Annals of University of Craiova, Math. Comp. Sci. Ser.
Volume 36(1), 2009, Pages 71–78
ISSN: 1223-6934

Building Decision Trees

Mircea Preda and Ana-Maria Mirea

Abstract. Decision tree learning represents a well known family of inductive learning algorithms that are able to extract, from the presented training sets, classification rules whose preconditions can be represented as disjunctions of conjunctions of constraints. The name of decision trees is due to the fact that the preconditions can be represented as a tree where each node is a constraint and each path from the root to a leaf node represents a disjunction composed from a conjunction of constraints, one constraint for each node from the path. Due to their efficiency, these methods are widely used in a diversity of domains like financial, engineering and medical. The paper proposes a new method to construct decision trees based on reinforcement learning. The new construction method becomes increasingly efficient as it constructs more and more decision trees because it can learn what constraint should be tested first in order to accurately and efficiently classify a subset of examples from the training set.

2000 Mathematics Subject Classification. Primary 68T05; Secondary 91C20.
Key words and phrases. Decision tree, Reinforcement learning, Inductive learning, Classification, Splitting criteria.

1. Introduction

Decision tree learning is a widely used class of methods for inductive inference. They approximate discrete valued target functions by representing them as decision trees. The instances presented as arguments for these functions are represented as collections of attribute value pairs. In accordance with [4], a decision tree represents a disjunction of conjunctions of constraints on the attribute values of the instances. Each node from the tree represents a test regarding the value of an attribute (or more attributes), each path starting from the tree's root to a leaf corresponds to a conjunction of attribute tests and the tree synthesizes the disjunction of these conjunctions. Consequently, a decision tree can be considered to be a collection of formulas from propositional logic and decision trees naturally represent disjunctive expressions.

Several efficient decision tree building algorithms like ID3 [5], C4.5 [6] and QUEST[3] were proposed over time, most of them being top down, simple to complex, greedy algorithms. They begin with the question "What constraint should be tested in the root of the decision tree?" and evaluate the available constraints in accordance with a statistical test to select the best one. A descendent of the root node is created for each value of this constraint and the entire process is then repeated for each descendant. Due to their accuracy, effectiveness and flexibility, decision trees were successfully applied to a wide range of learning problems from various domains like medical, financial and engineering. A comprehensive survey of the current developments in decision tree learning theory is [8].

Received: 05 June 2009.

The paper proposes a new method to build decision trees that uses reinforcement learning to select the attributes to be tested inside the nodes of the tree. Reinforcement learning ([9], [4]) was selected because it allows that the method to build better trees over time as it learns from the feedback provided by the previous building operations. Several statistical tests can be employed to describe the classification power of the attributes and their importance will be weighted in accordance with the accuracy of their results during learning. It can learn over time which attribute is more suitable to be tested for classifying a subset of examples and it can transfer the learned knowledge about an attribute to another attribute with similar discrimination characteristics. The knowledge can be transferred also when the training set is changed frequently and even when the classification rules can support changes over time. The situations when different attributes have different test costs can be handled by the proposed method. We mention that decision trees were commonly used in relational reinforcement learning [7] but the reverse situation where reinforcement learning is used to build decision trees received little attention until now.

The rest of the paper begins by describing the mathematical framework for a generic classification problem and the fundamentals of the reinforcement learning algorithms. The main part of the paper introduces the new method to adaptively build decision trees by reinforcement learning and discusses its properties. We conclude with experimental results for the proposed method on a small problem and with a synthesis of the desired features of the exposed algorithm.

2. Adaptive learning of a classifier for a generic classification problem

2.1. Framework. Let $\mathcal{E}$ be the set of elements that should be classified. The elements of $\mathcal{E}$ are described by a set of attributes $Attr$. Each attribute $at \in Attr$ has attached a function

$$at(.) : \mathcal{E} \to Val(at)$$

where $Val(at)$ is the set of the all possible values of the attribute at. An element $e \in \mathcal{E}$ is uniquely identified by the set of pairs

$$\{< at, at(e) > | at \in Attr\}.$$

The function $at(.)$ can be extended to subsets of examples as follows:

$$at(.) : 2^{\mathcal{E}} \to Val(at)$$

where

$$at(E) = \arg \max_{v \in Val(at)} |\{e \in E | at(e) = v\}|$$

is the dominant value of the attribute at over the set E.

Let $\mathcal{C}$ be the finite set of the categories of the elements from $\mathcal{E}$. A classifier for $\mathcal{E}$ is a function

$$c : \mathcal{E} \to \mathcal{C}.$$

Usually, the function c is not known. Instead, we have a subset of examples $Ex \subseteq \mathcal{E}$ for that the values $c(ex), \forall ex \in Ex$ are available. Our purpose is to build an approximation

$$c^* : \mathcal{E} \to \mathcal{C}.$$

for c based on the examples from Ex.

For each attribute $at \in Attr$, an equivalence relation $=_{at} \subseteq Ex \times Ex$ will be defined as follows: $(ex_1, ex_2) \in =_{at}$ if and only if $at(ex_1) = at(ex_2)$. Let $X \subseteq Ex$ a set of examples and $ex \in X$ an example. By $[ex]_{X,at}$ will be denoted the equivalence class

$$[ex]_{X,at} = \{x \in X | at(ex) = at(x)\}.$$

The set of all equivalence classes in X given the equivalence relation $=_{at}$ is denoted as $X/ =_{at}$ and called the quotient set of X by $=_{at}$. Sometimes $X/ =_{at}$ will be denoted by

$$X/_{=_{at}} = \{X_{at=v_1}, ..., X_{at=v_m}\}$$

where $Val(at) = \{v_1, ..., v_m\}$.

Similarly, equivalence classes can be defined for the target function c. Let us denote by $[ex]_X$ the equivalence class of ex

$$[ex]_X = \{x \in X | c(ex) = c(x)\}.$$

The set of the all equivalence classes in X given c will be denoted as X/c.

$$X/c = \{X_{c=c_1}, ..., X_{c=c_p}\}$$

where $\mathcal{C} = \{c_1, ..., c_p\}$.

2.2. Reinforcement learning. Let us consider a problem where an agent interacts with an environment. The problem is described by the following parameters:

- S The set of the all possible states of the environment. In the most cases the agent has only partial information about the current state of the environment. Usually, a subset of states $F \subseteq S$ is also known. The elements of F are named final states.
- A The actions that are available to the agent. For a state $s \in S$ of the environment we denote by $A(s) \subseteq A$ the set of the actions that can be performed in the state s.
- $T : S \times A \to \mathcal{P}(S)$ where $\mathcal{P}(S)$ represents the family of the probability distributions over S. The agent has not a complete control over environment. When it performs an action a in a state s it does not completly know which will be the next state of the environment. $T(s, a, s')$ represents the probability that $s' \in S$ to be the next state when the action a is performed in the state s.
- $r : S \times A \to \mathbb{R}$. The one step reward function. The learning agent is partially supervised. It is informed by rewards about what action is good in what state. $r(s, a)$ represents the reward that is received by the agent after it performs the action a in the state s.

A policy π is a function that maps states into probability distributions over the set of actions. A trajectory is a sequence of states, actions and rewards constructed based on a policy as follows. Let s_0 be the initial state of the trajectory. An action a_0 is chosen in accordance with $\pi(s_0)$. The one step reward $r_0 = r(s_0, a_0)$ granted for the pair (s_0, a_0) is observed. The new state of the environment s_1 is dependent on the environment dynamics described by the function T. If s_1 is a final state, the trajectory is completed, otherwise the process continues iteratively.

The action value function or, alternately, the Q function, measures, for the value $Q^\pi(s_t, a)$, the expected total reward that will be obtained by executing the action a in the state s_t and following the policy $\pi(.)$ to select the actions for the next states. The function Q that corresponds to a policy $\pi(.)$ is defined by:

$$Q^\pi(s_t, a_t) \equiv r_t + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1})). \tag{1}$$

$\gamma \in (0,1]$ is a parameter that quantify how important are the rewards received during the later interactions between agent and environment. The advantage provided by the Q function is that the agent can perform an one step search of the optimal action without to be needed to know the one step reward function and the dynamics of the environment. If the agent knows the optimal Q function, denoted by Q^*, which corresponds to the optimal policy π^*, then it can select the optimal action by using the search:

$$a_t^* = \pi^*(s_t) = \arg \max_{a_t \in A(s_t)} \{Q^*(s_t, a_t)\}. \tag{2}$$

The optimal Q function can be computed by using the dynamic programming if the one step reward function and the dynamics of the environment are completely known in advance. But even in this case the spaces of the states and actions can be to big to accurately compute the value $Q^*(s,a)$ for each state action pair (s,a). So, in most cases, a method to approximate the Q^* function is preferred. Reinforcement learning theory proposes several such methods like SARSA and Q-learning [9].

2.3. Building decision trees using reinforcement learning. In order to apply the reinforcement learning to the generalized classification problem we should identify the components of the reinforcement learning framework in the classification settings.

- The set of states S is the family of the all partitions of the set of examples. A state $s \in S$ is a set $s = \{s_1, ..., s_n\}$ such that $s_i \cap s_j = \emptyset$, $\forall i, j \in \{1, ..., n\}, i \neq j$ and $\bigcup_{i=1}^{n} s_i = Ex$. A state $s = \{s_1, ..., s_n\}$ is named final if and only if $s_i/c = \{s_i\}$, $\forall i = \overline{1,n}$ (all examples from s_i are classified in the same category). The family of the all final states will be denoted by $F(S)$.
- Let $s = \{s_1, ..., s_n\} \in S$ be a state. The set of actions $A(s)$ that can be performed in the state s is the set $A(s) = Attr \times \{1, ..., n\}$. An action is a pair (at, i) with the meaning that the attribute at will be used to further classify the subset of examples s_i.
- Let $s = \{s_1, ..., s_n\} \in S$ a state and $a = (at, i) \in A(s)$ an action. The next state s' of the environment after performing action a in the state s will be $s' = \{s_1, ..., s_{i-1}, s_{i_{v_1}}, ..., s_{i_{v_m}}, s_{i+1}, ..., s_n\}$ where $\{s_{i_{v_1}}, ..., s_{i_{v_m}}\} = s_i/_{=at}$. This can be also stated as:
$$s' = (s \setminus \{s_i\}) \cup s_i/_{=at}.$$
- For the reward function we will set $r(s,a) = -1$, $\forall s \notin F(S)$ and $\forall a \in A(s)$ and $r(s,a) = 0$ otherwise. This definition of the reward function is intended to encourage the completion of the decision tree building process in a minimum number of steps (with a minimum number of internal nodes). If the attributes have different test costs we can represent this feature naturally by using different rewards for different attributes.

The number of the all possible partitions of the set of examples Ex is too big, and consequently, the values of the function $Q(s,a)$ that approximates $Q^*(s,a)$ cannot be maintained by using a table. Q will be represented as a parameterized functional form with the parameter vector $\theta = (\theta_1, ..., \theta_k) \in \mathbb{R}^k$. To make the computations simpler, Q will be a linear function of the parameter vector θ. For every pair (s,a), there is a vector of features $\phi(s,a) = (\phi_1^{(s,a)}, ..., \phi_k^{(s,a)})^T \in \mathbb{R}^k$ with the same number of components as θ. The approximate action value function is given by

$$Q(s,a) = \theta \cdot \phi(s,a) = \sum_{i=1}^{k} \theta_i \phi_i^{(s,a)}$$

Algorithm 1 $ADAPTTREE0(Ex, Attr)$ - an adaptive algorithm for constructing decision trees. The algorithm uses linear gradient descendent SARSA with tile coding features and ϵ-greedy policy.

```
Require: $Ex \neq \emptyset$, $\alpha \in (0,1]$ a parameter named learning rate, $\gamma \in (0,1)$ a parameter
    that establishes how important are the rewards received in the future.
1: Initialize $s = \{Ex\}$.
2: Create the root node of the tree attached to unique set $Ex$ from partition.
3: $a, \phi(s,a), Q(s,a) \leftarrow \epsilon - Greedy(s)$. {Choose an action $a \in A(s)$ in accordance
   with the $\epsilon$-greedy policy based on $Q$.}
4: repeat
5:    Let $s = \{s_1, ..., s_n\}$, $a = (at, i) \in A(s)$ and $n_{s_i}$ the node in the tree attached to
      the set $s_i$ from the partition $s$.
6:    for each $s_{i_{v_j}} \in s_i/_{=at}$ do
7:       Create a new node $n_{s_{i_{v_j}}}$ attached to $s_{i_{v_j}}$ and add a new branch in the tree
         from $n_{s_i}$ to $n_{s_{i_{v_j}}}$ labeled with the test $at(.) = v_j$.
8:    end for
9:    Perform the action $a$, observe the next state $s'$ and the reward $r$
10:   $\delta \leftarrow r - Q(s,a)$
11:   $a', \phi(s',a'), Q(s',a') \leftarrow \epsilon - Greedy(s')$. {Choose an action $a' \in A(s')$ in accor-
      dance with the $\epsilon$-greedy policy based on $Q$.}
12:   $\delta \leftarrow \delta + \gamma Q(s',a')$
13:   $\theta \leftarrow \theta + \alpha\delta\phi(s,a)$
14:   $s \leftarrow s'$, $a \leftarrow a'$.
15: until $s \in F(S)$
```

and the resulting method is synthesized in the algorithms 1 and 2. The features will be constructed as follows:

$$\phi : S \times A \rightarrow \mathbb{R}^k,\ \phi(s,a) = f(g(s,a)).$$

The function $g : S \times A \rightarrow \mathbb{R}^{l+1}$ transforms a state action pair (s,a) into an array of real number because the majority of the function approximation methods are devised to work with numeric arguments. The values associated by the g function must synthesize both the status of the learning process in the current state s and the discrimination capabilities of the selected action a. The status of the learning process is described by computing the entropy function over the subsets of the partition s. As the learning process advances, the entropy of the subsets of s should become smaller because, finally, each subset will contain examples from only one category. In order to accurately describe the effects of an action a, the definition of g is also based on some of the most known measures for assessing the classification qualities of an attribute [8].

$$g(s,a) = (g_0(s), g_1(s,a), ..., g_l(s,a)) \text{ where}$$
$$g_0 : S \rightarrow \mathbb{R} \text{ and } g_i : S \times A \rightarrow \mathbb{R}, \forall i, 1 \leq i \leq l.$$

Let $s \in S$ be $s = \{s_1, ..., s_n\}$ and $a \in A(s)$ be $a = (at, i)$. The function g_0 describe the status of the learning process in the current state s (the progress of the classification of the training set) and is defined as follows:

$$g_0(s) = \sum_{i=1}^{n} \frac{|s_i|}{|Ex|} \cdot Entropy(s_i)$$

Algorithm 2 $\epsilon - Greedy(s)$ selects an action $a \in A(s)$ for the state s using the $\epsilon - greedy$ strategy.

```
Require: The exploration probability $\epsilon \in [0,1]$.
 1: if With probability $1-\epsilon$ then
 2:   for all $a \in A(s)$ do
 3:     $\phi(s,a) \leftarrow$ the vector of features for the pair $(s,a)$
 4:     $Q(s,a) \leftarrow \sum_{i=1}^{k} \theta_i \phi_i^{(s,a)}$
 5:   end for
 6:   $a \leftarrow \arg\max_a Q(s,a)$
 7: else
 8:   $a \leftarrow$ a random action $\in A(s)$
 9:   $\phi(s,a) \leftarrow$ the vector of features for the pair $(s,a)$
10:   $Q(s,a) \leftarrow \sum_{i=1}^{k} \theta_i \phi_i^{(s,a)}$
11: end if
12: return $a$, $\phi(s,a)$, $Q(s,a)$
```

where the entropy function is defined by:

$$Entropy(X) = -\sum_{i=1}^{p} \frac{|X_{c=c_i}|}{|X|} \log_2 \frac{|X_{c=c_i}|}{|X|},$$

$\forall X \subseteq Ex$.

The functions $g_i, 1 \le i \le l$ describe the classification effects of the actions. During tests, the following functions g_i were used:

a) Information gain function

$$g_1(\{s_1,...,s_i,...,s_n\},(at,i)) = Entropy(s_i) - \sum_{s_{i_v} \in s_i/=_{at}} \frac{|s_{i_v}|}{|s_i|} Entropy(s_{i_v}).$$

b) Gini index

$$g_2(\{s_1,...,s_i,...,s_n\},(at,i)) = Gini(s_i) - \sum_{s_{i_v} \in s_i/=_{at}} \frac{|s_{i_v}|}{|s_i|} Gini(s_{i_v})$$

where

$$Gini(s) = 1 - \sum_{i=1}^{p} \left(\frac{|s_{c=c_i}|}{|s|}\right)^2$$

c) Discriminant power function

$$g_3(\{s_1,...,s_i,...,s_n\},(at,i)) = \frac{\sum_{s_{i_v} \in s_i/=_{at}} \left(\frac{1}{|s_{i_v}/c|} \cdot |s_{i_v}|\right)}{|s_i|}$$

The function g_3 assigns to each pair (s,a) a number in the range $(0,1]$.

Remark 2.1. *It is difficult to choose between the various measure functions that can be used to select the next attribute used in the tree construction process. Several studies (*[1], [2]*) suggest that the most functions that evaluate the power of discrimination of an attribute regarding to a set of examples have similar performances. Each criterion is superior in some cases and inferior in others. The proposed adaptive tree induction method has the advantage that allows us to use several splitting criteria. During the adaptive process each criterion will gain or lose importance according with its performances.*

Several function approximation methods including artificial neural networks and linear methods, which are well suited for reinforcement learning, can be used to define the function f. In our tests, $f : \mathbb{R}^{l+1} \to \mathbb{R}^k$ was defined by using the tile coding method ([9]) with k the number of used layers of tiles. Finally, we should point that the usage of an approximation for Q^* has another advantage: knowledge can be transferred between similar (s, a) pairs. Consequently, the newly encountered (s, a) pairs can be evaluated based on the old ones.

3. Conclusion

The paper proposes a new method based on reinforcement learning to adaptively build decision trees. The adaptive capabilities of the reinforcement learning provide us the following desired features. Several statistical tests can be used to asses the classification capabilities of the attributes. The method will learn which of them are the most suitable for the current problem. Also, any other methods to represent partitions over the set of examples and constraints over examples can be used if they allow us to have sufficient capabilities to discriminate between the various (partition, constraint) pairs. The method can make use of previously learned knowledge even when the training set and the target function are changing over time. Due to its permanent exploration capacities, reinforcement learning is able to detect the changes in the characteristics of the training set and target function.

References

[1] S. L. Lim, S. L. Loh and S. L. Shih, A comparison of prediction accuracy, complexity and training time of thirty-three old and new classification algorithms, *Machine Learning*, 40, 2000, pp. 203–228.

[2] S. L. Loh and S. L. Shih, Families of splitting criteria for classification trees, *Statist. Comput.*, 9, 1999, pp. 309–315.

[3] T. Loh and T. Shih, Split selection methods for classification trees, *Statistica Sinica*, 7, 1997, pp. 815–840.

[4] Tom M. Mitchell, *Machine Learning*, McGraw Hill, 1997

[5] J.R. Quinlan, Induction of decision trees, *Machine Learning*, 1(1), 1986, pp. 81–106.

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

[7] Larry D. Pyeatt, Reinforcement Learning with Decision Trees, *Applied Informatics, AI 2003*, Innsbruck, Austria, Acta Press, 2003.

[8] Lior Rokach and Oded Maimon, Top-Down Induction of Decision Trees Classifiers - A Survey, IEEE Transactions on Systems, Man and Cybernetics, 35(4), 2005, pp. 476-487.

[9] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, A Bradford Book, The MIT Press, Cambridge, Massachusetts London, England, 1998

(Mircea Preda) University of Craiova Faculty of Mathematics and Computer Science, Department of Computer Science 13 Alexandru Ioan Cuza Street, Craiova, 200585, Romania
E-mail address: mpreda@acm.org

(Ana-Maria Mirea) University of Craiova Faculty of Mathematics and Computer Science, Department of Computer Science 13 Alexandru Ioan Cuza Street, Craiova, 200585, Romania
E-mail address: ammirea@acm.org