbb{D}_{\mathcal{F}}\, Hd) = \mathcal{O}(\#\mathbb{D}_{\mathcal{F}}\, \mathcal{H}(\mathbb{D}_{\mathcal{F}})\deg(\mathcal{F})),$$

this leads to our statement.