Abduction in Classification Tasks

Maurizio Atzori, Paolo Mancarella, and Franco Turini

Dipartimento di Informatica
University of Pisa, Italy
{atzori,paolo,turini}@di.unipi.it

Abstract. The aim of this paper is to show how abduction can be used in classification tasks when we deal with incomplete data. Some classifiers, even if based on decision tree induction like C4.5 [1], produce as output a set of rules in order to classify new given examples. Most of these rule-based classifiers make the assumption that at classification time we can know all about new given examples. Probabilistic approaches make rule-based classifiers able to get the most probable class, on the basis of the frequency of the missing attribute in the training set [2]. This kind of assumption sometimes leads to wrong classifications. We present an abductive approach to (help to) choose which classification rule to apply when a new example with missing information needs to be classified, using knowledge about the domain.

1 Introduction

Due to the availability of large amounts of data, easily collected and stored via computer systems, the field of so-called data mining is gaining momentum. Several important results have been obtained in the context of specific algorithms, in applying the techniques in several application fields, and in designing suitable environments in which the data mining step can be embedded. Such environments support the phases that come before (e.g. cleaning the data) and the ones that come after (e.g. visualization of results), and attempt also at providing a context in which one can process the results of the data mining step in order to answer higher level questions than the ones directly provided by the computed data mining model.

For example, extracting the association rules from a set of supermarket transactions is very useful, and can answer basic questions like "which are the items that induce a buying attitude towards other items?", but it would be even more interesting answering the question "did the new layout of the store influence the buying attitude?". Answering the last question requires taking the association rules computed with respect to the old layout and comparing them with the ones computed according to the new ones, possibly taking into account the rules underneath the new design.

In brief, we believe that the results of data mining algorithms may be the input to a reasoning environment, where high level questions can be answered by the exploitation of both the results of data mining steps and some "domain knowledge". A general environment suited for this endeavor is described in [3].

In this paper we concentrate on a very focussed goal: proving that a careful representation of the results of a data mining algorithm (a decision or classification tree in this case), a careful representation of extra domain knowledge (constraints in the case at hand), and a careful choice of a reasoning technique (abduction in the case at hand) can substantially improve the behavior of the extracted model (the activity of classification in the case).

Classification trees are one of the main models extracted from web logs data in the *Clickworld* Project, that aims at improving the management of web sites through knowledge discovery means.

In Sect. 2 we sketch some basic concepts on data mining and decision trees and on abductive reasoning, and we will set up the notations and terminology used throughout the paper. In Sect. 3 we formalize the concept of classification in decision trees and formally show how it can be viewed as an abductive problem. In Sect. 4 we show that the abductive view of the classification tasks suggests useful extensions of the latter by exploiting domain-specific knowledge represented as integrity constraints. Finally, in Sect. 5 we draw some lines of future research on the subject.

2 Preliminaries

In order to understand the idea that we are going to describe later in this paper, we briefly review some background on data mining and decision trees and on abduction.

2.1 Data Mining

Data mining can be defined as the process of finding correlations or patterns among dozens of fields in large relational databases. It is an essential step of the knowledge discovery process where intelligent methods are applied in order to extract data patterns from very large databases [4]. In particular, we are interested in the classification task, that is predicting categorical labels given some examples.

Classification of data requires two sequential steps: the first one consists in building a model that describes a given set of examples by associating a class label to each of them; the second one concerns using the model to classify new examples (i.e. predict the categorical label). The only model we are interested in in this paper is the one of the *decision trees* that we are going to briefly describe.

Decision Trees. A decision tree is a tree structure in which each internal node denotes a test on an attribute, each branch represents an outcome of the test and leaf nodes represent classes. Decision tree induction consists in building such a tree from a training set of examples and then using it (following a path from the root to a leaf) to classify new examples given their attribute values. Because of their structure, it is natural to transform decision trees into classification rules, that can be easily inserted into a reasoning framework. Notice that some machine learning tools, such as C4.5 [1], already includes a class rulesets generator. In the sequel we will see how class rulesets can be embedded into an abductive reasoning framework which will allow us, in some cases, to better classify new examples in presence of external information, such as specific domain knowledge.

Let us now set up the main notations and terminologies we will use throughout the paper as far as decision trees are concerned. Let $\mathcal{A}$ be a set of attribute names and $\mathcal{C}$ be a set of classes (possible classifications). For simplicity, we assume that each attribute can be assigned a value over a finite set of values $\mathcal{V}$. An *example* e is a set of attribute/values pairs

$$e = \{a_1 = v_1, \ldots, a_n = v_n\}$$

where all the a_is are distinct attribute names.

Definition 1. *A* d*ecision tree T over $\mathcal{A}$ and $\mathcal{C}$ is a tree such that:*

(i) *each non-leaf node is labelled by an attribute $a \in \mathcal{A}$;*
(ii) *each leaf node is labelled by a class $c \in \mathcal{C}$;*
(iii) *each branch is labelled by a value $v \in \mathcal{V}$;*
(iv) *the values labelling all the branches exiting from a given node are all distinct;*
(v) *the labels of a path are all distinct.*

Notice that *(v)* formalizes the fact that, in each path, only one test can be performed on each attribute.

Example 1. Let us consider a very well known example, taken from [1]. Given a training set of examples which represent some situations, in terms of weather conditions, in which it is or it is not the case that playing tennis is a good idea, a decision tree is built which can be used to classify further examples as good candidates for playing tennis (class Yes) and bad candidates to play tennis (class No). Table 1 shows the original training set, given as a relational table over the attributes $\{Outlook, Temperature, Humidity, Wind\}$. The last column of the table represents the classification of each row.

Using standard decision trees inductive algorithms (e.g., ID3), we may obtain the decision tree in Fig. 1 from the above training set. As we have already pointed out, each internal node represents a test on a single attribute and each branch represents the outcome of the test. A path in the decision tree represents the set of attribute/values pairs that an example should exhibit in order to be classified as an example of the class labelled by the leaf node. For instance, given the above tree, the example $\{Overlook = Sunny, Humidity = Low\}$ is classified as Yes, whereas the example $\{Overlook = Sunny, Humidity = High\}$ is classified as No. Notice that not all the attribute values have to be specified in order to find a classification of an example. On the other hand, if an example is too under-specified, it may lead to different, possibly incompatible, classifications. For instance, the example $\{Overlook = Sunny\}$ can be classified both as Yes or No, following the two left-most branches of the tree. It is also worth noticing that the decision tree may not consider all the attributes given in the training set. For instance, the attribute $Temperature$ is not taken into account at all in the previous decision tree.

Table 1. Training set of examples on attributes *Overlook, Temperature, Humidity, Wind*. The values *Yes* and *No* represent the concept to learn

Outlook	Temperature	Humidity	Wind	Class
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rainy	Mild	High	Weak	Yes
Rainy	Cool	Low	Weak	Yes
Rainy	Cool	Low	Strong	No
Overcast	Cool	Low	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Low	Weak	Yes
Rainy	Mild	Low	Weak	Yes
Sunny	Mild	Low	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Low	Weak	Yes
Rainy	Mild	High	Strong	No

2.2 Abductive Logic Programming

Abduction is a particular form of reasoning introduced by the philosopher Peirce [5] as a form of synthetic reasoning which infers the case from a rule and a result, i.e. from the fact that A implies B and the observation B, we can abduce A. Abduction can be viewed as the probational adoption of an hypothesis as an explanation for observed facts, according to known laws. Obviously, as Peirce noticed, "it is a weak kind of inference, because we cannot say that we believe in the truth of the explanation, but only that it may be true".

In recent years, abduction has been widely studied and adopted in the context of logic programming [6], starting from the work [7]. In this paper, we basically

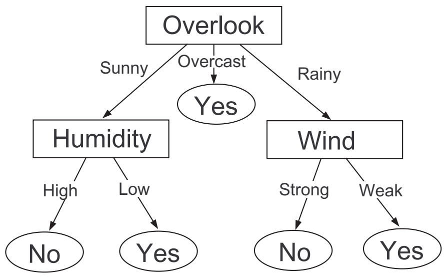

Fig. 1. A well-know example of decision tree, obtained from the training set of Table 1

adopt the definitions and terminologies of [8, 9], that we are going to briefly review next.

An Abductive Logic Programming (ALP) framework is defined as a triple $\langle P, A, Ic\rangle$ consisting of a logic program, P, a set of ground abducible atoms A and a set of logic formulas Ic, called the integrity constraints. The atoms in A are the possible abductive hypotheses which can be assumed in order to explain a given observation in the context of P, provided that these assumptions are consistent with the integrity constraints in Ic. In many cases, it is convenient to define A as a set of (abducible) predicate symbols, with the intended meaning that each ground atom whose predicate symbol is in A is a possible hypothesis.

Given an abductive framework as before, the general definition of abductive explanation is given as follows.

Definition 2. *Let $\langle P, A, Ic\rangle$ be an abductive framework and let G be a goal. Then an* abductive explanation *for G is a set $\Delta \subseteq A$ of ground abducible atoms such that:*

- $P \cup \Delta \models G$
- *$P \cup \Delta \cup Ic$ is consistent.*

Example 2. Let $\langle P, A, Ic\rangle$ be an abductive framework where:

- P is the logic program given by the three rules: $p \leftarrow a \quad p \leftarrow b \quad q \leftarrow c$
- $A = \{a, b, c\}$
- $Ic = \{\}$.

Then, there are three abductive explanations for the goal[1] (p, q) given by $\Delta_1 = \{a, c\}$, $\Delta_2 = \{b, c\}$ and $\Delta_3 = \{a, b, c\}$.

Consider now the abductive framework $\langle P, A, Ic'\rangle$, where P and A are as before and Ic contain the formula $\neg(a, c)$. In this new framework, there is only one explanation for the goal p, q, which is Δ_2 above, since the second potential explanation is inconsistent with the integrity constraint.

The given notion of abductive explanation can be easily generalized to the notion of abductive explanation given an initial set of abducibles Δ_0.

Definition 3. *Let $\langle P, Ab, Ic\rangle$ be an abductive framework, Δ_0 be a set of abducibles and G be a goal. We say that Δ is an abductive explanation for G* given *Δ_0 if $\Delta_0 \cup \Delta$ is an abductive explanation for G.*

Notice that this implies that the given set of abducibles Δ_0 must be consistent with the integrity constraints Ic. In some cases, we may be interested in *minimality* of abductive explanations.

Definition 4. *Let $\langle P, Ab, Ic\rangle$ be an abductive framework, Δ_0 be a set of abducibles and G be a goal. We say that Δ is a Δ_0-minimal explanation for G if Δ is an explanation for G given Δ_0 and for no proper subset Δ' of Δ ($\Delta' \subset \Delta$), Δ' is an explanation for G given Δ_0.*

[1] As in standard logic programming, "," in rule bodies and goals denotes logical conjunction.

For instance, in Example 2 with $Ic = \{\}$, Δ_0 is the empty set (we do not have an initial set of "pre-abduced" abducibles) and both Δ_1 and Δ_2 are Δ_0-minimal explanations for (p, q) while Δ_3 is not (in fact $\Delta_1 \subset \Delta_3$).

The notion of abduction as described above has a natural computational counterpart within a proper extension of the standard SLD-based computational engine of logic programming. In particular, when integrity constraints are in the form of *denials*, i.e. in the form $\neg(p_1, \ldots, p_n)$, the so called KM-proof procedure can be successfully adopted to compute (minimal) abductive explanations. Due to lack of space we omit the details of the definition of the KM-proof procedure which can be found in [8, 9].

3 Classification as an Abductive Problem

In this Section we show how the classification task using decision trees can be directly seen as an abductive problem. To do this, we first need to spend some words on the notation and then formally define what decision tree classification means and on the transformation of decision trees into abductive logic programs.

Given a tree T and a path π in T, we denote by $Attr(\pi)$ the example $\{a_1 = v_1, \ldots, a_k = v_k\}$ where each a_i is a non-leaf node in T and each v_i is the attribute value labelling the branch exiting from a_i. Moreover, we denote by $Class(\pi)$ the class labelling the leaf node of π.

The following defines the concept of classifying an example by means of a decision tree in case of missing information. The usual definition of classifying an example is given as a special case, in which there are no missing information.

Definition 5. *Let* T *be a decision tree,* e *be an example and* c *be a class. We say that the example* e **may be** *classified as* c *by* T *via* δ*, denoted by* $T \overset{c,\delta}{\Longrightarrow}$ e *if there exists a path* π *in* T *with leaf node* $Class(\pi) = c$ *such that* $e \cup Attr(\pi)$ *is an example and* $\delta = Attr(\pi) \setminus e$*.*
If $T \overset{c,\{\}}{\Longrightarrow}$ e *we say that the example* e **is** *classified as* c *by* T*, and we simply write* $T \overset{c}{\Longrightarrow}$ e

Notice that, in $T \overset{c,\delta}{\Longrightarrow} e$, the condition that $e \cup Attr(\pi)$ must be an example ensures that the attribute values of e are compatible with the set of tests represented by π. In other words, if for some attribute a, $a = v \in e$ and $a = v' \in Attr(\pi)$, then $v = v'$. Moreover, notice that δ represents the extension to the example e which is needed in order to classify it as c from the chosen path.

Example 3. Let T be the decision tree of Example 1. We have:

– $T \overset{Yes}{\Longrightarrow} \{Overlook = Sunny, Wind = Strong, Humidity = Low\}$
– $T \overset{No,\delta}{\Longrightarrow} \{Overlook = Sunny\}$, where $\delta = \{Humidity = High\}$

Now, let us explain how to transform a decision tree into an abductive logic program. Let T be a decision tree and π be a path in T. The rule r_π associated with π is the Horn clause $c \leftarrow a_1(v_1), \ldots, a_n(v_n)$ such that:

- $Class(\pi) = c$, and
- $Attr(\pi) = \{a_1 = v_1, \ldots, a_n = v_n\}$

Notice that attribute names are viewed as unary predicate symbols, and that an attribute/value pair $a = v$ is mapped into an atom $a(v)$. Moreover, by *(v)* in Def. 1, given a path π, $Attr(\pi)$ is an example. The Horn clause program P_T associated with a decision tree T is the set of rules

$$P_T = \{r_\pi \,|\, \pi \text{ is a path in } T\}.$$

Finally, we associate with T the set IC_T of *canonical* integrity constraint containing a denial

$$\leftarrow a(x), a(y), x \neq y$$

for each attribute $a \in \mathcal{A}$.

As we will see next, attribute names are viewed as abducible predicates, and sets of abducibles will represent possible classifications. The integrity constraints we have just defined ensure that in each consistent set of abducibles, an attribute occurs at most once (cfr. case (v) of Def. 1).

Given an example $e = \{a_1 = v_1, \ldots, a_k = v_k\}$, let Δ_e be the set of atoms

$$\Delta_e = \{a_1(v_1), \ldots, a_k(v_k)\}.$$

We have set up all the ingredients which are needed to associate an abductive framework to a decision tree T.

Definition 6. *Given a decision tree T, the abductive framework AB_T associated with T is the triple*

$$AB_T = \langle P_T, \mathcal{A}, IC_T \rangle$$

where P_T is the Horn program associated with T, IC_T is the set of canonical integrity constraints associated with T and $\mathcal{A}$ is the set of attribute names.

The following theorem formalizes the correspondence between classifications in T and abductive explanations in AB_T.

Theorem 1. *Let T be a decision tree and AB_T be the corresponding abductive framework. Let also* e *be an example. Then Δ is a Δ_e-minimal explanation for* c *with respect to AB_T if and only if for some path π in T we have $T \overset{c,\delta}{\Longrightarrow} e$ and $\Delta = \Delta_\delta$.*

Proof.
($\Longleftarrow$) Assume that $T \overset{c,\delta}{\Longrightarrow} e$. Then, by Def. 5, for some path π in T, $e \cup Attr(\pi)$ is an example and $\delta = Attr(\pi) \setminus e$. Consider the rule r_π

$$c \leftarrow a_1(v_1), \ldots, a_n(v_n).$$

and let $body(r_\pi) = \{a_1(v_1), \ldots, a_n(v_n)\}$. It is clear that $body(r_\pi) = \Delta_{Attr(\pi)}$ and $P_T \cup body(r_\pi) \models c$. Let $\Delta = body(r_\pi) \setminus \Delta_e$: clearly, $P_T \cup \Delta_e \cup \Delta \models c$ and $\Delta =$

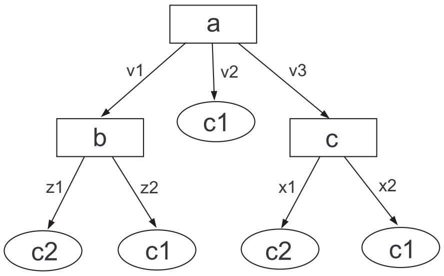

Fig. 2. A simple example of decision tree

Δ_δ. Moreover, since $e \cup Attr(\pi)$ is an example and $(\Delta_e \cup \Delta) \subseteq \Delta_{e\cup Attr(\pi)}$, $P_T \cup \Delta_e \cup \Delta \cup IC_T$ is consistent. Hence, Δ is an explanation for c given Δ_e. By construction, Δ is also a Δ_e-minimal explanation.
$(\Longrightarrow)$ Assume Δ is a Δ_e-minimal explanation for c. Clearly, $\Delta \cap \Delta_e = \{\}$ by minimality of Δ. Since $P_T \cup \Delta_e \cup \Delta \models c$, by construction of P_T there exists a path π in T such that r_π is a rule of the form

$$c \leftarrow a_1(v_1), \dots, a_n(v_n)$$

and $\{a_1(v_1), \dots, a_n(v_n)\} \subseteq \Delta_e \cup \Delta$.

Let $\delta = Attr(\pi) \setminus e$. By construction and minimality of Δ, we have $\Delta = \Delta_\delta = \{a_1(v_1), \dots, a_n(v_n)\} \setminus \Delta_e$. Finally, it is clear that $e \cup Attr(\pi)$ is an example, by the consistency of $P_T \cup \Delta_e \cup \Delta \cup IC_T$ and the observation that $\Delta_{e\cup Attr(\pi)} = \Delta_e \cup \Delta$. $\square$

Example 4. Let us consider the following decision tree (which is nothing but the decision tree of Example 1 using symbolic names and symbolic values). In the abductive framework $AB_T = \langle P_T, \mathcal{A}, IC_T \rangle$ associated with T we have:

- P_T is the following set of rules:
  $c1 \leftarrow a(v2).$ $\quad c2 \leftarrow b(z1), a(v1).$
  $c1 \leftarrow b(z2), a(v1).$ $\quad c2 \leftarrow c(x1), a(v3).$
  $c1 \leftarrow c(x2), a(v3).$
- IC_T is the following set of canonical integrity constraints:
  $\leftarrow a(x), a(y), x \neq y.$ $\quad \leftarrow b(x), b(y), x \neq y.$ $\quad \leftarrow c(x), c(y), x \neq y.$
- $\mathcal{A} = \{a, b, c\}$.

Let us consider the example $e = \{a = v1, b = z1\}$ and the corresponding initial set of abducibles $\Delta_e = \{a(v1), b(z1)\}$. Clearly $\Delta = \{\}$ is a Δ_e-minimal explanation for $c2$, and indeed it corresponds to the leftmost path in the tree. Notice that also $\Delta' = \{c(x1)\}$ is an explanation for $c2$ given Δ_e, but it is not Δ_e-minimal and indeed it does not correspond to any path in the tree matching the attribute/values pairs given in e.

4 Improving Classification Exploiting Domain Specific Knowledge

The abductive view of classification given in the previous Section can be seen as an alternative, though equivalent, way of performing classification given some decision tree. It is worth noting that the abductive reasoning required in this alternative view is very limited and does not fully exploit the potential power of abductive reasoning in logic programming as sketched in Sect. 2.2. In particular, the transformation requires the use of a pre-defined, canonical set of integrity constraints that simply avoid explanation to contain different values for the same attribute. We show in this Section that one way to exploit abductive reasoning is to add *domain specific* knowledge in order to improve the classification task. Domain specific knowledge may be taken into account during classification in many ways, e.g. to rule out some classifications or to prefer one particular classification over another. Up to our knowledge, taking into account domain specific knowledge in standard decision-tree based classification algorithms may be not straightforward and may require substantial modifications of these algorithms. On the other hand, abductive frameworks, and existing concrete implementations of them, are already equipped with mechanisms that can be directly exploited to represent and handle domain specific knowledge.

In this paper we exploit integrity constraints (beyond the canonical ones) in abductive frameworks as a way to express domain specific knowledge. Let us show some examples.

Example 5. Consider the decision tree T of Example 4. As we have already pointed out, the attributes which label the internal nodes of decision trees may not be all the attributes that examples are equipped with. In the current example, assume that the original set of attributes contained also an attribute d, beyond the attributes $\{a, b, c\}$ occurring in the tree. Assume now that an example is given $e = \{d = w1\}$: it is clear that, since d does not occur in the decision tree, any classification of e can be done using T. However, assume that we have extra knowledge on the domain at hand, expressing the fact that the value $w1$ for the attribute d is incompatible both with the value $v1$ of the attribute a and with the value $x1$ of the attribute c. It is easy to see that this extra knowledge can be used to classify the example e as belonging to class $c1$. In the abductive framework corresponding to T the extra knowledge can be easily coded by adding the following integrity constraints:

$$\neg(d(w1), a(v1)) \qquad \neg(d(w1), c(x1))$$

If we consider now the full abductive framework $\langle P, \mathcal{A}', Ic\rangle$, where P is the program obtained as in Example 4, $\mathcal{A}' = \{a, b, c, d\}$ and Ic are the integrity constraints we have just shown, we observe that $c2$ has no abductive explanation given $\Delta_e = \{d(w1)\}$, whereas the goal $c1$ has two Δ_e-minimal explanations given Δ_e, namely $\Delta_1 = \{a(v2)\}$ and $\Delta_2 = \{a(v3), c(x2)\}$. Notice the correspondence between these two solutions and the two right-most paths with $c1$ as the leaf node. From a computational point of view the two explanations can be computed using, e.g., the KM-proof procedure. The consistency checking phase of the proof procedure rules out the two potential Δ_e-minimal explanations $\{b(z1), a(v1)\}$ and $\{a(v3), c(x1)\}$ for the goal $c2$. Indeed, the first one is inconsistent with the integrity constraint $\neg(d(w1), a(v1))$, and the second one is inconsistent with the integrity constraint $\neg(d(w1), c(x1))$.

As the previous example points out, integrity constraints can be used to add knowledge relating the values of the various attributes, including those which do not occur in the original decision tree. This can help and improve the classification task. The next example shows that integrity constraints may add knowledge which is relevant to the attributes already occurring in the decision tree, although it is not explicit in the decision tree itself.

Example 6. Let us consider again the decision tree of Example 1. Assume that, as it is often the case, whenever there is strong wind the humidity is not high:

$$\neg(Humidity(High), Wind(Strong)).$$

It is important to point out that this kind of knowledge may not be implicit in the training set from which the original decision tree was built. Actually, the examples in the training set may even contradict this knowledge (see, e.g., Table 1). Indeed, this knowledge may arise from knowledge sources different from the ones which provide the training set (in this particular example this knowledge may be associated with typical weather forecast knowledge bases). Assume now that we want to classify an example described simply as $e = \{Overlook = Sunny, Wind = Strong\}$. In the corresponding abductive framework, given the initial set $\Delta_e = \{Overlook(Sunny), Wind(Strong)\}$, the classification Yes has an abductive explanation $\Delta_1 = \{Humidity(Low)\}$ and the classification No has an abductive explanation $\Delta_2 = \{Humidity(High)\}$. If we consider now the above integrity constraint, the abductive explanation Δ_2 is ruled out, due to the fact that the full explanation given by $\Delta_e \cup \Delta_2 = \{Overlook(Sunny), Wind(Strong), Humidity(High)\}$ is inconsistent. Thus, by adopting the very same computational mechanism we obtain a correct classification with respect to the extra domain specific knowledge.

In many cases, decision trees can be equipped with probability measures on the outcome of attribute/values test. In other words, each branch of the tree is labelled both by a value corresponding to the attribute labelling the parent node, as well as with a probability measure which indicates how likely is that an observation exhibits this value for the given attribute. This kind of probability information is clearly useful when we try to classify new examples with missing attribute information. Even in this case, extra domain knowledge can be taken into account in the classification task in order to dynamically get better probability measures on possible classification of new under-specified examples. To show this let us consider a very simple example.

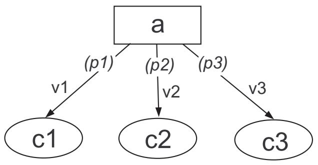

Fig. 3. A simple example of a decision tree with probabilities associated to each branch

Example 7. Let us consider the simple decision tree in Fig. 3. There is only one internal node, the root, which represents a test on a 3-valued attribute named a. Probabilities are represented between parenthesis. The classifications rules extracted from the above tree are the following:

$$c1 \leftarrow a(v1) \qquad c2 \leftarrow a(v2) \qquad c3 \leftarrow a(v3).$$

Notice that we could have more information about a new observation than the a attribute only, but the tree-based classification (and its associated class ruleset) will use only information about a. But what happens if we know nothing about a? If we try to classify a new example with no information about a we get as output the class with highest probability. Again, using integrity constraints in the corresponding abductive framework which makes explicit extra information about the domain, we can cut some branches during the classification task, thus obtaining better classifications and also dynamically improving probability values. For instance, let this domain knowledge be represented by the integrity constraint $\neg(a(v1), b(z2))$. In this case, if we are asked to classify the example $\{b = z2\}$ we will discard the classification $c1$ whose abductive explanation $\Delta_1 = \{a(v1)\}$ is inconsistent with the integrity constraint. On the other hand, we would compute the classifications $c2$ and $c3$ with the corresponding explanations $\Delta_2 = \{a(v2)\}$ and $\Delta_3 = \{a(v3)\}$. We can also dynamically re-compute the associated probabilities which will be $\frac{p2}{p2+p3}$ as far as $c2$ is concerned, and $\frac{p3}{p2+p3}$ as far as $c3$ is concerned.

5 Conclusions

Starting from the idea that framing the data mining step into a logic framework provides us with the possibility of exploiting also domain knowledge in the knowledge extraction and exploitation process, we showed that the result of a classification tree can be improved if the tree is not simply traversed, but it is visited in abductive mode. Indeed, the basic observation is that the application of a classification tree to a new observation can be viewed as a straightforward abductive computation, as soon as the tree is represented as a collection of rules in the obvious way. This observation opens up a world of possibilities, since the abductive computations can become very sophisticated, and they can take into account several types of knowledge. In this paper we restricted our attention to domain knowledge represented as simple constraints involving equalities, conjunction and negation. However, even with this limited power, we could show examples in which the classification task was radically improved. We are confident that more sophisticated use of knowledge can lead us to much better improvements.

Several frameworks in the context of logic programming have been already developed in order to integrate induction and abduction (see, e.g., [10, 11, 12]).

The framework we have outlined is anyway the first one, as far as we know, in which the inductive task performed by external machine learning algorithms is not influenced by abductive procedures, but post-processed through abduction reasoning in order to take into account the domain knowledge and to improve in some cases the classification task itself.

Acknowledgments

This work has been carried out within the project *Clickworld* (MIUR S585-P).

References

[1] Quinlan, J. R.: C4.5: programs for machine learning. Morgan Kaufmann (1993) 213, 215
[2] Quinlan, J. R.: Unknown attribute values in induction. In: Proc. of the Sixth International Machine Learning Workshop, Morgan Kaufmann (1989) 164–168 213
[3] Giannotti, F., Manco, G., Pedreschi, D., Turini, F.: Experiences with a logic based knowledge discovery support environment (Springer-Verlag 1999) 213
[4] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000) 214
[5] Peirce, C. S.: (Collected papers of Charles Sanders Peirce) Vol. 2, Hartshorn et al. eds., Harvard Univesity Press. 216
[6] Kakas, A., Kowalski, R., Toni, F.: Abductive logic programming. Journal of Logic and Computation **2** (1993) 719–770 216
[7] Eshgi, K., Kowalski, R.: Abduction compared with negation by failure. In Levi, G., Martelli, M., eds.: Proc. of the 1989 International Conference on Logic Programming, MIT Press (1989) 234–254 216
[8] Kakas, A., Mancarella, P.: Generalized stable models: a semantics for abduction. In: Proc. of 9th European Conference on Artificial Intelligence, Pitman (1990) 385–391 217, 218
[9] Kakas, A., Michael, A., Mourlas, C.: Aclp: Abductive constraint logic programming. Journal of Logic Programming **44** (2000) 129–177 217, 218
[10] Lamma, E., Mello, P., Milano, M., Riguzzi, F.: Integrating induction and abduction in logic programming. Information Science **116** (1999) 25–54 224
[11] Kakas, A., Riguzzi, F.: Abductive concept learning. New Generation Computing **18** (2000) 224
[12] Flach, P. A., Kakas, A. C., eds.: Abduction and Induction: Essays on their relation and integration. Kluwer Academic Publishers (2000) 224