A Higher-order Approach to Meta-learning

H. Bensusan, C. Giraud-Carrier, and C. J. Kennedy

Department of Computer Science
University of Bristol
Merchant Venturers Building
Woodland Road
Bristol BS8 1UB, UK
{hilanb,cgc,kennedy}@cs.bris.ac.uk

Abstract. Meta-learning, as applied to model selection, consists of inducing mappings from tasks to learners. Traditionally, tasks are characterised by the values of pre-computed meta-attributes, such as statistical and information-theoretic measures, induced decision trees' characteristics and/or landmarkers' performances. In this position paper, we propose to (meta-)learn directly from induced decision trees, rather than rely on a hand-crafted set of pre-computed characteristics. Such meta-learning is possible within the framework of the typed higher-order inductive learning framework we have developed.

1 Introduction

Most current approaches to meta-learning for model selection rely on some form of task characterisation, which allows a meta-learner to induce a mapping from particular characteristics to suitable learning models. To date and to our knowledge, three distinct kinds of task characterisations have been used, as follows.

1. *Statistical and information-theoretic characterisation* [1, 7, 14, 16]. Here, tasks are characterised by a number of measures computed directly from their datasets (e.g., number of attributes, number of target concepts, class entropy, concept concentration, mutual information, signal-to-noise ratio).
2. *Landmarking* [15]. Here, tasks are characterised by how well, in terms of predictive accuracy, a set of simple and efficient learning algorithms (i.e., the landmarkers) perform on them. The assumption is that the performances of the landmarkers give an indication of the performances of other learners.
3. *Decision-tree-based characterisation* [2, 3]. Here, tasks are characterised by a number of measures (e.g., depth, shape, balance) computed from an unpruned decision tree induced from them. The assumption is that the structure of unpruned induced decision trees summarises salient features of data/tasks (e.g., depth is dependent upon noise and class distribution).

Using induced models (as in 3 above) can be viewed as shifting the search space of the meta-learner from the example space to the hypothesis space. Although any characterisation leads to a new search space, there is something rather attractive in using as search space at the meta-level the hypothesis space of a base-level learning algorithm. In particular, provided the hypothesis space is rich enough and the (base-level) learner can search it effectively, the induced hypotheses may be expected to compress and summarise the original data, thus offering more meaningful features to the meta-learner. In some sense, this shift may effect a kind of useful (see [1]) feature selection and construction for meta-learning.

Motivated by extensive empirical evidence highlighting a number of important connections between tasks' characteristics and unpruned induced decision trees' structures, as well as the results reported in [2, 3], we propose to take the third approach above one step further by using induced trees directly for task characterisation and letting an appropriate meta-learner "discover" those features relevant to model selection. In this way, the meta-learner need not be limited by a pre-defined set of meta-attributes.

In order to work directly with trees, we make use of the typed higher-order inductive learning framework we have developed recently [4–6, 8, 9, 12, 13]. In principle, this framework allows the representation of arbitrarily rich structures. Hence, it would be conceivable to characterise a task by several induced models (e.g., a decision tree, a neural network, etc), as well as any other relevant information (e.g., landmarkers' performances as in 2 above). Here, we restrict our attention to characterisation by a single induced decision tree.

The paper is organised as follows. Section 2 gives a brief overview of our typed higher-order inductive learning framework. Section 3 reviews previous work on tree characteristics and shows how induced trees could be represented. It also gives an idea of the form of the hypotheses that a meta-learner would induce from such representations. Finally, section 4 concludes the paper.

2 Typed Higher-order Inductive Learning

In order to capture complex physical structures, such as trees, as directly as possible, we need correspondingly complex abstract data structures or types. Our representation formalism, whose details are in [5, 12], is expressive enough to allow the representation of arbitrary structures as closed terms of the corresponding type. Thus, all information is typed.

We note here that, in this sense, our formalism is a natural extension of the attribute-value framework. Indeed, in the attribute-value language, each attribute has a type (e.g., nominal, real) and examples are tuples (another simple type) of constants drawn from the domains of the corresponding types. We wish to make types an intrinsic part of first and higher-order representations. This enables the examination of possible relations among distinct elements that have the same type whilst limiting the associated explosion of the search space. We use the programming language Escher [11] as the vehicle for typed higher-order logic knowledge representation. Escher is an integrated functional and logic programming language, based on Church's theory of types. The syntax of Escher coincides with that of Haskell for the functional subset and includes extensions for quantifiers and set constructs. A formal account of Escher is beyond the scope of this paper. For our purposes here, it is sufficient to state that Escher implements the necessary computational machinery to handle standard abstract data types such as tuples, lists, sets, multisets, trees and graphs. In fact, one can in principle construct any arbitrary abstract data type with its associated structure and operations.

As mentioned above, examples are represented as closed terms of the abstract data type that "best" captures the physical structure of their domain. We give a few examples to illustrate the approach. So as not to encumber the paper with unnecessary Escher-specific syntax, we stick to standard mathematical (or familiar) notation as much as possible.

Example 1. Consider a problem domain involving trains made up of load-carrying cars. Each car has a size and contains 0 or more loads, each of which consists of a type and number of instances. A useful abstraction for such trains is *lists* of cars, i.e.,

$$\texttt{type Train} = [\texttt{Car}].$$

Each car can in turn be represented by a *tuple* consisting of a size and a *set* of loads, i.e.,

$$\texttt{type Car} = (\texttt{Size}, \{\texttt{Load}\}),$$

and each load as a *tuple* consisting of a type and number of instances, i.e.,

$$\texttt{type Load} = (\texttt{Ltype}, \texttt{NumInstances}).$$

Appropriate base types such as enumerated types, integers and reals are also declared for `Size`, `Ltype` and `NumInstances`. The following is a train with 2 cars, one empty, and one carrying 3 black boxes and 1 red one.

$$[(\texttt{20.0}, \{\}), (\texttt{16.3}, \{(\texttt{BlackBox}, \texttt{3}), (\texttt{RedBox}, \texttt{1})\})]$$

Example 2. Consider a problem domain involving molecules made up of atoms connected by bonds. Each atom has a label and a type. Each bond has a type and connects two atoms. An useful abstraction for such molecules is *graphs*, where the vertices represent atoms and the edges represent bonds, i.e.,

$$\texttt{type Molecule} = (\{\texttt{Atom}\}, \{\texttt{Bond}\}).$$

We choose to define atoms as *tuples* of constants, i.e.,

$$\texttt{type Atom} = (\texttt{Label}, \texttt{Atype}),$$

and bonds as *tuples* consisting of a type and an *unordered pair* of atom labels, i.e.,

$$\texttt{type Bond} = (\texttt{Btype}, < \texttt{Label}, \texttt{Label} >).$$

Again, appropriate base types are declared for `Label`, `Atype` and `Btype`. The following is the simplest carbene, methylene (CH_2).

$$(\{(\texttt{1}, \texttt{H}), (\texttt{2}, \texttt{H}), (\texttt{3}, \texttt{C})\}, \{(\texttt{double}, < 3, 1 >), (\texttt{double}, < 3, 2 >)\})$$

We will see in the following section how decision trees can also be represented. A number of learning algorithms have been designed and implemented to manipulate the complex structures captured by the above representation. A discussion of these algorithms is beyond the scope of this paper. For details, we refer the reader to [5, 6] for the description of a decision tree learner, to [13] for the description of a sequential covering algorithm and to [9, 10] for the description of an evolutionary system.

3 Meta-learning from Induced Decision Trees

The experiments described in [2, 3] were the first ones to make use of information computed from induced decision trees to characterise tasks in meta-learning. Before showing our approach to automating the extraction of such information, we briefly revise the descriptors used in this earlier work.

3.1 Decision Tree Task Descriptors

The following lists the descriptors computed from induced decision trees and used for meta-learning in [2, 3]. Decision trees are assumed to be unpruned.

Nodes per attribute: the ratio of the number of tree nodes to the number of attributes. In tasks with large attribute vectors, this measure may indicate the amount of irrelevant attributes in the task, but more often it gives an idea of how many attributes are used as splitting points in more than one branch of the tree.

Nodes per instance: the ratio of the number of tree nodes to the number of training instances. This is also intended to measure the number of splits that the training set requires.

Average leaf corroboration: the average strength of support of each tree leaf. Support is measured by the number of training instances that correspond to the paths terminating in each leaf. This descriptor is intended to measure how much support each element of the tree receives from the sample.

Average *gain-ratio* difference: *gain-ratio* is the information-theoretic measure used for splitting in C4.5. It is designed to measure the quality of a splitting point in an attribute. This descriptor indicates the difference in the *gain-ratio* between the attributes at the first splitting point of the tree building process. Large *gain-ratio* difference indicates the presence of attributes that on their own carry information about the label and therefore that a linear partition oriented soft bias is likely to perform well.

Maximum depth: this measures the size of the longest path from the root to a leaf. The depth of a path is measured by the number of nodes it contains. The depth of the tree indicates how hard it is to represent the task by a decision tree.

Number of repeated nodes: this descriptor measures how many repeated attributes appear in the tree. This descriptor indicates the need for attribute redescription.

Shape: a function of the probabilities of arriving at the various leaves given a random walk down the tree. Assume the probability $p(N_i)$ of arriving at node N_i among the m sibling nodes from the ancestor N_A is given by:

$$p(N_i) = p(N_A)/m.$$

where $p(N_A) = 1$ if A is the root of the tree. The shape of a tree with n leaves is then measured using the probability of the leaves $p(L_j)$ by the expression

$$-\sum_{j=1}^{n} p(L_j)\log_2(p(L_j)).$$

Homogeneity: the number of leaves divided by tree shape. This shows how the leaves are distributed within the tree, indicating the strength of attribute-label correlations for the task.

Imbalance: given all possible values V_i for $p(L_j)$, define $G(V_i)$ as

$$G(V_i) = nV_i$$

where n is the number of times V_i occurs in the set of all the leaves of the tree. The imbalance is then measured by the following sum over all possible values for $p(L_j)$:

$$-\sum_{j=1}^{n} G(V_j)\log_2(G(V_j)).$$

Internal symmetry: this is the number of identical multi-node subtrees repeated in the tree. This descriptor, like number of repeated nodes, indicates the need for attribute redescription.

In the next subsection, we show how decision trees could be captured by our representation and some of the above information obtained automatically and on demand during meta-learning.

3.2 Representing and Learning from Induced Decision Trees

The standard representation for a tree, where the subtrees are ordered, is

```
data Tree T = Node T [Tree T];
```

Here `(Tree T)` is the type of a tree whose nodes have type `T`, and `Node` is a data constructor whose first argument is the root node and the second is the list of subtrees of the tree. Hence, the type of a tree whose nodes are integers would be given by

```
type TreeOfInt = Tree Int;
```

Note that the keyword `data` indicates the declaration of a type and the data constructors of that type, whilst the keyword `type` indicates a type synonym. For the sake of argument, let the type of a decision tree be given by

```
type DecisionTree = Tree DecisionNode;
```

We now have to define the type of the nodes, i.e., provide the data constructors for `DecisionNode`. The choice of what information to put into a node is driven by experience with decision tree learning.

It makes good sense to aggregate all of a node's information in a tuple. Hence, the type of `DecisionNode` is given by

```
type DecisionNode = (Attribute, NumExamples, MajClass,
                     NumMajClass, Heuristic);
```

where `Attribute` is the name of the attribute to be tested at the node, `NumExamples` is the number of examples associated with the node, `MajClass` is the majority class of the examples at the node, `NumMajClass` is the number of examples belonging to the majority class, and `Heuristic` is the heuristic value of the node (e.g., the value of the gain ratio for the node).

In addition to decision nodes, a decision tree also contains a number of leaf nodes that provide the classification of new instances. These leaf nodes can be viewed as a special case of decision nodes, where the attribute name and heuristic value are not applicable. This is indicated by providing the special value `NA` at the relevant positions in the tuples describing leaf nodes. The class assigned to new instances reaching a leaf node is, by definition, the majority class of the training examples at that node, i.e., `MajClass`.

Fig.1 shows a sample decision tree. The corresponding Escher term is given in Fig.2. A meta-example is constructed simply by adding a label to a decision tree, i.e.,

```
type MetaExample = (DecisionTree, Label);
```

where, in the case of model selection, `Label` is the name of the learning model that performs best (according to some criteria) on the dataset from which the tree was induced.

Given the above representation, the problem of model selection is automated through meta-learning as follows.

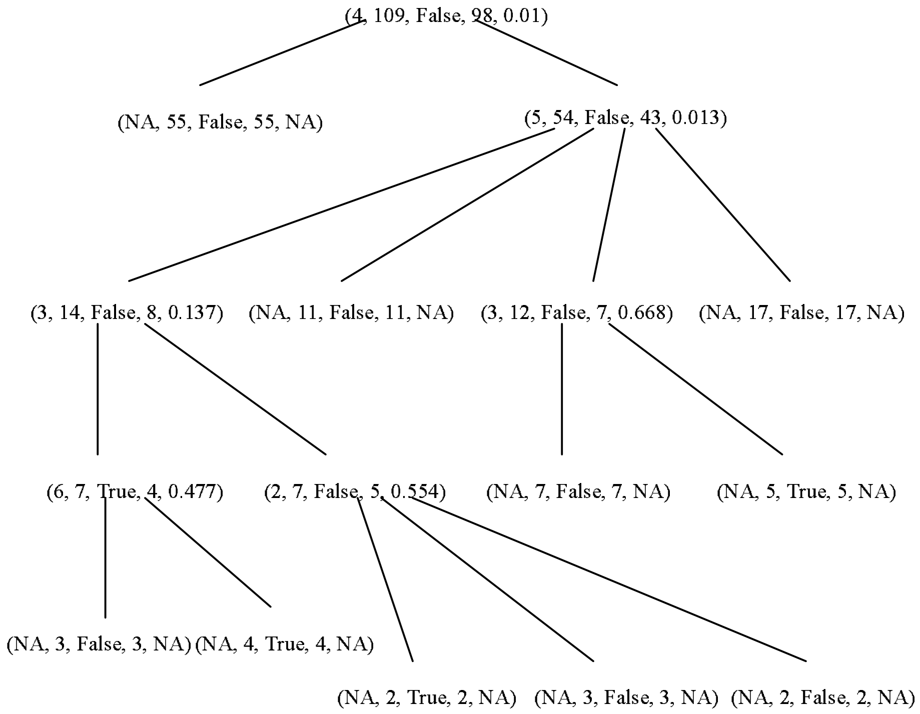

Fig. 1. A sample decision tree

1. Select a (reasonably) large collection, `D`, of datasets.
2. For each `d`∈`D`, induce a decision tree `t`using a standard decision tree learner (e.g., C4.5). This gives rise to a collection, `T`, of decision trees.
3. For each `t`∈`T`, construct the corresponding Escher term and label it to form a meta-example.
4. Using the meta-examples and some higher-order learner, induce a meta-theory.

For example, the following definition (or meta-theory) could be induced from an example set consisting of decision trees, and used to identify which of two learners would be most suitable for a particular learning task.

```
data Learner = BackProp | NaiveBayes;

bestLearner :: Tree DecisionNode -> Learner;
bestLearner t =
  if shape(t) > 0.5 &&
    (length [x | x 'elem' subtrees(t), proj3(root(x)) < 0.003]) > 1
  then BackProp
```

```
(Node (4, 109, False, 98, 0.01) [
    (Node (NA, 55, False, 55, NA []),
    (Node (5, 54, False, 43, 0.013) [
        (Node (3, 14, False, 8, 0.137) [
            (Node (6, 7, True, 4, 0.477) [
                (Node (NA, 3, False, 3, NA) []),
                (Node (NA, 4, True, 4, NA) [])]),
            (Node (2, 7, False, 5, 0.554) [
                (Node (NA, 2, True, 2, NA) []),
                (Node (NA, 3, False, 3, NA) []),
                (Node (NA, 2, False, 2, NA) [])])]),
        (Node (NA, 11, False, 11, NA) []),
        (Node (3, 12, False, 7, 0.668) [
            (Node (NA, 7, False, 7, NA) []),
            (Node (NA, 5, True, 5, NA) [])]),
        (Node (NA, 17, False, 17, NA) [])])]);
```

Fig. 2. Escher term for the decision tree of Fig.1

```
  else NaiveBayes;
```

The English equivalent of the function `bestLearner` is: "If the shape of the decision tree is greater than 0.5 and there are at least two children of the root with heuristic values less than 0.003, then use Backprop, otherwise use NaiveBayes."

This definition makes use of a number of standard functions that can be applied to tuples, trees and lists, namely,

```
root :: Tree a -> a;
subtrees :: Tree a -> [Tree a];
length :: [a] -> Int;
proji :: (a1, ..., ak) -> ai; (for each 1<=i<k=)
```

where `root` returns the root node of a tree, `subtrees` returns the list of all subtrees of a tree, `length` returns the length of a list and `proji` returns the `i`th component of a tuple. These functions and their use in inductive learning are detailed in [6]. The definition also makes use of the special function `shape :: Tree a -> Float` defined in subsection 3.1. Other tree characteristics mentioned in subsection 3.1, such as maximum depth and nodes per instances, may similarly be defined and made available to the meta-learner during induction.

Finally, to predict the most suitable model for a new dataset, $\mathtt{d_{new}}$, one simply needs to induce a decision tree, $\mathtt{t_{new}}$, from it, construct the corresponding Escher term, $\mathtt{e_{new}}$ and apply the meta-theory to $\mathtt{e_{new}}$.

4 Conclusion

We have proposed that meta-learning be applied to tasks represented by induced decision trees. We have briefly described a framework in which such a representation is possible and speculated on the kind of meta-knowledge one might expect from this. Experiments must now be set up to validate the idea. We are currently extending the system described in [10] to carry out experiments.

Our motivation for using the trees directly is that the predefined properties used in decision-tree-based characterisation (see subsection 3.1) only make explicit properties implicit in the tree structure. In addition, these properties have to be calculated *a priori* regardless of their utility in the task characterisation problem. Provided a sufficiently expressive representation language, such as the one we propose, good solutions may be obtained directly from the induced decision trees themselves.

If higher-order meta-learning in this perspective is possible, different data structures can be used to characterise tasks. A natural set of candidates are the structures created by learning algorithms, including not only decision trees as outlined here, but also trained neural networks and rule sets. A task can be characterised by a number of learners' hypotheses, all of which can be manipulated by meta-learners based on our higher-order framework. For decision trees, we use trees; for rule sets, we would use sets and for neural networks, we would use graphs.

In this approach to task characterisation, instead of using learners' performances to describe tasks, as in landmarking (e.g., see [15]), or statistical properties of datasets, as in the traditional approach (e.g., see [14]), we use learners' hypotheses. In effect, we further "blur" the distinction between simple and cheap statistical properties and complete models as produced by learning systems.

The proposed approach can go even further. Indeed, it can easily incorporate other, attribute-based approaches to task characterisation since those meta-attribute vectors could also be expressed in Escher. The gist of the approach, however, is to characterise a task as a consistent hypothesis. Since there are several ways to view a task, we choose to view it with a learner's eyes.

There is however a disadvantage to using as search space a space of learned models (i.e., a hypothesis space). Indeed, it is likely to be more difficult to define notions of "neighbourhoods" in structured spaces (e.g., the space of decision trees) than in vector-like spaces (e.g., a space whose dimensions are (number-valued) statistical measures. Hence, it may be more difficult to generate artificial data in a systematic way as is done in [1] to generalise and increase the validity of case studies. At this stage it is not clear whether this issue will prove a real problem in meta-learning for model selection.

Acknowledgements

This work is supported by an ESPRIT Framework IV LTR Grant (Nr. 26.357).

References

1. D.W. Aha (1992). Generalizing from Case Studies: A Case Study. In *Proceedings of the Ninth International Conference on Machine Learning*, 1-10.
2. H. Bensusan (1999). Automatic Bias Learning; An Inquiry into the Inductive Bias of Induction. PhD Thesis, University of Sussex, School of Cognitive and Computing Sciences.
3. H. Bensusan (1998). God doesn't always shave with Occam's Razor – learning when and how to prune. In *Proceedings of the Tenth European Conference on Machine Learning*, 119-124, Springer.
4. A.F. Bowers, C. Giraud-Carrier, C. Kennedy, J.W. Lloyd and R. MacKinney-Romero (1997). A Framework for Higher-Order Inductive Machine Learning. In *Proceedings of the COMPULOGNet Area Meeting on Representation Issues in Reasoning and Learning*, 19-25.
5. A.F. Bowers, C. Giraud-Carrier and J.W. Lloyd (2000). Classification of Individuals with Complex Structure. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 81-88.
6. A.F. Bowers, C. Giraud-Carrier and J.W. Lloyd (2000). An Unifying View of Knowledge Representation for Inductive Learning. In preparation.
7. R. Engels and C. Theusinger (1998). Using a Data Metric for Offering Preprocessing Advice in Data Mining Applications. In *Proceedings of the Thirteenth European Conference on Artificial Intelligence*, 430-434.
8. P.A. Flach, C. Giraud-Carrier and J.W. Lloyd (1998). Strongly-Typed Inductive Concept Learning. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, 185-194.
9. C. J. Kennedy and C. Giraud-Carrier (1999). An Evolutionary Approach to Concept Learning with Structured Data. In *Proceedings of the Fourth International Conference on Artificial Neural Networks and Genetic Algorithms*, 331-336.
10. C. J. Kennedy (2000). Strongly Typed Evolutionary Programming. PhD Thesis, University of Bristol, Department of Computer Science.
11. J.W. Lloyd (1999). Programming in an Integrated Functional and Logic Language. *Journal of Functional and Logic Programming*, 1999(3).
12. J.W. Lloyd (2000). Knowledge Representation, Computation and Learning in Higher-Order Logic. In preparation.
13. R. MacKinney-Romero and C. Giraud-Carrier (2000). Inducing Classification Rules from Highly-structured Examples Using Composition. Submitted.
14. D. Michie, D.J. Spiegelhalter and C.C. Taylor (Eds.) (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood.
15. B. Pfahringer, H. Bensusan and C. Giraud-Carrier (2000). Meta-learning by Landmarking Various Learning Algorithms. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 743-750.
16. L. Rendell and H.H. Cho (1990). The Effect of Data Character on Empirical Concept Learning. *Machine Learning*, **5**:267-298.