Learning Multi-label Alternating Decision Trees from Texts and Data

Françesco de Comite, Rémi Gilleron, Marc Tommasi

Learning Multi-label Alternating Decision Trees from Texts and Data *

Francesco De Comité[1], Rémi Gilleron[2], and Marc Tommasi[2]
équipe Grappa — EA 3588

[1] Lille 1 University, 59655 Villeneuve d'Ascq Cedex, France
decomite@lifl.fr
[2] Lille 3 University, 59653 Villeneuve d'Ascq Cedex, France
gilleron,tommasi@univ-lille3.fr

Abstract. Multi-label decision procedures are the target of the supervised learning algorithm we propose in this paper. Multi-label decision procedures map examples to a finite set of labels. Our learning algorithm extends Schapire and Singer's Adaboost.MH and produces sets of rules that can be viewed as trees like Alternating Decision Trees (invented by Freund and Mason). Experiments show that we take advantage of both performance and readability using boosting techniques as well as tree representations of large set of rules. Moreover, a key feature of our algorithm is the ability to handle heterogenous input data: discrete and continuous values and text data.

Keywords : boosting - alternating decision trees - text mining - multi-label problems

1 Introduction

When a patient spends more than 3 days in center X, measures of albuminuri as well as proteinuri are made. But if the patient is in center Y, then only measures of albuminuri are made.

These sentences can be viewed as a multi-label classification procedure because more than one label among $\{albuminuri, proteinuri\}$ may be assigned to a given description of a situation. That is to say we are faced a categorization task for which the categories are not mutually exclusive. This work is originally motivated by a practical problem in medicine where each patient may be described by continuous-valued attributes (*e.g.* measures), nominal attributes (*e.g.* sex, smoker,...) and text data (*e.g.* descriptions, comments,...). It was important to produce rules that can be interpreted by physicians and also rules that reveal correlations between labels predictions. These requirements have lead to the realization of the algorithm presented in this paper[1].

* Partially supported by project DATADIAB: "ACI télémédecine et technologies pour la santé" and project TACT/TIC Feder & CPER Région-Nord Pas de Calais.

[1] The algorithm and a graphical user interface are available from `http://www.grappa.univ-lille3.fr/grappa/index.php3?info=logiciels`

Multi-label classification problems are ubiquitous in real world problems. Learning algorithms that can hold multi-label problems are therefore valuable. Of course there are many strategies to apply a combination of many binary classifiers to solve multi-label problems ([ASS00]). But most of them ignore correlations between the different labels. AdaBoost.MH algorithm proposed by Schapire and Singer ([SS00]) efficiently handles multi-label problems. For a given example, it also provides a real value as an outcome for each label. For practical applications, these values are important because they can be interpreted as a confidence rate about the decision for the considered label. AdaBoost.MH implements boosting techniques that are theoretically proved to transform a weak learner — called base classifier — into a strong one. The main idea of boosting is to combine many simple and moderately inaccurate rules built by the base classifier into a single highly accurate rule. The combination is a weighted majority vote over all simple rules. Boosting has been extensively studied and many authors have shown that it performs well on standard machine learning tasks ([Bre98], [FS96] [FS97]). Unfortunately, as pointed by Freund and Mason an others authors, the rule ultimately produced by a boosting algorithm may be difficult to understand and to interpret.

In [FM99], Freund and Mason introduce Alternating Decision Trees (ADTrees). The motivation of Freund and Mason was to obtain intelligible classification models when applying boosting methods. ADTrees are classification models inspired by both decision trees and option trees ([KK97]). ADTrees provide a symbolic representation of a classification procedure and give together with classification a measure of confidence. Freund and Mason also propose an alternating decision tree learning algorithm called ADTBoost in [FM99]. ADTBoost algorithm originates from two ideas: first, decision tree learning algorithms may be analyzed as boosting algorithms (see [DKM96], [KM96]); second, boosting algorithms could be used because ADTrees generalize both voted decision stumps and voted decision trees. Recently, the ADTree formalism has been extended to the multiclass case ([HPK$^+$02]).

We propose in this paper to extend ADTrees formalism to handle multi-label decision procedure. Decision procedures are learned from data described by nominal, continuous and text data. Our algorithm can be understood as an extension of AdaBoost.MH that permits a better readability of the classification rule ultimately produced as well as an extension to ADTBoost in order to handle multi-label classification problems. The multi-label ADTree formalism gives an intelligible set of rules viewable as a tree. Moreover, rules allow to combine atomic tests. In our implementation, we can handle test on (continuous or discrete) tabular data as well as tests on text data. This is particularly valuable in medicine where descriptions of patients combine diagnostic analysis, comments, dosages, measures, and so on. For instance, a rule can be built on both temperature and diagnostic, *if temperature* $>$ *37.5 and diagnostic contains "Cardiovascular" then ....* This kind of combination in a unique rule is not considered by others algorithms like Boostexter which implements AdaBoost.MH. We expect the algorithm to find more concise set of rules thanks to such combinations. We

are convinced that rules with several tests in their premises provide useful informations that can be interpreted by experts. In this paper, we only present results on freely available data sets. We compare our algorithm ADTBoost.MH with AdaBoost.MH on two data sets: the reuters collection and a new data set built from news articles[2].

In Section 2, we define multi-label problems and we recall AdaBoost.MH's functioning. Alternating Decision Trees are presented in Section 3. We define multi-label ADTrees which generalize ADTrees to the multi-label case in Section 4. An example is given in Figure 3. A Multi-label ADTree is an easily readable representation of both different ADTrees (one ADTree per label) and of many decision stumps (one per boosting round). We propose a multi-label ADTree learning algorithm ADTboost.MH based on both ADTboost and Adaboost.MH [SS98]. Experiments are given in Section 5. They show that our algorithm reaches the performance of well tuned algorithms like Boostexter.

2 Boosting and Multi-label Problems

Most of supervised learning algorithms deal with binary or multiclass classification tasks. In such a case, an instance belongs to one class and the goal of learning algorithms is to find an hypothesis which minimizes the probability that an instance is misclassified by the hypothesis. Even when a learning algorithm do not apply to the multiclass case, there exist several methods that can combine binary decision procedures in order to solve multiclass problems ([DB95], [ASS00]). In this paper, we consider the more general problem, called multi-label classification problem, in which an example may belong to any number of classes. Formally, let $\mathcal{X}$ be the universe and let us consider a set of labels $\mathcal{Y} = \{1, \ldots, k\}$. The goal is to find with input a sample $S = \{(x_i, Y_i) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y}, 1 \leq i \leq m\}$ an hypothesis $h : \mathcal{X} \mapsto 2^{\mathcal{Y}}$ with low error. It is unclear to define the error in the multi-label case because different definitions are possible, depending on the application we are faced with. In this paper, we only consider the Hamming error.

Hamming error: The goal is to predict the set of labels associated with an example. Therefore, one takes into account *prediction errors* (an incorrect label is predicted) and *missing errors* (a label is not predicted). Let us consider a target function $c : \mathcal{X} \mapsto 2^{\mathcal{Y}}$ and an hypothesis $h : \mathcal{X} \mapsto 2^{\mathcal{Y}}$, the *Hamming error* of h is defined by:

$$E_H(h) = \frac{1}{k} \sum_{l=1}^{k} D(\{x \in \mathcal{X} \mid (l \in h(x) \wedge l \notin c(x)) \vee (l \notin h(x) \wedge l \in c(x))\}). \quad (1)$$

[2] Data sets and perl scripts are available on `http://www.grappa.univ-lille3.fr/recherche/datasets`.

The factor $\frac{1}{k}$ normalizes the error in the interval $[0,1]$. The training error over a sample S is:

$$E_H(h,S) = \frac{1}{km}\sum_{i,l}(\parallel (l \in h(x_i) \wedge l \notin Y_i \parallel + \parallel l \notin h(x_i) \wedge l \in Y_i \parallel) \qquad (2)$$

where $\parallel a \parallel$ equals 1 if a holds and 0 otherwise.

In the rest of the paper, we will consider learning algorithms that output mappings h from $\mathcal{X} \times \mathcal{Y}$ into $\mathbb{R}$. The real value $h(x,l)$ can be viewed as a prediction value for the label l for the instance x. Given h, we define a multi-label interpretation h^m of h: h^m is a mapping $\mathcal{X} \to 2^{\mathcal{Y}}$ such that $h^m(x) = \{l \in \mathcal{Y} \mid h(x,l) > 0\}$.

AdaBoost.MH

Schapire and Singer introduce AdaBoost.MH in [SS00]. Originally, the algorithm supposes a weak learner from $\mathcal{X} \times \mathcal{Y}$ to $\mathbb{R}$. In this section we focus on boosting decision stumps. We are given a set of conditions $\mathcal{C}$. Weak hypotheses are therefore rules of the form *if* c *then* $(a_l)_{l\in\mathcal{Y}}$ *else* $(b_l)_{l\in\mathcal{Y}}$ where $c \in \mathcal{C}$ and a_l, b_l are real values for each label l. Weak hypotheses therefore make their predictions based on a partitioning of the domain $\mathcal{X}$.

The AdaBoost.MH learning algorithm is given in Algorithm 1. Multi-label data are firstly transformed into binary data. Given a set of labels $Y \subseteq \mathcal{Y}$, let us define $Y[l]$ to be $+1$ if $l \in Y$ and to be -1 if $l \notin Y$. Given an input sample of m examples, the main idea is to replace each training example (x_i, Y_i) by k examples $((x_i,l), Y_i[l])$ for $l \in \mathcal{Y}$. AdaBoost.MH maintains a distribution over $\mathcal{X} \times \mathcal{Y}$. It re-weights the sample at each boosting step. Basically, examples that were misclassified by the hypothesis in the previous round have an higher weight in the current round. It is proved in [SS98] that the normalization factor Z_t induced by the re-weighting realizes a bound on the empirical Hamming loss of the current hypothesis. Therefore, this bound is used to guide the choice of the weak hypothesis at each step. AdaBoost.MH tries to minimize error while minimizing the normalization factor denoted by Z_t at each step.

Let us denote $W_+^l(c)$ (resp. $W_-^l(c)$) the sum of weights of the positive (resp. negative) examples that satisfies condition c and have label l.

$$W_+^l(c) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } c \wedge Y_i[l] = +1 \parallel$$

$$W_-^l(c) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } c \wedge Y_i[l] = -1 \parallel$$

Therefore, one can prove analytically that the the best prediction values a_l and b_l which minimize Z_t at each step are:

$$a_l = \frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)} \; ; \; b_l = \frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)} \qquad (3)$$

leading to a normalization factor Z_t :

$$Z_t(c) = 2\sum_{l=1}^{l=k} \sqrt{W_+^l(c)W_-^l(c)} \tag{4}$$

Algorithm 1 AdaBoost.MH(T) where T is the number of boosting rounds

Input: a sample $S = \{(x_1, Y_1), \ldots, (x_m, Y_m) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y} = \{1, \ldots, k\}\}$; $\mathcal{C}$ is a set of base conditions

1: Transform S into $S^k = \{((x_i, l), Y_i[l]) \mid 1 \leq i \leq m, l \in \mathcal{Y}\} \subseteq (\mathcal{X} \times \mathcal{Y}) \times \{-1, +1\}$
2: Initialize the weights: $1 \leq i \leq m$, $l \in \mathcal{Y}$, $w_1(i, l) = 1$
3: **for** $t = 1..T$ **do**
4: choose c which minimize $Z_t(c)$ according to Equation 4
5: build the rule r_t : if c then $\frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)}$ else $\frac{1}{2} ln \frac{W_+^l(c)}{W_-^l(c)}$
6: Update weights : $w_{t+1}(i, l) = w_t(i, l) e^{-Y_i[l] r_t(x_i, l)}$
7: **end for**

Output: $f(x, l) = \sum_{t=1}^{T} r_t(x, l)$.

3 Alternating Decision Trees

Alternating Decision Trees (ADTrees) are introduced by Freund and Mason in [FM99]. They are similar to option trees developed by Kohavi *et al* [KK97]. An important motivation of Freund and Mason was to obtain intelligible classification models when applying boosting methods. Alternating decision trees contain *splitter nodes* and *prediction nodes*. A splitter node is associated with a test, a prediction node is associated with a real value. An example of ADTree is given in Figure 1. It is composed of four splitter nodes and nine prediction nodes. An instance defines a set of paths in an ADTree. The classification which is associated with an instance is the sign of the sum of the predictions along the paths in the set defined by this instance. Consider the ADTree in Figure 1 and the instance $x = (color = red, year = 1989, \ldots)$, the sum of predictions is $+0.2 + 0.2 + 0.6 + 0.4 + 0.6 = +2$, thus the classification is $+1$ with high confidence. For the instance $x = (color = red, year = 1999, \ldots)$, the sum of predictions is $+0.4$ and the classification is $+1$ with low confidence. For the instance $x = (color = white, year = 1999, \ldots)$, the sum of predictions is -0.7 and the classification is -1 with medium confidence.

ADTree depicted in Fig. 1 can also be viewed as consisting of a root prediction node and four units of three nodes each. Each unit is a decision rule and is composed of a splitter node and two prediction nodes that are its children. It is easy to give another description of ADTrees using sets of rules. For instance, the ADTree in Figure 1 is described by the set of rules:

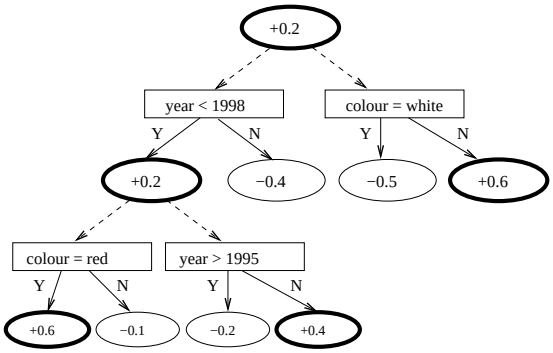

Fig. 1. an example of ADTree; bold prediction nodes define the set of nodes associated with the instance $x = (color = red, year = 1989, \ldots)$

If TRUE	then (if TRUE	then +0.2 else 0	) else 0
If TRUE	then (if year $<$ 1998	then +0.2 else -0.4	) else 0
If year $<$ 1998	then (if colour = red	then +0.6 else -0.1	) else 0
If year $<$ 1998	then (if year $>$ 1995	then -0.2 else +0.4	) else 0
If TRUE	then (if colour = white	then -0.5 else +0.6	) else 0

ADTBoost

Rules in an ADTree are similar to decision stumps. Consequently, one can apply boosting methods in order to design an ADTree learning algorithm. The algorithm proposed by Freund and Mason [FM99] is based on this idea. It relies on Schapire and Singer [SS98] study of boosting methods. A rule in an ADTree defines a partition of the instance space into three blocks defined by $C_1 \wedge c_2$, $C_1 \wedge \neg c_2$ and $\neg C_1$. Following this observation, Freund and Mason apply a variant of Adaboost proposed in [SS98] for domain-partitioning weak hypotheses. Basically, the learning algorithm builds an ADTree with a top down strategy. At every boosting step, it selects and adds a new rule or equivalently a new unit consisting of a splitter node and two prediction nodes. Contrary to the case of decision trees, the reader should note that a new rule can be added below any prediction node in the current ADTree.

Freund and Mason's algorithm ADTboost is given as Algorithm 2. Similarly to Adaboost algorithm, ADTboost updates weights at each step (line 8). The quantity $Z_t(C_1, c_2)$ is a normalization coefficient. It is defined by

$$Z_t(C_1, c_2) = 2\left(\sqrt{W_+(C_1 \wedge c_2)W_-(C_1 \wedge c_2)} + \sqrt{W_+(C_1 \wedge \neg c_2)W_+(C_1 \wedge \neg c_2)}\right) + W(\neg C_1) \quad (5)$$

where $W_+(C)$ (resp. $W_-(C)$) is the sum of the weights of the positive (resp. negative) examples that satisfy condition C. It has been shown that the product of all such coefficients gives an upper bound of the training error. Therefore, ADTboost selects a precondition C_1 and a condition c_2 that minimize $Z_t(C_1, c_2)$ (line 5) in order to minimize the training error. The prediction values a and b (line 6) are chosen according to results in [SS98].

ADTboost is competitive with boosting decision tree learning algorithms such as C5 + Boost. Moreover, the size of the ADTree generated by ADTboost is often smaller than the model generated by other methods. These two points have strongly motivated our choices although ADTboost suffers from some drawbacks *e.g.* the choice of the number of boosting rounds is difficult and overfitting occurs. We discuss these problems in the conclusion.

Algorithm 2 ADTboost(T) where T is the number of boosting rounds

Input: a sample $S = \{(x_1, y_1), \ldots, (x_m, y_m) \mid x_i \in \mathcal{X}, y_i \in \{-1, +1\}\}$; a set of base conditions $\mathcal{C}$.

1: Initialize the weights: $1 \leq i \leq m$, $w_{i,1} = 1$
2: Initialize the ADTree: $\mathcal{R}_1 = \{r_1 : (\text{if } \boldsymbol{T} \text{ then } (\text{if } \boldsymbol{T} \text{ then } \frac{1}{2} \ln \frac{W_+(\boldsymbol{T})}{W_-(\boldsymbol{T})}) \text{ else } 0) \text{ else } 0)\}$
3: Initialize the set of preconditions: $\mathcal{P}_1 = \{\boldsymbol{T}\}$
4: **for** t = 1..T **do**
5: Choose $C_1 \in \mathcal{P}_t$ and $c_2 \in \mathcal{C}$ which minimize $Z_t(C_1, c_2)$ according to Equation 5
6: $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_{t+1}$: (if C_1 then (if c_2 then $\frac{1}{2} \ln \frac{W_+(C_1 \wedge c_2)}{W_-(C_1 \wedge c_2)}$) else $\frac{1}{2} \ln \frac{W_+(C_1 \wedge \neg c_2)}{W_-(C_1 \wedge \neg c_2)}$) else 0)$\}$
7: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{C_1 \wedge c_2, C_1 \wedge \neg c_2\}$
8: update weights: $w_{i,t+1}(i) = w_{i,t}(i) e^{-y_i r_t(x_i)}$
9: **end for**

Output: ADTree $\mathcal{R}_{T+1}$

4 Multi-label Alternating Decision Trees

Multi-label ADTrees

We generalize ADTrees to the case of multi-label problems. A prediction node is now associated with a set of real values, one for each label. An example of such an ADTree is given in Figure 3.

Let $\mathcal{X}$ be the universe, the conjunction is denoted by $\wedge$, the negation is denoted by $\neg$, and let $\boldsymbol{T}$ be the `True` condition. Let $(0)_{l \in \mathcal{Y}}$ be a vector of l zeros.

Definition 1. *Let $\mathcal{C}$ be a set of base conditions where a base condition is a boolean predicate over instances. A precondition is a conjunction of base conditions and negations of base conditions. A rule in an ADTree is defined by a precondition C_1, a condition c_2 and two vectors of real numbers $(a_l)_{l \in \mathcal{Y}}$ and $(b_l)_{l \in \mathcal{Y}}$:*

if C_1 *then (if* c_2 *then* $(a_l)_{l\in\mathcal{Y}}$ *else* $(b_l)_{l\in\mathcal{Y}}$*) else* $(0)_{l\in\mathcal{Y}}$*,*

A multi-label alternating decision tree (multi-label ADTree) is a set $\mathcal{R}$ *of such rules satisfying properties (i) and (ii):*

(i) the set $\mathcal{R}$ *must include an initial rule for which the precondition* C_1 *is* $\boldsymbol{T}$*, the condition* c_2 *is* $\boldsymbol{T}$*, and* b *equals* $(0)_{l\in\mathcal{Y}}$*;*

(ii) whenever the set $\mathcal{R}$ *contains a rule with a precondition* C_1'*,* $\mathcal{R}$ *also contains another rule with precondition* C_1 *and there is a base condition* c_2 *such that either* $C_1' = C_1 \wedge c_2$ *or* $C_1' = C_1 \wedge \neg c_2$*.*

A multi-label ADTree maps each instance to a vector of real number in the following way:

Definition 2. *A rule r: if* C_1 *then (if* c_2 *then* $(a_l)_{l\in\mathcal{Y}}$ *else* $(b_l)_{l\in\mathcal{Y}}$ *) else* $(0)_{l\in\mathcal{Y}}$ *associates a real value* $r(x,l)$ *with any* $(x,l) \in \mathcal{X}\times\mathcal{Y}$*. If* (x,l) *satisfies* $C = C_1 \wedge c_2$ *then* $r(x,l)$ *equals* a_l*; if* (x,l) *satisfies* $C = C_1 \wedge \neg c_2$ *then* $r(x,l)$ *equals* b_l*; otherwise,* $r(x,l)$ *equals* 0*.*

An ADTree $\mathcal{R} = \{r_i\}_{i\in I}$ *associates a prediction value* $R(x,l) = \sum_{i\in I} r_i(x,l)$ *with any* $(x,l) \in \mathcal{X}\times\mathcal{Y}$*. A multi-label classification hypothesis is associated with* H *defined by* $H(x,l) = sign(R(x,l))$ *and the real number* $|R(x,l)|$ *is interpreted as the confidence assigned to* $H(x,l)$*,* i.e. *the confidence assigned to the label* l *for the instance* x*.*

ADTBoost.MH

Our multi-label alternating decision tree learning algorithm is derived from both ADTboost and AdaBoost.MH algorithms. Following [SS98], we now make precise the calculation of the prediction values and the value of the normalization factor $Z_t(C_1, c_2)$. The reader should note that we consider partitions over $\mathcal{X}\times\mathcal{Y}$.

On round t, let us denote the current distribution over $\mathcal{X}\times\mathcal{Y}$ by D_t, and let us consider $W_+^l(C)$ (resp. $W_-^l(C)$) as the sum of the weights of the positive (resp. negative) examples that satisfy condition C and have label l:

$$W_+^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \wedge Y_i[l] = +1 \parallel$$

$$W_-^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \wedge Y_i[l] = -1 \parallel$$

$$W^l(C) = \sum_{i=1}^{i=m} D_t(i,l) \parallel x_i \text{ satisfies } C \parallel$$

It is easy to prove that the normalization factor Z_t and the best prediction values a_l and b_l are:

$$a_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge c_2)}{W_-^l(C_1 \wedge c_2)} \; ; \; b_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge \neg c_2)}{W_-^l(C_1 \wedge \neg c_2)} \tag{6}$$

$$Z_t(C_1, c_2) = 2 \sum_{l=1}^{l=k} \left[\sqrt{W_+^l(C_1 \wedge c_2) W_-^l(C_1 \wedge c_2)} + \sqrt{W_+^l(C_1 \wedge \neg c_2) W_-^l(C_1 \wedge \neg c_2)} \right] + W(\neg C_1) \tag{7}$$

The algorithm ADTboost.MH is given as Algorithm 3. In order to avoid extrem values for confidence values, we use the following formulas:

$$a_l = \frac{1}{2} ln \frac{W_{+1}^l(C_1 \wedge c_2) + \epsilon}{W_{-1}^l(C_1 \wedge c_2) + \epsilon} \; ; \; b_l = \frac{1}{2} ln \frac{W_{+1}^l(C_1 \wedge \neg c_2) + \epsilon}{W_{-1}^l(C_1 \wedge \neg c_2) + \epsilon} \tag{8}$$

with $\epsilon = \frac{1}{2mk}$.

Algorithm 3 ADTboost.MH(T) where T is the number of boosting rounds

Input: a sample $S = \{(x_1, Y_1), \ldots, (x_m, Y_m) \mid x_i \in \mathcal{X}, Y_i \subseteq \mathcal{Y} = \{1, \ldots, k\}\}$; $\mathcal{C}$ is a set of base conditions

1: Transform S into $S^k = \{((x_i, l), Y_i[l]) \mid 1 \leq i \leq m, l \in \mathcal{Y}\} \subseteq (\mathcal{X} \times \mathcal{Y}) \times \{-1, +1\}$
2: Initialize the weights: $1 \leq i \leq m$, $l \in \mathcal{Y}$, $w_1(i, l) = 1$
3: Initialize the multi-label ADTree:
$\mathcal{R}_1 = \{r_1 : (\text{if } \boldsymbol{T} \text{ then } (\text{if } \boldsymbol{T} \text{ then } (a_l = \frac{1}{2} \ln \frac{W_+^l(\boldsymbol{T})}{W_-^l(\boldsymbol{T})})_{l \in \mathcal{Y}}) \text{ else } (b_l = 0)_{l \in \mathcal{Y}}) \text{ else } 0)\}$.
4: Initialize the set of preconditions: $\mathcal{P}_1 = \{\boldsymbol{T}\}$.
5: **for** $t = 1..T$ **do**
6: choose $C_1 \in \mathcal{P}_t$ and $c_2 \in \mathcal{C}$ which minimize $Z_t(C_1, c_2)$ according to Equation 7
7: $\mathcal{R}_{t+1} = \mathcal{R}_t \cup \{r_{t+1} : (\text{if } C_1 \text{ then } (\text{if } c_2 \text{ then } (a_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge c_2)}{W_-^l(C_1 \wedge c_2)})_{l \in \mathcal{Y}}) \text{ else } (b_l = \frac{1}{2} ln \frac{W_+^l(C_1 \wedge \neg c_2)}{W_-^l(C_1 \wedge \neg c_2)})_{l \in \mathcal{Y}}) \text{ else } 0)\}$
8: $\mathcal{P}_{t+1} = \mathcal{P}_t \cup \{C_1 \wedge c_2, C_1 \wedge \neg c_2\}$
9: Update weights : $w_{t+1}(i, l) = w_t(i, l) e^{-Y_i[l] r_t(x_i, l)}$
10: **end for**

Output: multi-label ADTree $\mathcal{R}_{T+1}$

Relations with other formalisms

Multi-label ADTrees trivially extends several common formalisms in machine learning.

Voted decision stumps produced for instance by AdaBoost.MH presented as Algorithm 1 are obviously multi-label ADTrees where each rule has $\boldsymbol{T}$ as a precondition. These "flat" multi-label ADTrees can be the output of ADTBoost.MH

if we impose a very simple control. Line 6 of Algorithm 3, we choose c_2 in $\mathcal{C}$ which minimize $Z_t(\boldsymbol{T}, c_2)$ according to Equation 7. Thus, AdaBoost.MH can be considered as a parameterization of ADTBoost.MH.

(Multi-label) decision trees. A (multi-label) decision tree is an ADTree with the following restrictions: any inner prediction node contains 0; there is at most one splitter node below every prediction node; prediction nodes at a leaf position contain values that can be interpreted as classes (using for instance the sign function in the binary case).

Weighted vote of (Multi-label) decision trees. Voted decision trees $t_1, \ldots, t_k$ associated with weights $w_1, \ldots, w_k$ are also simply transformed into ADTrees. One needs to add prediction nodes containing the weight w_i at every leaf of the tree t_i and graft all trees at the root of an ADTree.

5 Experiments

In this section, we describe the experiments we conduct with ADTBoost.MH. We mainly argue that we can obtain both accurate and readable classifiers over tabular and text data using multi-label alternating decision trees.

Implementation

Our implementation of ADTBoost.MH supports items descriptions that include discrete attributes, continuous attributes and texts. In the case of a discrete attribute A whose domain is $\{v_1, \ldots, v_n\}$, the set of base conditions are binary conditions of the form $A = v_i$, $A \neq v_i$. In the case of a continuous attribute A, we consider binary conditions of the form $A < v_i$. Finally, in the case of text-valued attributes over a vocabulary V, base conditions are of the form *m occurs in A* where m belongs to V.

Missing values are handled in ADTBoost.MH as in Quinlan's C4.5 software ([Qui93]). Let us consider an ADTree R, a position p in R associated with condition c based on an attribute A and an instance for which the value of A is missing. We estimate probabilities for the assertions c to be true or false, based on the observation of the training set. The obtained values are assigned to the missing value of this instance and then propagated below p.

Data sets

The Reuters collection is the most commonly-used collection for text classification. We use a formatted version of Reuters version 2 (also called Reuters-21450) prepared by Y. Yang and colleagues[3]. Documents are labeled to belong to at least one of the 135 possible categories. A "sub-category" relation governs categories.

[3] available at `http://moscow.mt.cs.cmu.edu:8081/reuters_21450/parc/`

Nine of them constitute the top level of this hierarchy. Because most of the articles in the whole Reuters data set belong to exactly one category, in the experiments we select categories and articles for which overlaps between categories are more significant.

News collection We prepare a new data set from newsgroups archives in order to build a multilabel classification problem where cases are described by texts, continuous and nominal values. We obtain from `ftp://ftp.cs.cmu.edu/user/ai/pubs/news/comp.ai/` news articles posted in the comp.ai newsgroup in july 1997. Some articles in this forum have been cross posted in several newsgroups. The classification task consists in finding in which newsgroup a news has been cross posted.

Each news is described by seven attributes. Two of them are textual data: the subject and the text of the news. Four attributes are continuous (natural numbers): the number of lines in the text of the news, the number of references (that is the number of parents in the thread discussion), the number of capitalized words and the number of words in the text of the news. One attribute is discrete: the top level domain of sender's email address. We have droped small words, less than three letters, and non purely alphabetic words (*e.g.* R2D2)[4]. There are 524 articles and we keep only the five most frequent cross posted newsgroups as labels. The five newsgroups are: misc.writing (61 posts), sci.space.shuttle (68 posts), sci.cognitive (70 posts), rec.arts.sf.written (70 posts) and comp.ai.philosophy (73 posts). Only 171 articles were cross posted to at least one of these five newsgroups (60 in one, 51 in two, 60 in three).

Results

We first train our algorithm on the Reuters dataset in order to evaluate it against Boostexter (Available implementation of AdaBoost.MH [5]). Reuters dataset consists in a train set and a test set. But following the protocol explained in [SS00], we merge these two sets and we focused on the nine topics constituting the top hierarchy. We then select the subsets of the k classes with the largest number of articles for $k = 3 \ldots 9$. Results were computed on a 3-fold cross-validation. The number of boosting steps being set to 30. We report in table 1 one-error, coverage and average precision for Boostexter and ADTBoost.MH.

In the news classification problem, ranks of labels are less relevant. We only report the hamming error. Our algorithm ADTBoost.MH builds rules that may have large preconditions. This feature allows to partition the space in a very fine way and the training error can decrease very quickly. This can be observed on the news data set. After 30 boosting steps, the training error is 0 for the model generated by ADTBoost.MH. This is achieved by Boostexter after 230 boosting steps. On the one hand, smaller models can be generated by ADTBoost.MH. On

[4] Data sets and perl scripts are available on `http://www.grappa.univ-lille3.fr/recherche/datasets`.

[5] `http://www.cs.princeton.edu/~schapire/boostexter.html`

	ADTree			Boostexter		
k	Error	Cover	Prec	Error	Cover	Prec
3	6.01%	0.07	0.97	6.48%	0.08	0.97
4	7.03%	0.10	0.96	7.93%	0.11	0.96
5	8.31%	0.12	0.95	8.99%	0.14	0.95
6	12.70%	0.24	0.92	12.34%	0.24	0.92
7	14.72%	0.31	0.91	14.32%	0.30	0.91
8	16.01%	0.34	0.91	15.90%	0.35	0.90
9	16.77%	0.40	0.89	16.60%	0.39	0.89

Table 1. Comparing Boostexter and ADTree on the Reuters data set. The number k is the number of labels in the multi-label classification problem.

the other hand, both ADTBoost.MH and ADTBoost tend to overspecialize and this phenomenon seems to occur more quickly for ADTBoost.MH.

Table 2 reports the hamming error on the cross posted news data set computed with a ten-fold cross validation. Note that the hamming error of the procedure that associate the empty set of label to each case is 0.306.

Boosting steps	ADTBoost.MH	Boostexter
10	0.024	0.022
30	0.023	0.019
50	0.017	0.017
100	0.017	0.014

Table 2. Hamming error of ADTBoost.MH and Boostexter on the cross posted news data set.

Figure 2 shows an example of rules produced by ADTBoost.MH on this data set and its graphical representation is depicted in Fig. 3. Both representations allow to interpret the model generated by ADTBoost.MH. For instance, according to the weights computed in the five first rules, one may say that when an article is cross posted in sci.space.shuttle it is not in sci.cognitive. On the contrary rec.arts.sf.written seems to be correlated with sci.space.shuttle. Below is an example of a rule with conditions that mix tests over textual data and tests over continuous data.

```
If subject  (not contains  Birthday) and  (subject  not contains  Clarke)
   and (subject  not contains  Secrets)
                     Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
  If #lines >= 22.50 then       -0.37   -3.24   0.24    0.17    0.08
  If #lines < 22.50 then        -0.12   -2.88   -3.51   -0.41   -5.07
Else 0
```

```
Rule 0
if TRUE
 Class :                misc_w  sci_sp  sci_co  comp_a  rec_ar
   If TRUE then         -1.01   -0.95   -0.93   -0.91   -0.93
Else 0
-------------------------
Rule 1
if TRUE
                           Class :      misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Birthday then       1.71    1.50    -6.35   -6.35   1.71
   If subject  not contains  Birthday then  -7.35   -2.02   -0.86   -0.84   -1.96
Else 0
-------------------------
Rule 2
If subject  not contains  Birthday
                           Class :      misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Emotional then      -2.93   -5.59   7.03    2.65    -5.62
   If subject  not contains  Emotional then -4.12   0.05    -0.41   -0.37   0.05
Else 0
-------------------------
Rule 3
If subject  not contains  Birthday
                              Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Clarke then         -0.46   6.92    -5.30   -5.33   6.89
   If subject  not contains  Clarke then    -2.31   -6.92   0.01    0.01    -1.10
Else 0
-------------------------
Rule 4
If subject  not contains  Birthday
If subject  not contains  Clarke
                              Class :   misc_w  sci_sp  sci_co  comp_a  rec_ar
   If subject Contains  Secrets then        -0.17   -1.99   2.61    -5.83   -4.92
   If subject  not contains  Secrets then   -1.32   -3.59   -0.33   0.02    0.02
Else 0
```

Fig. 2. Output of ADTBoost.MH on the news data set

Fig. 3. A five rule multilabel ADTree built on the cross posted news data set.

6 Conclusion

We have proposed a learning algorithm ADTBoost.MH that can handle multi-label problems and produce intelligible models. It is based on boosting methods and seems to reach the performance of well tuned algorithms like AdaBoost.MH. Further works concern a closer analysis of the overspecialization phenomenon.

The number of rules in a multi-label ADTree is related to the number of boosting rounds in boosting algorithms like AdaBoost.MH. Readability of multi-label ADTree is clearly altered when the number of rules becomes large. But, this possibly large set of rules depicted as a tree comes with weights and with an ordering that permits "stratified" interpretations. Indeed, due to the algorithmic bias in the algorithm, rules that are firstly generated contribute to reduce the most the training error. Nonetheless, navigation tools and high quality user interfaces should be built to improve readability.

References

[ASS00] Erin L. Allwein, Robert E. Schapire, and Yoram Singer. Reducing multiclass to binary:a unifying approach for margin classifiers. In *Proc. 17th International Conf. on Machine Learning*, pages 9–16, 2000.

[Bre98] L. Breiman. Combining predictors. Technical report, Statistic Department, 1998.

[DB95] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[DKM96] T. Dietterich, M. Kearns, and Y. Mansour. Applying the weak learning framework to understand and improve C4.5. In *Proc. 13th International Conference on Machine Learning*, pages 96–104. Morgan Kaufmann, 1996.

[FM99] Yoav Freund and Llew Mason. The alternating decision tree learning algorithm. In *Proc. 16th International Conf. on Machine Learning*, pages 124–133, 1999.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, August 1997.

[HPK+02] G. Holmes, B. Pfahringer, R. Kirkby, E. Frank, and M. Hall. Multiclass alternating decision trees. In *Proceedings of the European Conference on Machine Learning*. Springer Verlag, 2002.

[KK97] Ron Kohavi and Clayton Kunz. Option decision trees with majority votes. In *Proc. 14th International Conference on Machine Learning*, pages 161–169. Morgan Kaufmann, 1997.

[KM96] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 459–468, 1996.

[Qui93] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[SS98] Robert F. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-98)*, pages 80–91, New York, July 24–26 1998. ACM Press.

[SS00] Robert E. Schapire and Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168, 2000.