A Collaborative Classifier Construction Framework

Mircea PREDA

Faculty of Mathematics and Computer Science,
Department of Computer Science,
University of Craiova, Romania
mirceapreda@central.ucv.ro

Abstract. $ADAPTTREE0$ algorithm, introduced in [6], is a decision tree construction algorithm whose performances increase over time due to the usage of a reinforcement learning method. Decision trees are approximations of discrete functions created based on some sets of examples for that the functions values are known and the classification results provided by a decision tree are inherently imprecise. The situations when several intelligent agents are interested by same classification problem are pretty common and the collaboration between these agents brings at least two advantages: an agent can compare the results of its classification process with the results of the rest of the agents and can adjust its classification process and even it can renounce to perform classification if the task was already completed by other trusted agents. The paper proposes a collaborative framework that includes several agents that use $ADAPTTREE0$ decision tree construction algorithm and cooperate to solve same classification problem. The framework employs a client server architecture, the server mediating the differences between the client agents.
Keywords: decision tree, reinforcement learning, inductive learning, classification, distributed systems
Math. Subject Classification 2000: 68T05, 68T30

1 Introduction

$ADAPTTREE0$ algorithm, introduced in [6], is a decision tree construction algorithm whose performances increase over time due to the usage of a reinforcement learning method that evaluates the decisions taken during of a tree construction process and favors the decisions that conducted to good results in the past. Decision trees are approximations of discrete functions created based on some sets of examples for that the functions values are known. Consequently, the classification results provided by a decision tree are inherently imprecise and this imprecision has two sources: first, the set of examples used to construct the tree can contain noise or can be too small to contain all relevant situations and, second, the tree itself can be pruned in order to obtain generalization capabilities. The situations when several intelligent agents

are interested by same classification problem are pretty common, for example several e-commerce sites can be interested to classify their clients profiles in order to provided customized content to them, the members of an organization receive approximately similar e-mails and can be interested to classify them in regular e-mails or spam, the nodes from a computer network experience similar traffic and can be interested to classify it as regular or attack, etc. If the agents interested by the same classification problem collaborate, this brings two advantages for an individual agent: it can compare the results of its classification process with the results of the rest of the agents and can adjust its classification process and even it can renounce to perform classification if the task was already completed by other trusted agents. The paper proposes a collaborative framework that includes several agents that use $ADAPTTREE0$ decision tree construction algorithm and cooperate to solve same classification problem. The framework employs a client server architecture, the server mediating the differences between the client agents. The rest of the paper is organized in two main sections: the first one presents the theoretical grounds of two reinforcement learning based decision tree building algorithms; the second part sketches the generic collaborative classifier construction framework that uses the above algorithms and explains the functionality of the framework's components.

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

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

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:

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

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

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

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 completely 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.

– $r : S \times A \rightarrow \mathbb{R}$. 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 [10].

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

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

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

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.

Initialize $s = \{Ex\}$.
Create the *root* node of the tree attached to unique set Ex from partition.
$a, \phi(s, a), Q(s, a) \leftarrow \epsilon - Greedy(s)$. {Choose an action $a \in A(s)$ in accordance with the ϵ-greedy policy based on Q.}
repeat
  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.
  for each $s_{i_{v_j}} \in s_i/_{=at}$ **do**
    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 $at(.) = v_j$.
  end for
  Perform the action a, observe the next state s' and the reward r
  $\delta \leftarrow r - Q(s, a)$
  $a', \phi(s', a'), Q(s', a') \leftarrow \epsilon - Greedy(s')$. {Choose an action $a' \in A(s')$ in accordance with the ϵ-greedy policy based on Q.}
  $\delta \leftarrow \delta + \gamma Q(s', a')$
  $\theta \leftarrow \theta + \alpha\delta\phi(s, a)$
  $s \leftarrow s'$, $a \leftarrow a'$.
until $s \in F(S)$

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 [9].

$$g(s, a) = (g_0(s), g_1(s, a), ..., g_l(s, a)) \text{ where}$$

$$g_0 : S \rightarrow \mathbb{R} \text{ and } g_i : S \times A \rightarrow \mathbb{R}, \forall i, 1 \leq i \leq l.$$

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:

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

where the entropy function is defined by:

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

$\forall X \subseteq Ex$.

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 $\epsilon \in [0,1]$.
if With probability $1-\epsilon$ **then**
  for all $a \in A(s)$ **do**
    $\phi(s,a) \leftarrow$ the vector of features for the pair (s,a)
    $Q(s,a) \leftarrow \sum_{i=1}^{k} \theta_i \phi_i^{(s,a)}$
  end for
  $a \leftarrow \arg\max_a Q(s,a)$
else
  $a \leftarrow$ a random action $\in A(s)$
  $\phi(s,a) \leftarrow$ the vector of features for the pair (s,a)
  $Q(s,a) \leftarrow \sum_{i=1}^{k} \theta_i \phi_i^{(s,a)}$
end if
return a, $\phi(s,a)$, $Q(s,a)$

The functions $g_i, 1 \leq i \leq l$ 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 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, $f : \mathbb{R}^{l+1} \rightarrow \mathbb{R}^k$ was defined by using the tile coding method ([10]) 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.

A variant of the $ADAPTTREE0$ algorithm is constituted by the algorithm 3, which is named $ADAPTTREE1$. The main difference between these two algorithms is given by the states representation method. The states in the $ADAPTTREE0$ algorithm are represented by the frontiers of the partial trees, in $ADAPTTREE1$ the main reinforcement learning problem is decomposed in several smaller reinforcement learning problems, one for each descendant of the node that is currently expanded and the states are represented by subsets of the initial set of training examples. Both methods can be used interchangeably in the collaborative classifier construction framework, which will be exposed in the next section.

3 The generic classification system

3.1 Architecture

The global architecture of a generic classification system is depicted in the figure 1. The system is designed to work in collaborative environments being divided in a client part that will work on several workstations that perform similar classification tasks and a server part that runs on server and shares the knowledge between workstations.

The attributes used by the generic classification system are divided in two categories: raw attributes whose values are directly available from the example and processed attributes. For the last ones, their values are obtained by using more complicate procedures.

The interaction between the client application interested to resolve the classification problem and the generic adaptive classification (GAC) system will proceed as follows:

- The client application sends to the client side of the GAC system a set of (attribute, value) pairs (*AV pairs*) that describe a thing or a situation.

Algorithm 3 $ADAPTTREE1(Ex, Attr, D)$ - an algorithm that adaptively builds a decision tree for a set of examples Ex and a set of attributes used for classification $Attr$. The values $c(ex)$ of the target function c are known for each $ex \in Ex$. The maximum depth of the constructed tree cannot exceed the value D. Other parameters used by the algorithm are: α the learning rate, γ shows how important are the future rewards, m a constant named the equivalent sample size

Require: $Ex \neq \emptyset$

Create a root node for the tree

if $[c(ex) = c(ex')\ \ \forall ex, ex' \in Ex]$ or $[Attr = \emptyset]$ or $[D = 0]$ **then**

  $Q(Ex, stop) = (1 - \alpha)Q(Ex, stop) + \alpha r$ where r is the reward received by performing the action *stop* in the state Ex.

  return the single node tree root labeled with $c(ex^*)$ where

$$ex^* = \arg\max_{ex \in Ex} |[ex]|$$

else

  Choose $a \in Attr$ using the ϵ-greedy policy generated from Q where the state is constituted from the examples in Ex.

  Observe the reward r

  $Q(Ex, a) = (1 - \alpha)Q(Ex, a) + \alpha r$

  for each $v_i \in Val(a)$ **do**

    Add a new tree branch below root corresponding to the test $a = v_i$.

    Let $Ex_{v_i} \subseteq Ex$ the subset of examples for that $a(ex) = v_i\ \forall ex \in Ex_{v_i}$

    if $Ex_{v_i} \neq \emptyset$ **then**

      $Q(Ex, a) += \alpha\gamma \frac{|Ex_{v_i}| + m\frac{1}{|Val(a)|}}{|Ex| + m} \max_{a' \in Attr - \{a\}} Q(Ex_{v_i}, a')$

      Below this new branch add the subtree

$$ADAPTTREE1(Ex_{v_i}, Attr - \{a\}, D - 1)$$

    else

      $Q(Ex, a) += \alpha\gamma \frac{1}{|Val(a)|} r'$ where r' is the reward received by performing the action *stop* in the state $\emptyset$.

      Below this new branch add a single tree node labeled with $c(ex^*)$ where

$$ex^* = \arg\max_{ex \in Ex} |[ex]|$$

    end if

  end for

end if

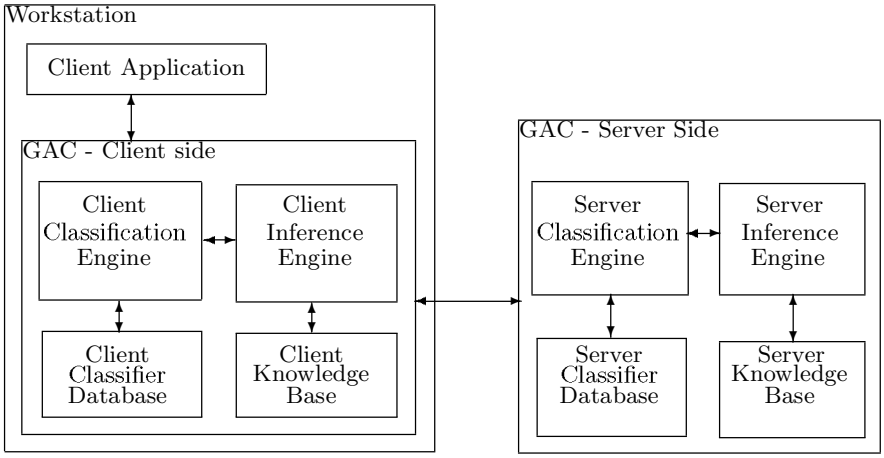

Fig. 1. The global architecture of the Generic Adaptive Classification (GAC) system. The system is divided in two parts: a client side and a server side. The server side is able to gather experiences from several clients and consequently the server side adaptive classifier will converge faster than its client side counterpart. On the other side, the client side classification engine will be able to adapt better but slower to the particular needs of a specific client application.

The client application is interested to receive a classification (category) for the thing or situation.

- Beside the *AV pairs* received from the client application the client side component of the GAC uses also its own derived *AV pairs*. These derived *AV pairs* will be obtained by the Client Inference Engine using the available Client Knowledge Base. For example, Client Inference Engine and Client Knowledge Base can have logic programming foundations. In an e-mail classification system used by an organization is important to establish if the sender of a message is a person that is hierarchically superior. This meaning can be captured by the following logical rule:

$$sender(L, X) \wedge destination(L, Y) \wedge boss(X, Y) \rightarrow important(L)$$

  The values of the attributes *sender* and *destination* are provided by the client application. The Client Knowledge Base contains the definition of the binary relation *boss*. Using the above logical rule it can be obtained the value for the derived attribute *important*.
- The client side of the GAC sends the *AV pairs*, which do not raise privacy concerns (personal or sensitive information) to the server side of the GAC to perform a partial classification. The server side of the GAC allows to combine the experiences of several clients. The server side of the GAC

receives far more examples to classify than a particular client and, consequently, the adaptive classification engine from server will converge faster than its client side counterparts. The server side will first try to derive new *AV pairs* by using Server Inference Engine and Server Knowledge Base. After that, Server Classification Engine will compute a partial classification that will be sent back to the client side of the GAC.
- The Client Classification Engine will perform its own inner classification using the base *AV pairs* received from the client application, the derived *AV pairs* and the partial result received from server which will be considered as a separate attribute.
- The client side of the GAC will return to the client application an unique identifier of the example and the result of the classification process. The client application will use the unique identifier to provide feedback information about the quality of the classification result. The client side of GAC will use the feedback to adaptively improve the performances of the Client Classification Engine. The feedback will be also sent to the Server Side of the GAC to update Server Classification Engine.

The description of the collaborative classifier construction framework should also discuss the program configuration. Both the client side and the server side must be configured before use. The configuration will include the attributes that will be used during classification process *(attribute name, attribute type (optional) and the attribute values)*. The possible categories (classes) and the contents of the Client and Server Knowledge Bases should be also considered.

Privacy was always a concern for the collaborative systems. A mutual authentication, authorization and privacy protocol between the client classifier and the server classifier should be devised. In this way, the server will not accept false clients and the client false servers that can provide wrong classification results or wrong feedback information.

3.2 Applications

E-mail classification system. The inputs of the e-mail classification system will include among other things the e-mail source address, the classification of the e-mail subject, the classification of the attachments, the importance specified by sender and the classification of the e-mail content. The output will consist in a category assigned to the e-mail. The category can be a simple binary one like *spam* and *not spam* or a more elaborated an such a *rank* indicating the desirability of the received e-mail. The e-mail classification system involves collaboration with a text classification system used to categorize the subject and the body of the e-mail and with a binary payload classification system used to classify the attachments. These two classification subsystems provide base input attributes for the main e-mail classification system.

Adaptive Web recommendation system. A Web recommendation system will maintain a decision tree for each registered user of the Web site. This tree

will receive as inputs the AV pairs that describe a product and will return a discrete rank representing the willingness of the customer to buy the product. The products with the highest ranks will be recommended to the customer.

4 Conclusion

The paper outlines a collaborative classifier construction framework built around $ADAPTTREE0$, an adaptive decision tree building algorithm. The framework was devised to be used by several intelligent agents that must perform similar classification tasks. The framework promotes collaboration between these agents and through cooperation the agents have several benefits: they can adjust their adaptive classification mechanisms and they skip a classification problem if the problem was already solved by other trusted agents. The role and functionality of the all system components is presented as well as the details of the interaction between the client applications and the generic classifier system. It worths also to mention the $ADAPTTREE1$ decision tree building algorithm introduced by the paper, an algorithm derived from $ADAPTTREE0$ that decomposes the initial reinforcement learning problem in several small reinforcement learning problems.

Acknowledgments. The research was supported by the Romanian National University Research Council (AT grant No. 102/2007).

References

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

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

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

[4] **T. M. Mitchell**:- Machine Learning, McGraw Hill, 1997

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

[6] **M. Preda**:- Adaptive Building of Decision Trees by Reinforcement Learning, The 7th WSEAS International Conference on Applied Informatics and Communications (AIC'07), Athens, Greece, 2007, 34-39.

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

[8] **L. D. Pyeatt**:- Reinforcement Learning with Decision Trees, Applied Informatics, AI 2003, Innsbruck, Austria, Acta Press, 2003.

[9] **L. Rokach, O. Maimon**:- Top-Down Induction of Decision Trees Classifiers - A Survey, IEEE Transactions on Systems, Man and Cybernetics, 35(4), 2005, 476-487.

[10] **R. S. Sutton, A. G. Barto**:- Reinforcement Learning: An Introduction, A Bradford Book, The MIT Press, Cambridge, Massachusetts London, England, 1998