MetaAP: a meta-tree-based ranking algorithm optimizing the average precision from imbalanced data

Rémi Viola, Léo Gautheron, Amaury Habrard, Marc Sebban

MetaAP: a meta-tree-based ranking algorithm optimizing the average precision from imbalanced data

Rémi Viola, Léo Gautheron, Amaury Habrard, Marc Sebban

October 23, 2022

Abstract

In this paper, we address the challenging problem of learning to rank from highly imbalanced data. This scenario requires to resort to specific metrics able to account the scarcity of the so-called positive examples. We present **MetaAP**, a tree-based ranking algorithm, which induces meta-trees by optimizing directly during the learning process the *Average Precision* (*AP*). This latter has been shown to be more relevant than the area under the *ROC* curve (*AUC-ROC*) when the objective is to push the examples of interest at the very top of the list. This effect of the *AP* in tree-based ranking is particularly wished to address fraud detection tasks where (i) the budget is often constrained (in terms of possible controls) and (ii) the interpretability of the induced models is required to support decision making. After an extensive comparative study on 28 public datasets showing that **MetaAP** is significantly better than other tree-based ranking methods, we tackle a tax fraud detection task coming from a partnership with the French Ministry of Economy and Finance. The results show that **MetaAP** is able to make the tax audit process much more efficient.

1 Introduction

Learning to rank from imbalanced datasets where positive examples are very scarce has received much attention during the past years from the machine learning community. Indeed, this challenging topic opened the door to many methodological questions: *Which loss function to optimize? Can we derive generalization guarantees? Are there ways to efficiently balance the datasets? How to rank imbalanced data under budget constraints? Can we induce interpretable models in this setting?* etc.

Fraud detection [1] falls into this scope of imbalanced learning to rank, where the number of fraudsters is small compared to the huge amount of normal cases (also called negative examples). Fraud detection has become a key issue for e-commerce companies and government agencies which are facing a tremendous growth of the data collected that have to be processed by a relatively limited number of human controllers. Therefore, fraud detection is subject nowadays to a compelling need for automatic and interpretable systems for supporting human decision making. Unlike a standard anomaly detection task [2] where an abnormal data often takes the form of an outlier, the peculiarity of fraud detection is that fraudsters often aim to mimic a normal behavior that makes the identification much more challenging.

One way to address this task is to resort to *sampling* strategies. While oversampling/data augmentation techniques can be used to generate dummy data artificially like in SMOTE-based methods [8] or in adversarial approaches [12], undersampling aims at removing irrelevant samples from the majority class as done with Tomek's Link [29] or in ENN [30] (see also [21, 20]). In [18], the authors investigate the strengths and weaknesses of both approaches and explain why oversampling is usually reported to outperform undersampling thanks to its capacity to produce a larger proportion of so-called safe examples. Even though a sampling method has the indisputable advantage of (re)balancing the datasets and allowing then the use of classical learning algorithms, in highly imbalanced scenarios, these methods do not succeed in generating enough diversity for improving significantly the results compared to the required effort, as recently shown in [7]. Other techniques to deal with the class imbalance problem include *cost-sensitive methods* [13] which nevertheless require a difficult tuning of the miss-classification costs, *(deep) metric learning methods* [14, 24, 19] which often requires a large amount of data and/or a costly optimization process, or *boosting-based models* [9, 17, 15] which are not easily interpretable. It turns out that this latter property is key in fraud detection. Indeed, the detected suspicious cases are typically sent as alerts to the control department according to their position in the ranking, *i.e.* their probability of being a fraud. These top-ranked cases that are judged as frauds by the automatic system are then meticulously checked by a human controller whose analysis needs to be guided by the criteria that led to this decision. It is therefore often crucial for the prediction model to be supported by explainable decisions. As mentioned in [28], it is key to create methods that are readily interpretable rather than creating black boxes that will have to be explained later, often in imprecise ways.

In this context, decision trees seem to provide a good trade-off between accuracy and interpretability, beyond their natural capacity to deal with both quantitative and qualitative features. However, in order to address the issues induced by imbalanced datasets, they have to be optimized according to criteria that are able to take into account the scarcity of the positive examples compared to the large amount of negative samples. Moreover, from a learning to rank perspective, the decision tree should be learned as a scoring function projecting the data onto the real line. This is the goal of *TreeRank*, introduced in the seminal work of Clemençon and Vayatis [11] and exploited in several variants since then. *TreeRank* recursively maximizes the area under the *ROC* curve (*AUC-ROC*) allowing the induction of a tree that optimizes the probability to rank a positive example above a negative one. As illustrated in this paper, *TreeRank* seems to behave better when facing imbalanced datasets than standard decision trees induced by using the classic Gini or Entropy criteria for recursively splitting the nodes. However, as pointed out by [6, 15], in applications where only the very top rank will be used because of budget constraints, like in fraud detection where the number of human controllers is limited, the *AUC-ROC* does not seem to be the most suitable criterion. Indeed, a *AUC-ROC*-based algorithm will put a lot of effort into the enhancement of the scoring of currently poorly ranked samples at the expense of the positive data that are already favorably positioned in the ranking. To overcome this issue, Frery et al. [15] have shown that the *Average Precision* (*AP*) is a more adapted metric when we are mainly interested in the top of the list, even if it is at the price of definitely dropping out positive examples that are a bit further down in the ranking.

Inspired by *TreeRank*, we design in this paper a new algorithm, called **MetaAP**, which optimizes the *AP* by building a tree of local trees, referred to as meta-tree. A natural but sub-optimal method (as seen later in the experimental part) to address this task would consist in optimizing the hyperparameters of the trees (depth, splitting criterion, etc.) according to the *AP*. The novelty of **MetaAP** comes from the direct optimization of this measure, viewed as a loss function, during

the learning process. However, this task is hard because the AP takes the form of a non convex (even with well-made surrogates) and non separable function (*i.e.* the loss for one point depends on the others). The originality of the proposed approach consists in exploiting the slope coefficient of the tangents to the (1-Precision)-Recall curve (which plays a key role in the definition of AP) to order and merge the leaves of the local tree. In this way, **MetaAP** aims at recursively optimizing the AP and presents the valuable property of generating compact and interpretable models. Our method is supported by an extensive comparative study on 28 public datasets showing that the smaller the proportion of positives we learn from, the better **MetaAP** outperforms the competing algorithms. This comment is confirmed when the $Precision@k$ (corresponding to the number of positives among the top k ordered samples) is used as the evaluation measure that shows that **MetaAP** is particularly adapted to applications where the number of possible controls is limited. We further analyze a random forest version of our method and compare it with classic random forests and gradient boosting. Finally, we study the behavior of **MetaAP** to address a tax fraud detection task coming from our partnership with the French Ministry of Economy and Finance. The results show that **MetaAP** is able to make the tax audit process more efficient.

The rest of this paper is organized as follows: Section 2 is devoted to the presentation of the notations and evaluation measures. Section 3 is dedicated to the state of the art. We introduce **MetaAP** in Section 4 and present the experiments in Section 5.

2 Notations and Evaluation Measures

Let us consider a binary supervised learning task, with a training set $\mathcal{S} = \{z_i = (x_i, y_i)\}_{i=1}^m$ composed of m labeled examples with $x_i \in \mathcal{X} = \mathbb{R}^d$, a feature vector and $y_i \in \mathcal{Y} = \{-1; +1\}$, a label. When $y_i = 1$ (*resp.* $y_i = -1$), x_i is a positive (*resp.* negative) example belonging to the minority (*resp.* majority) class. $\mathcal{S}$ is supposed to be independently and identically drawn according to an unknown joint distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

Considering that we address a learning to rank task on imbalanced datasets, the choice of the metric to be optimized is key. In such a setting, three measures are usually used: the *AUC-ROC*, the *Average Precision* (AP) and the $Precision@k$. Let us briefly introduce these three concepts.

The *ROC* curve is the representation of the True Positive Rate (TPR / *Recall*) versus the False Positive Rate (FPR / False alarm rate) at different thresholds. TPR measures the capacity of the model to retrieve positives while FPR corresponds to the proportion of false alarms (fraction of negatives predicted as positive). More formally,

$$TPR = Recall = \frac{TP}{TP + FN} \quad \text{and} \quad FPR = \frac{FP}{FP + TN} \tag{1}$$

where FP (*resp.* FN) is the number of false positives (*resp.* false negatives) and TP (*resp.* TN) is the number of true positives (*resp.* true negatives). The *AUC-ROC* corresponds to the area under this *ROC*-curve and measures the probability of the model to rank a positive example above a negative one. It can be computed as follows:

$$AUC\text{-}ROC = \frac{1}{PN} \sum_{i=1}^{P} \sum_{j=1}^{N} \mathbf{I}_{0.5}(f(x_i^+) - f(x_j^-)),$$

- P (*resp.* N) is the number of positives (*resp.* negatives) in $\mathcal{S}$,
- x_i^+ (*resp.* x_j^-) is the i^{th} (*resp.* j^{th}) positive (*resp.* negative) sample,

- f is the scoring function assigning a probability to be positive,
- $\mathbf{I}_{0.5}$ is an indicator function equal to 1 when $f(x_i^+) - f(x_j^-) > 0$, $\frac{1}{2}$ when $f(x_i^+) - f(x_j^-) = 0$ and 0 otherwise.

Another important tool when working with imbalanced data is the *Precision-Recall* curve. It represents the *Precision* as a function of *Recall* for different thresholds, where the former is defined as follows: $Precision = \frac{TP}{TP+FP}$. Unlike the *AUC-ROC*, the *Precision-Recall* curve considers via the *Precision* the confidence in a positive prediction which can play a key role in imbalanced scenarios. The *Average Precision* (*AP*) is a summary of this confidence as the area under the curve [4, 25] and can be analytically computed as follows: $AP = \frac{1}{P}\sum_{i=1}^{P} p(k_i)$, where $p(k_i)$ is the *Precision* at the rank k_i corresponding to the i^{th} positive in the ranking. Note that *AP* can also be defined as

$$AP = \sum_t (R_t - R_{t-1})P_t, \tag{2}$$

with R_t and P_t the *Recall* and *Precision*, respectively, at the t^{th} threshold of distinct prediction values. It has been shown in [6] that *AP* focuses more on the top of the ranking, contrary to the *AUC-ROC* which takes equally into account the entire ranking and tries to move up as many positives as possible. This phenomenon is illustrated in Figure 1 where two rankings composed of 4 positives (in blue) and 6 negatives (in grey) are compared in terms of *AUC-ROC* and *AP*. We can note that for both situations, $AUC\text{-}ROC = 0.5$, while *AP* is higher on the right than on the left case. Therefore *AUC-ROC* is not able to distinguish the two situations while *AP* prefers a scenario where two positives are ranked at the very top of the list even if this is at the expense of missing the two remaining positives. It turns out that this latter behavior can be very useful in situations where the constraints related to the application at hand require to focus on the first part of the ranking. This is the case when the budget in terms of number of allowed checks by human controllers is limited, like in bank or tax fraud detection.

The capacity of a system to optimize the number of positives at the very top of the ranking can be explicitly measured with a third criterion, called *Precision@k*. This measure corresponds to the number of positives in the top k of the ranking. As the choice of k for a given application can vary over time, so requiring to re-train the model, it is more convenient to directly optimize the *AP* as a surrogate of this measure. This is what we suggest to do in our tree-based algorithm. We will see in the experiments that maximizing *AP* presents the nice property of optimizing at a cheaper cost the *Precision@k*.

3 Related Work

In this section, we present different methods that can be used for learning to rank from imbalanced datasets by inferring tree-based models, in the form of decision trees, meta-trees, random forests and tree ensembles. While the first two directly provide explicit decision rules, random forests and tree ensembles learned by gradient boosting are not readily interpretable, so reducing their benefit in applications like fraud detection despite the fact that they constitute the state of the art.

Decision Trees Although decision trees (DT), like CART [22], ID3 [26] or C4.5 [27], have been originally designed to address classification tasks, the discrete predictions can be used to establish

Figure 1: Evaluation of the *AUC-ROC* and *AP* on two rankings. Blue (*resp.* grey) lines represent positive (*resp.* negative) samples. While *AUC-ROC* behaves similarly on both cases (*AUC-ROC* $= \frac{1}{2}$), the average precision is equal to 0.43 on the left and 0.68 on the right, illustrating that *AP* favors the ranking that put (at least some) positives at the very top of the list.

a ranking. The splitting decision of CART is based on the minimization of the *Gini* impurity I_G. Assuming that the class label takes its value in the discrete set $\{1, 2, ..., k\}$, and that f_i denotes the fraction of the elements of the set with label i, I_G is defined as $I_G = \sum_{i=1}^{k} f_i(1 - f_i) = 1 - \sum_{i=1}^{k} f_i^2$ which is minimal when the leaf is pure, *i.e.* only composed of samples of the same class. On the other hand, ID3 and C4.5 make use of the information gain, based on the Shannon entropy. This latter allows to measure the disorder in a set and thus to select the split threshold maximizing the information gain I_E defined as $I_E = -\sum_{i=1}^{k} f_i \log_2 f_i$.

Whatever the splitting criterion, once the decision tree is induced, several strategies can be applied to get a scoring function providing a ranking. First, the local probability at each leaf of being positive can be used as a continuous score. The ranking is obtained by sorting the leaves according to this score. On the other hand, in [3], the authors suggest to assign a score to each example of a leaf based on its distance to the boundaries of the hypercube representing the leaf and induced by the DT. The score is defined as the distance to the boundaries for every leaf with a majority of positive examples, or minus the distance with a majority of negative samples. As in Support Vector Machines, this signed distance is used to rank the examples. Finally, in [23], the authors introduce a confusion factor at each internal node and use it to weight the leaves contribution and get the probability of an example.

Note that all the aforementioned methods share the same property. The optimization of the ranking is not part of the learning process. It is obtained by a post-process after the induction of the tree. In order to be efficient when addressing imbalanced settings, note also that the hyperparameters of the previous DT-based methods can be tuned by maximizing the *AUC-ROC*, *AP* or *Precision@k*, as introduced in Section 2.

The next paragraph introduces *TreeRank* [11], a learning to rank algorithm that has been specifically designed to generate a ranking that directly optimizes the *AUC-ROC*.

Meta-Trees *TreeRank* [11] is a decision tree algorithm that comes with theoretical guarantees for the bipartite ranking problem. A tree of trees (referred to as a meta-tree) is learned by directly optimizing the *AUC-ROC*. The principle is as follows. First, *TreeRank* induces a classical decision tree using a splitting procedure that takes into account the class imbalance. Then, the leaves are ordered according to a criterion based on the *AUC-ROC* curve. More precisely, for each leaf, the ratio β/α is computed with β (*resp.* α) the number of positives (*resp.* negatives) in the leaf divided by the number of positives (*resp.* negatives) in the root node. By assigning the label +1 to the

considered leaf and -1 to all the others, we can notice that β corresponds to TPR and α to FPR as defined in Eq. 1. Ordering the leaves according to the ratio β/α is equivalent to sorting them according to the slope coefficient of the tangent at the origin of the ROC curve, which boils down to maximizing the AUC-ROC. Then the leaves are partitioned into two sets, *left* and *right*, so that the partition maximizes an entropy defined as $\beta' - \alpha'$ where β' (*resp.* α') is the TPR (*resp.* FPR) of the left leaves. By doing so, *TreeRank* aims at finding the point with coordinates (α', β') maximizing the area under the piecewise function going from the origin $(0, 0)$ to the point (α', β') and then to the point $(1, 1)$. The procedure is repeated by learning two new DTs, one for all leaves that fall on the left part and one for all the leaves that fall on the right. Finally, *TreeRank* assigns to each leaf a score based on the order in which the leaves were explored from left to right: the first leaf receives a score of 1 then the scores gradually decrease until the last leaf that receives a score of 1 divided by the total number of leaves.

Random Forests Random Forests (RF) [5] aim to overcome the risk of overfitting of DTs by using a collection of partially independent trees. Each tree in the forest is still learned from n training examples, but the latter are randomly drawn (with replacement) from the training set $\mathcal{S}$ (bootstrap method). A sampling method is also performed over the features. At each node, the splits are optimized according to a subset of features randomly selected from the original feature space. A majority vote is finally applied from the predictions of these DTs. Note that the reduction of the variance in RFs is obtained at the price of losing the interpretability of the induced model. In [10], the authors introduced *Ranking Forest*, a RF variant of TreeRank which aggregates Meta-trees and combines the corresponding rankings to optimize the AUC-ROC.

Gradient Tree Boosting (GB) GB [16] is an ensemble method that consists in aggregating DTs, fitted sequentially on the residuals of the linear combination obtained at the previous step. In [15], the authors use a surrogate of the *Average Precision* (AP) that can be directly optimized in XGBoost [9], considered as one of the currently most effective GB methods. The resulting algorithm has been shown to be very efficient to address a highly imbalanced bank fraud detection task. Like Random Forest, the drawback of GB comes from its difficulty to induce a model that is directly interpretable.

4 MetaAP

In this section, we present our algorithm **MetaAP**. Inspired by *TreeRank*, its peculiarity comes from the optimization of the *Average Precision* (AP) instead of the AUC-ROC. The objective is to benefit from the capacity of AP to focus more on the top of the ranking to induce an interpretable model useful in applications where the budget in terms of human controllers is limited.

Let n be the number of examples of a given node of the current tree and n^+ be the corresponding number of positive samples. Let us consider a splitting decision that would send n_l (*resp.* n_r) examples to the left (*resp.* right) child node. We aim at selecting the split that would maximize $n_l AP_{left} + n_r AP_{right}$ where AP_{left} and AP_{right} are two proxies of the *Average Precision*. AP_{left} is built so that all the examples in the left (*resp.* right) child are classified as $+1$ (*resp.* -1). Conversely, AP_{right} is constructed so that all the examples in the right (*resp.* left) child are classified as $+1$ (*resp.* -1). We further define n_l^+ and n_r^+ as the number of positive examples in

the left and right children. In this binary setting, three thresholds are used in the computation of AP_{left} with the associated *Recall* and *Precision* defined as follows:

threshold	t_0	t_1	t_2
Recall	0	$\frac{n_l^+}{n^+}$	1
Precision	1	$\frac{n_l^+}{n_l}$	$\frac{n^+}{n}$

Plugging these values in Equation (2), we get:

$$AP_{left} = \frac{n_l^+}{n^+}\frac{n_l^+}{n_l} + \left(1 - \frac{n_l^+}{n^+}\right)\frac{n^+}{n} = \frac{n_l^{+2}}{n^+ n_l} + \frac{n_r^+}{n}$$

and similarly $AP_{right} = \dfrac{n_r^{+2}}{n^+ n_r} + \dfrac{n_l^+}{n}$.

Once a decision tree is learned (called DTAP as shown in Figure 2, bottom left), we need to order the leaves as done in *TreeRank*. However, since we aim at maximizing AP, we have to work with the *Precision-Recall* (PR) curve instead of the *ROC* curve (see Section 2), which is more challenging for two main reasons. First, the PR curve is not increasing on $[0; 1]$ from 0 to 1. It decreases from 1 to a value corresponding to the positive rate of the dataset. To keep the same strategy as *TreeRank*, we rather minimize the area under the (1-P)R curve which boils down to maximizing the area under the PR curve (Figure 2, top right). Second, unlike the *ROC* curve, the (1-P)R is not a monotonically increasing function, having local maxima each time negatives are highly ranked. To overcome this issue, our ordering strategy will relegate the leaves leading to such a scenario to the end of the list preventing them from being merged on the left part of the meta-tree, the part that really matters to get a high AP. More precisely, we rank in ascending order the directing coefficients of the tangents to the (1-P)R curve at 0 by calculating for each leaf the $(1 - Precision)/Recall$ ratio (Figure 2, top left). This implies that leaves with a majority of positives will appear at the beginning of the list, those with the most positives first because leading to the lowest directing coefficient corresponding to a *Precision* close to 1. Finally, according to the obtained ranking, we merge the leaves into two subsets (the left and right parts of the meta-tree) by accumulating the positives and negatives according to the leaves ranking and then selecting the merging threshold maximizing the AP on the left (Figure 2, bottom right).

Note that similarly to DTs, it is possible to adapt our tree-based method to build random forests by training several times **MetaAP** from a certain number of bootstraps of the training set. We denote this adaptation **MetaAPForest** in the next section.

5 Experiments

In this section, we present the experiments conducted on 28 public datasets coming from the UCI[1] and KEEL[2] repositories. We also performed experiments on private datasets provided by the French Ministry of Economy and Finance (DGFiP).

[1] `https://archive.ics.uci.edu/ml/datasets.html`
[2] `https://sci2s.ugr.es/keel/datasets.php`

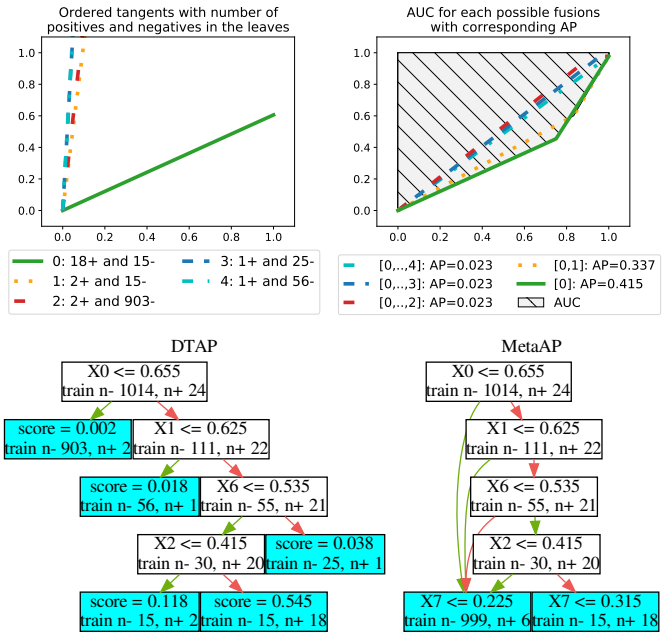

Figure 2: **MetaAP** run on *yeast*6 (one of the most imbalanced public dataset) with maximum depth of 4: (top left) Some tangents to the (1-P)R curve at 0 whose slopes are used to order the leaves. (top right) Corresponding approximation of the (1-P)R curve by a piece-wise function maximizing AP_{left} by accumulating the sorted leaves from left to right. MetaAP aims at maximizing the hatched area above the curve. Here $max(AP_{left}) = 0.415$ is achieved by putting the first leaf on the left and merging all the others on the right. (bottom left) Corresponding root's DT with splitting rules that maximize the AP. (bottom right) Corresponding merged DT.

5.1 Datasets and experimental setup

The main characteristics of the 28 public datasets are summarized in Table 1. The private datasets correspond to the tax and VAT declarations of French companies. They are used for the detection of (i) over-evaluated charges (called 1st fraud in the following) and (ii) international VAT frauds (2nd fraud). There are 40 datasets with an average of about 7,000 samples and 250 features. The imbalance ratio ranges from 0.3% to 24.3%. Note that each year, only 50,000 controls can be carried out from a panel of more than 3 million companies. Therefore, the output ranking must contain as many positives as possible in the very top list in order to optimize the recovery of sums due. Moreover, the model must be interpretable to justify why a company has been selected. The Python code of the algorithms, experiments, plots are publicly available[3].

For each dataset, we split the data into training (70%) and test (30%) sets. We select the hyperparameters by maximizing the AP through a 5-folds cross validation over the training set. We repeat the process over 20 runs and average the results in both terms of AP and $Precision@k$. For the latter, we select k as the percentage of the number of positives in the test set.

[3] `https://github.com/LeoGautheron/submission-PRL`

Table 1: Public datasets, sorted by positive rate. %+: percentage of positives, n: number of examples, d: number of features.

datasets	% +	n	d	datasets	% +	n	d	datasets	% +	n
abalone20	00.62%	4177	10	segmentation	14.29%	2310	19	wdbc	37.26%	569
abalone17	01.39%	4177	10	hayes	22.73%	132	4	autompg	37.50%	392
yeast6	02.36%	1484	8	vehicle	23.52%	846	18	spambase	39.42%	4597
wine4	03.31%	1599	11	german	30.00%	1000	24	bupa	42.03%	345
libras	06.67%	360	90	newthyroid	30.23%	215	5	heart	44.44%	270
satimage	09.73%	6435	36	glass	32.71%	214	9	australian	44.49%	690
pageblocks	10.23%	5473	10	wine	33.15%	178	13	balance	46.08%	625
yeast3	10.98%	1484	8	pima	34.90%	768	8	sonar	46.63%	208
bankmarketing	11.70%	45211	51	iono	35.90%	351	34	splice	48.09%	3175
abalone8	13.60%	4177	10							

5.2 Comparison with Decision Tree methods

We carried out a first series of experiments consisting in comparing **MetaAP** to three DT baselines: the ranking DT algorithm [3] using either (i) the **Gini** ot (ii) the **Entropy** criterion, and (iii) **TreeRank** [11]. The depths of internal trees and the global tree of **TreeRank** and **MetaAP** are tuned in the set $\{2, 3, ..., 9, 10\}$ which can lead when building the full tree to a maximum depth of $10 \times 10 = 100$. To make the experiments fair, the tree depths of **Gini** and **Entropy**-based DTs are tuned in the set $\{2, 3, ..., 8, 9, 10, 20, ..., 80, 90, 100\}$.

The mean APs over the 28 public datasets are reported in Figure 3. It shows five comparisons from 50% to 10%: the former encompasses the datasets with at most an imbalance ratio of 50% (thus, considers all the 28 datasets), while the latter takes into account the 6 most imbalanced datasets with at most 10% of positives. Whatever the percentage between 50% and 10%, we can note that our method (in green) outperforms the other competitors. The superiority of **MetaAP** seems to be even larger as the imbalance ratio increases. In order to evaluate the significance of these results, we performed a Wilcoxon signed-rank test in each scenario by comparing **MetaAP with the first best competitor**. It is worth noting that at 50%, thus comparing with **TreeRank** over the 28 datasets, we obtain a p-value smaller than 0.05. For the other imbalance ratios, we get a p-value between 0.1 (for 10%) and 0.2 (for 20%, 30% and 40%). Because of the limited allowed space, we do not report the results on the fraud detection task, but note that the same conclusions can be made.

The usefulness of our method is emphasized by Figure 6 (top) which reports the results in terms of $Precision@k$ (only for the cases 10%, 30% and 50% for the sake of conciseness). While, as expected, the gap between the methods in terms $Precision@k$ tends to be smaller and smaller as the percentage of positives grows, for small values of k (the ones that are considered in the case of budget limitation), **MetaAP** is most of the time much better that **TreeRank** and all the other competitors (green curve above the others). This behavior is confirmed and even amplified on the fraud detection task. The results in terms of $Precision@k$ reported in Figure 4 show that **MetaAP** significantly outperforms all the other methods for the two cases of studied frauds up to a value of k equivalent to 25% of the number of positives in the test set. This impressive result opens the door to the use of **MetaAP** for making the tax audit process much more efficient.

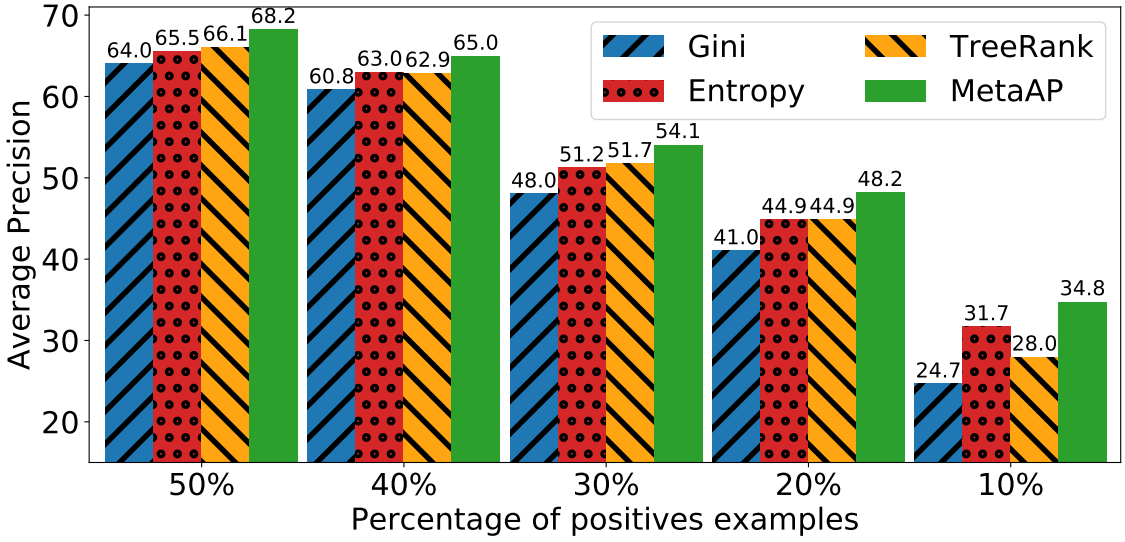

Figure 3: Mean AP as a function of the percentage of positives. For 50%, AP is computed from all the 28 datasets. For 10%, only 6 datasets are used.

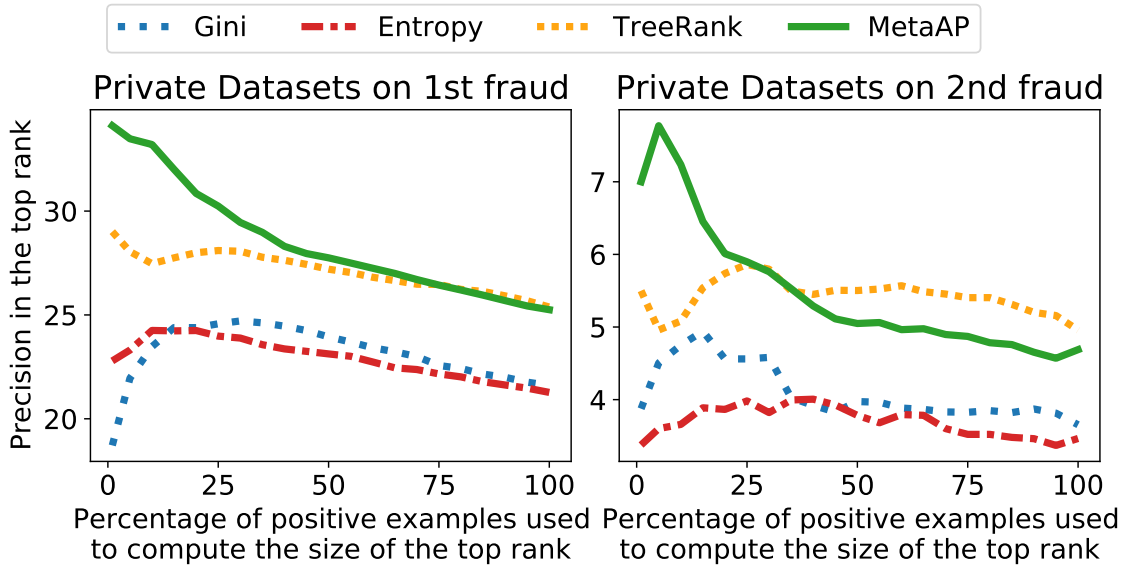

Figure 4: *Precision@k* according to the % of positives on the private datasets.

5.3 Analysis of an early stopping strategy

To analyze the impact of the tree depth on the behavior of the methods, we studied an early stopping in the construction of the models at fixed depths p from 1 to 10 for **MetaAP** and

TreeRank. For the sake of fairness, we used a depth of p^2 for **Gini** and **Entropy**-based DTs which leads to the same expressiveness for all the resulting (meta)-trees. The results on the public datasets are reported in Figure 5. We can note that as soon as **MetaAP** has a depth greater than 3, it outperforms all the other methods, at equivalent depth, whatever the percentage of positives considered.

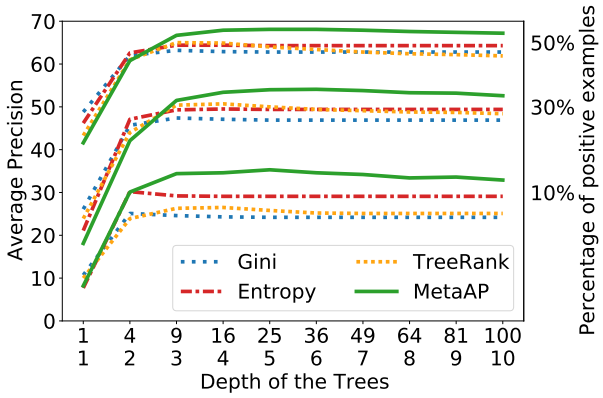

Figure 5: Mean *AP* as a function of the depth on the public datasets with a proportion of positives of at most 10%, 30% and 50%. The 1st line (*resp.* 2nd) of the x-axis represents the depth of the **Gini** and **Entropy**-based DTs (*resp.* **TreeRank** and **MetaAP**).

5.4 Comparison with forest-based methods

In a last series of experiments, even though we are mainly interested in interpretable models, we compared our Random Forest version **MetaAPForest** with 3 baselines: (i) the RF **EntropyForest** using the **Entropy** DT [3], (ii) the ranking version of *XGBoost* [9]: **XGBRanker**, (iii) **SGBAP** [15] that optimizes the *AP*. The number of DTs in the forests and in the gradient tree boosting methods was set to 100 and the depth of the trees is tuned between 2 and 10. For the forests, we considered a bootstrap of the size of the number of training examples for each tree, but we did not use bootstrap for the features and considered all of them at each node. We did not make use of the RF version of *TreeRank* because its implementation in R has been shown to be too much memory consuming during the experiments. We neither considered **GiniForest** because it led to slightly worse performances than its counterpart **EntropyForest**.

As expected, Gradient Boosting outperforms the other approaches in terms of *AP*. Over the 28 public datasets, **XGBRanker** reaches 77.0% while **MetaAP** leads to an *AP* equal to 74.8%, 74.9% for **EntropyForest** and 73.3% for **SGBAP**. However, recall that both Gradient Boosting and Random Forest methods do not fulfill our requirement of inferring interpretable models. Interestingly, Figure 6 (bottom) shows that in terms of *Precision@k*, **MetaAP** is competitive by outperforming both **EntropyForest** and **SGBAP** and being just slightly below **XGBRanker**.

5.5 Compact and interpretable meta-trees

We end this section by illustrating in Figure 7 the fact that **MetaAP** allows to induce much more compact models with the same fixed depth than standard DT methods. This compression comes from the fact that by construction **MetaAP** allows leaves to be reached by several paths. In terms of AP, we can see that such compact meta-trees can allow to avoid overfitting phenomena as illustrated in the Figure by a smaller gap between the APs at training and test time (49.82 versus 43.68) compared to that of the Entropy-based DT (58.62 versus 35.16).

6 Conclusion and Perspectives

In this paper, we address the challenging problem of learning to rank from imbalanced data. We design an algorithm that builds meta-trees by optimizing during the learning process the Average Precision (AP). As far as we know, this paper is the fist attempt to optimize directly this non-convex and non separable function in a tree-based ranking method. The resulting algorithm **MetaAP** shows very promising results on public datasets and on a tax fraud detection task where the need of generating a short list of alerts maximizing the AP is of great interest. **MetaAP** also comes with the valuable property of inducing compact interpretable models. In future work, we plan to investigate the optimization of the $Precision@k$. This would allow us to directly take into account the number of controls k allowed by the application at hand. However, we will need to figure out solutions, for example using transfer learning methods, to avoid retraining from scratch a model learned for a given k. Finally, even if our setting in not as favourable as that of **TreeRank** (unlike the ROC curve, the (1-P)R is not a monotonically increasing function), we will investigate the possibility to derive generalization guarantees.

References

[1] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. Fraud detection system: A survey. Journal of Network and Computer Applications, 68:90–113, 2016.

[2] Shikha Agrawal and Jitendra Agrawal. Survey on anomaly detection using data mining techniques. In KES, pages 708–713. Elsevier, 2015.

[3] Isabelle Alvarez and Stephan Bernard. Ranking cases with decision trees: a geometric method that preserves intelligibility. In IJCAI, 2005.

[4] Kendrick Boyd, Kevin H Eng, and C David Page. Area under the precision-recall curve: point estimates and confidence intervals. In ECML-PKDD, pages 451–466, 2013.

[5] Leo Breiman. Random forests. Machine learning, 45(1):5–32, 2001.

[6] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. Learning, 11(23-581):81, 2010.

[7] Ramiro Camino and Chris Hammerschmidt. Oversampling tabular data with deep generative models: Is it worth the effort? In "I Can't Believe It's Not Better!" NeurIPS 2020 workshop, 2020.

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. Journal of artificial intelligence research, 16, 2002.

[9] Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. Xgboost: extreme gradient boosting. R package version 0.4-2, 1(4), 2015.

[10] Stéphan Clémençon, Marine Depecker, and Nicolas Vayatis. Ranking forests. Journal of Machine Learning Research, 14(Jan):39–73, 2013.

[11] Stéphan Clémençon and Nicolas Vayatis. Tree-based ranking methods. IEEE Transactions on Information Theory, 55(9):4316–4336, 2009.

[12] Georgios Douzas and Fernando Bação. Effective data generation for imbalanced learning using conditional generative adversarial networks. Expert Syst. Appl., 91:464–471, 2018.

[13] Charles Elkan. The foundations of cost-sensitive learning. In IJCAI, volume 17, pages 973–978, 2001.

[14] L. Feng, H. Wang, B. Jin, H. Li, M. Xue, and L. Wang. Learning a distance metric by balancing kl-divergence for imbalanced datasets. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018.

[15] Jordan Frery, Amaury Habrard, Marc Sebban, Olivier Caelen, and Liyun He-Guelton. Efficient top rank optimization with gradient boosting for supervised anomaly detection. In ECML-PKDD, pages 20–35, 2017.

[16] Jerome H Friedman. Stochastic gradient boosting. Computational statistics & data analysis, 38(4):367–378, 2002.

[17] Mikel Galar, Alberto Fernandez, Edurne Barrenechea, Humberto Bustince, and Francisco Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 42(4):463–484, 2011.

[18] Vicente García, J. Salvador Sánchez, A. I. Marqués, Rogelio Florencia Juárez, and Gilberto Rivera Zárate. Understanding the apparent superiority of over-sampling through an analysis of local information for class-imbalanced data. Expert Syst. Appl., 158:113026, 2020.

[19] Léo Gautheron, Amaury Habrard, Emilie Morvant, and Marc Sebban. Metric learning from imbalanced data with generalization guarantees. Pattern Recognit. Lett., 133:298–304, 2020.

[20] Muhammad Ibrahim. Reducing correlation of random forest–based learning-to-rank algorithms using subsample size. Computational Intelligence, 35(4):774–798, 2019.

[21] Muhammad Ibrahim. Sampling non-relevant documents of training sets for learning-to-rank algorithms. International Journal of Machine Learning and Computing, 10:406–415, 05 2020.

[22] Breiman LI, Jerome Friedman, RA Olshen, and C.J. Stone. Classification and Regression Trees (CART), volume 40. 09 1984.

[23] Charles X Ling and Robert J Yan. Decision tree with better ranking. In ICML, pages 480–487, 2003.

[24] Kun Liu, Jiangrui Han, Haiyong Chen, Haowei Yan, and Peng Yang. Defect detection on el images based on deep feature optimized by metric learning for imbalanced data. In ICAC, pages 1–5. IEEE, 2019.

[25] Aditya Krishna Menon and Robert C Williamson. Bipartite ranking: a risk-theoretic perspective. The Journal of Machine Learning Research, 17(1):6766–6867, 2016.

[26] J. Ross Quinlan. Induction of decision trees. Machine learning, 1(1):81–106, 1986.

[27] J Ross Quinlan. C4. 5: programs for machine learning. Elsevier, 2014.

[28] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. Nature Machine Intelligence, 1(5):206–215, 2019.

[29] Ivan Tomek. Two modifications of cnn. In IEEE Transactions on Systems Man and Communications, pages 769–772, 1976.

[30] D. L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. IEEE Transactions on Systems, Man, and Cybernetics, SMC-2(3):408–421, 1972.

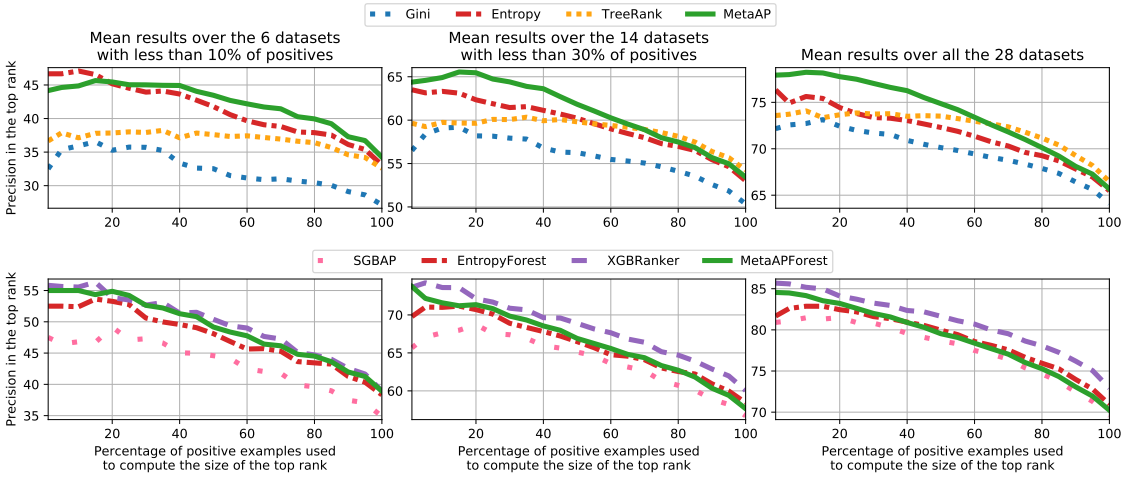

Figure 6: *Precision@k* according to the percentage of positives. (Top) DT methods. (Bottom) RF and Gradient Tree Boosting methods.

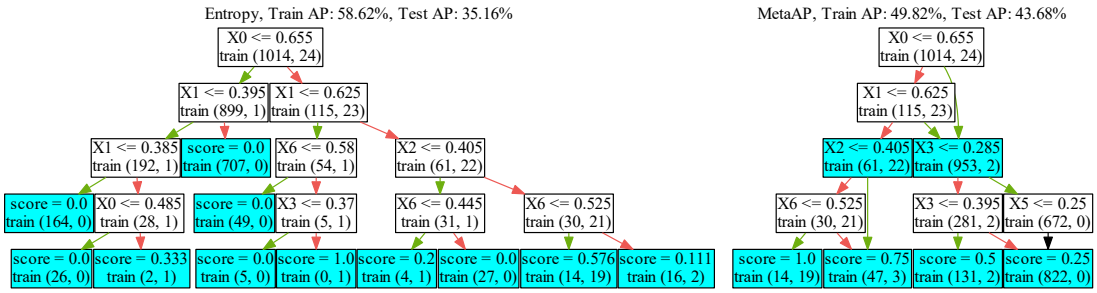

Figure 7: Trees learned on *yeast*6 using the **Entropy** (left) and **MetaAP** (right). A green (*resp.* red) arrow means that the feature considered in the node is lower or equal to (*resp.* larger than) the threshold. A black arrow is used when both the green and red arrows are going from the same parent to the same child. Each node (*resp.* leave) contains the splitting criterion (*resp.* the ranking score) and the number of training negatives and positives (n^-, n^+).