ECML 2007 PKDD
WARSAW POLAND

THE 18TH EUROPEAN CONFERENCE ON MACHINE LEARNING
AND
THE 11TH EUROPEAN CONFERENCE ON PRINCIPLES AND PRACTICE
OF KNOWLEDGE DISCOVERY IN DATABASES

PROCEEDINGS OF THE PLANNING TO LEARN WORKSHOP

PlanLearn-07

September 17, 2007

Warsaw, Poland

Editors:
Pavel Brazdil, University of Porto
Abraham Bernstein, University of Zurich

Typesetting:
Rui Leite, University of Porto

Preface

The task of constructing composite systems, that is systems composed of more than one part, can be seen as interdisciplinary area which builds on expertise in different domains. The aim of this workshop is to explore the possibilities of constructing such systems with the aid of Machine Learning and exploiting the know-how of Data Mining. One way of producing composite systems is by inducing the constituents and then by putting the individual parts together.

For instance, a text extraction system may be composed of various subsystems, some oriented towards tagging, morphosyntactic analysis or word sense disambiguation. This may be followed by selection of informative attributes and finally generation of the system for the extraction of the relevant information. Machine Learning techniques may be employed in various stages of this process.

The problem of constructing complex systems can thus be seen as a problem of planning to resolve multiple (possibly interacting) tasks. So, one important issue that needs to be addressed is how these multiple learning processes can be coordinated. Each task is resolved using certain ordering of operations. Meta-learning can be useful in this process. It can help us to retrieve previous solutions conceived in the past and re-use them in new settings.

Of particular interest are methods and proposals that address the following issues:

- Planning to construct composite systems,
- Exploitation of ontologies of tasks and methods,
- Representation of learning goals and states in learning,
- Control and coordination of learning processes,
- Recovering / adapting sequences of DM operations,
- Meta-learning and exploitation of meta-knowledge,
- Layered learning,
- Multi-task learning,
- Transfer learning,
- Multi-predicate learning (and other relevant ILP methods),
- Combining induction and abduction,
- Multi-strategy learning,
- Learning to learn.

The aim of the workshop is to explore the possibilities of this new area, offer a forum for exchanging ideas and experience concerning the state-of-the art, permit to bring in knowledge gathered in different but related and relevant areas and outline new directions for research.

Warsaw, September 2007

Pavel Brazdil
Abraham Bernstein

Workshop Organization

Workshop Chairs

Pavel Brazdil (University of Porto)
Abraham Bernstein (University of Zurich)

ECML/PKDD Workshop Chair

Marzena Kryszkiewicz (Warsaw University of Technology)

Workshop Program Committee

Abraham Bernstein, Switzerland	Oliver Ray, Great Britain
Pavel Brazdil, Portugal	Ashwin Ram, USA
Christophe Giraud-Carrier, USA	Luc de Raedt, Belgium
Peter Flach, Great Britain	Michele Sebag, France
Larry Hunter, USA	Carlos Soares, Portugal
Rui Leite, Portugal	Maarten van Someren, The Netherlands
Katharina Morik, Germany	Ricardo Vilalta, USA

Table of Contents

20 Years of Planning to Learn

Lawrence Hunter
Center for Computational Pharmacology
University of Colorado School of Medicine
Box 6511, Mail stop 8303, Aurora, CO 80045-9802, USA
Larry.Hunter@uchsc.edu, +1 303 724 3399

Abstract. To my knowledge, the first publication that described the idea of planning to learn was my paper *Knowledge Acquisition Plannin*g in the Third Knowledge Acquisition for Knowledge Based Systems Workshop (Banff, Canada, 1988); it won the "best student paper" award at the workshop. That paper, as well as the following PhD thesis [Hunter, 1989] conference publications [Hunter 1990a, 1990b, Hunter & Ram 1991], journal articles [Hunter 1992, Hunter & Ram 1992] and book chapters [Hunter 1993, Hunter 1994, Hunter & Ram 1995] explored how explicit goals for knowledge could be used to select among and organize the application of a wide variety of machine learning and other computational methods to address particular needs.

That work made what I still believe to be compelling theoretical arguments about the importance of goals in controlling inference, the need for sequences of distinct although inter-related activities for learning (data gathering, data transformations, data selections, model selection, model fitting, model evaluation), the many alternative choices available for each activity, the dependence of the outcome of learning on subtle interactions among those choices, and the value of AI planning methods for selecting (and monitoring and revising) sequences of activities likely to achieve specific goals for knowledge. A theory of the origins of specific goals for knowledge was also described in that body of work, as was the application of these ideas to building systems to aid scientific discovery in molecular biology.

Despite all of those efforts, systems based on this theory with the goal of generating novel scientific insights had not been particularly successful. By 1996, my interest in applying machine learning (and other methods) to advance human understanding of molecular biology had superceded my interest in learning theory. Genomic and other high throughput molecular data was overwhelming biologists with more valuable data than could be assimilated in traditional ways. Computational (and statistical) approaches to extracting knowledge relevant to human health from this tidal wave of data was (and remains!) a pressing concern. While planning to learn remained theoretically attractive, it was very difficult to devise principled ways to prefer any particular sequence of learning actions over any other when the goals for knowledge were so difficult. I developed a more bottom-up approach, called Coevolution Learning [Hunter, 1996; Abramson & Hunter, 1996], which used an evaluation function for alternative representation schemes combined with a more traditional

induction accuracy based evaluation to explore the space of possible combinations of induction and representational schemes. The method was patented [US Patents 6,449,603 and 6,917,926] and remains in use in industry, although to my knowledge no meaningful scientific discoveries arose from its application, either. By now there have been a large number of successful applications of machine learning to molecular biology (e.g. most molecular biologists now know how to apply hidden Markov models to molecular sequence data), and I like to think that I have made some useful contributions along the way, even if the theory has not.

For the last five years or so, my research has focused on the extraction of knowledge from the biomedical literature using natural language processing techniques. Extensive development work has resulted in the creation of the OpenDMAP concept recognition system (see http://opendmap.sourceforge.net), which recently turned in the best performance in a global biomedical information extraction evaluation [Krallinger, et al., 2007]. The OpenDMAP system recognizes concepts (and slot fillers within concepts) through the use of patterns which can include other concepts, text strings, and the output of a wide variety of text analysis tools—ranging from named entity finders to full syntactic parsers [Hunter et al., submitted]. In the first applications [Lu, 2007] these patterns were created by hand. In more recent work [Caporaso, et al., 2007], systematic approaches to corpus analysis were used to produce patterns. Our near-term research goal is to develop increasingly automated approaches to inducing these patterns from lightly annotated training data and modest input from experts. Since the concepts for which patterns need to be produced are taken from a community curated ontology [GO Consortium, 2000] which has been used to annotated tens of thousands of genes and is thereby linked to the literature, it is our belief that planning to learn these patterns may turn out to be more effective than any other approach. It is very gratifying both to find an application area where this nearly 20 year old theory seems likely to succeed in application, and to participate in the workshop where so many others see the value in the approach.

References

Abramson. M. Z. and Hunter, L (1996) Classification using Cultural Coevolution and Genetic Programmin. *Genetic Programming: Proc. of the First Anuual Conf*, pp. 249-254, MIT Press

Caporaso, JG, et al., (2007) Rapid pattern development of concept recognition systems: application to point mutations, *Journal of Bioinformatics and Computational Biology,* forthcoming.

The Gene Ontology Consortium (2000) Gene Ontology: tool for the unification of biology. *Nature Genetics* 25: 25-29

Hunter, L. (1998) Knowledge Acquisition Planning. in *Third Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Alberta, Canada

Hunter, L. (1989) *Knowledge Acquisition Planning: Gaining Expertise Through Experience* Ph.D. Thesis, Yale University Computer Science Department

Hunter, L. (1990a) Planning to Learn, *The Proceedings of The Twelfth Annual Conference of the Cognitive Science Society*, pp. 26-34, Lawrence Erlbaum Associates, Hillsdale, NJ.

Hunter, L. (1990b) Knowledge Acquisition Planning for Inference from Large Datasets*, The Proceedings of The Twenty Third Annual Hawaii International Conference on System Sciences, Kona, HI. vol. 2, Software track*, pp. 35-44. IEEE Press.

Hunter, L. & Ram, A. (1991) The Use of Explicit Goals for Knowledge to Guide Inference and Learning, *Proceedings of the Eighth International Workshop on Machine Learning,* pp. 265-269, Morgan Kaufmann, San Mateo, CA.

Hunter, L. (1992) Knowledge Acquisition Planning: Using Multiple Sources of Knowledge to Answer Questions in Biomedicine, *Mathematical and Computer Modeling,* 16(6/7):79-91.

Hunter, L**.** & Ram, A. (1992) Goals for Learning and Understanding. *Journal of Applied Intelligence.* 2(1):47-73.

Hunter, L. (1993) Planning to Learn About Protein Structure, in *Artificial Intelligence and Molecular Biology,* L. Hunter, ed., AAAI Press, 1993.

Hunter, L. (1994) Classifying for Prediction: A Multistrategy Approach to Predicting Protein Structure, in *Machine Learning IV*, ed. by R. Michalski & G. Tegucci, Morgan Kaufmann.

Hunter & Ram (1995) Planning to Learn, in *Goal-Driven Learning,* ed. by Ashwin Ram and David B. Leake, MIT Press.

Hunter, L**.** (1996) Coevolution Learning: Synergistic Evolution of Learning Agents and Problem Representations, *Proceedings of 1996 Multistrategy Learning Conference*, pp. 85-94, Menlo Park, CA: AAAI Press.

Hunter, et al., (submitted) OpenDMAP: An open-source, ontology-driven concept analysis engine, with applications to capturing knowledge regarding protein transport, protein interactions and cell-type-specific gene expression. *BMC Bioinformatics*, submitted.

Krallinger, M., et al., (2007) *Proceedings of the Second BioCreAtIvE Challenge Workshop: Critical Assessment of Information Extraction in Molecular Biology*. See also a special issue of *Genome Biology* devoted to this evaluation, forthcoming in 2007, and http://biocreative.sourceforge.net/biocreative_2.html

Lu, Z., (2007) *Text Mining on GeneRIFs.* PhD thesis, Computional Bioscience Program, University of Colorado School of Medicine, CO, USA.

Towards Intelligent Assistance for a Data Mining Process

Abraham Bernstein

University of Zurich, Department of Informatics,
8050 Zürich, Switzerland
bernstein@ifi.uzh.ch

Joint work with
Foster Provost, New York Univ., Stern School of Business, New York NY, USA,
Shawndra Hill, The Wharton School, Univ. of Pennsylvania, Philadelphia PA, USA
and Michael Dänzer Univ. of Zürich, Dep. of Informatics, 8050 Zürich, Switzerland

Abstract. A data mining (DM) process involves multiple stages. A simple, but typical, process might include preprocessing data, applying a data-mining algorithm, and post-processing the mining results. There are many possible choices for each stage, and only some combinations are valid. Because of the large space and non-trivial interactions, both novices and data-mining specialists need assistance in composing and selecting DM processes.

Extending notions developed for statistical expert systems we present a prototype Intelligent Discovery Assistant (IDA), which provides users with (i) systematic enumerations of valid DM processes, in order that important, potentially fruitful options are not overlooked, and (ii) effective rankings of these valid processes by different criteria, to facilitate the choice of DM processes to execute. We use the prototype to show that an IDA can indeed provide useful enumerations and effective rankings in the context of simple classification processes. We discuss how an IDA could be an important tool for knowledge sharing among a team of data miners. Furthermore, we illustrate the claims with a comprehensive demonstration of cost-sensitive classification using a more involved process and data from the 1998 KDDCUP competition.

Finally, we discuss, how new technologies arising in the Semantic Web domain might help to build IDAs more efficiently. Specifically, we briefly discuss how semantic data and data mining operator descriptions can help to leverage off-the-shelf semantic web service functionality to build easy to use IDAs.

References

1. Bernstein, Abraham and Provost, Foster and Hill, Shawndra. Towards Intelligent Assistance for a Data Mining Process: An Ontology-based Approach for Cost-sensitive Classification. IEEE Transactions on Knowledge and Data Engineering, vol. 17, n. 4, pag. 503-518, 2005.
2. Bernstein, Abraham and Daenzer, Michael. The NExT System: Towards True Dynamic Adaptions of Semantic Web Service Compositions (System Description). In *Proceedings of the 4th European Semantic Web Conference (ESWC '07)*. Springer 2007.

Designing Complex Systems: Role of Learning and Domain Specific Meta-knowledge

Pavel Brazdil[1] [2]

[1] LIAAD - INESC Porto L.A., University of Porto, Rua de Ceuta, 118-6, 4050-190, Porto, Portugal; pbrazdil@liacc.up.pt
[2] Faculty of Economics, University of Porto, Porto, Portugal

Abstract. Our aim is to discuss the problem of employing learning methods in the design of complex systems. The term *complex systems* is used here to identify systems that cannot be learned in one step, but rather require several phases of learning. Our aim will be to show how domain specific meta-knowledge can be used to facilitate this task.

As we will see, *dynamic selection of bias* plays an important role here. But let us review first what is meant by *bias*. According to DesJardins and Gordon [2], bias is *any factor that influences the definition or selection of inductive hypotheses.*

So if meta-knowledge acquired in the course of dealing with a set of problems is used to pre-select a subset of learning algorithms (e.g. as in [1]), we can consider this as *dynamic selection of bias*. By eliminating some Machine Learning (ML) / Data Mining (DM) algorithms we are, in effect, excluding some forms of inductive hypotheses from consideration.

Let us now consider another possible interpretation of bias when applying ML/DM algorithms. Without loss of generality, let us simply focus just on one ML algorithm to simplify the exposition. Let us further assume that the aim is to predict a categorical (or a numeric) value of some variable, but the rest of the data includes potentially a *very large number of attributes*. So a question arises what should be done in this case.

A typical solution adopted is to gather the data first and then use some standard feature elimination methods to reduce the number of features as appropriate (e.g. [7]). However, this approach has the following shortcoming. Someone has to decide which attributes / features are potentially relevant for the task at hand. If a wrong decision is made, this can have a negative impact on the outcome of learning. If the relevant attributes are not included, a sub-optimal hypothesis may be generated. If on the other hand the set of attributes is too large and includes unnecessary information, it may again be difficult for the system to generate the right hypothesis (the search space of inductive hypotheses may be too large). So, obviously it is advantageous to have methods that help us to determine this automatically.

Determining which attributes should be used can be considered as the problem of dynamic selection of bias, as it satisfies the definition given earlier. So, our aim in this talk is to discuss this issue in more detail, clarify the relationship to meta-learning and suggest how the issue of dynamic selection of bias could be handled.

This problem has been noted by innumerous people in AI and ML in the past. Various researhers (Hunter and Ram [4], [5], Michalski [6]) have argued that it is important to define explicit goals that guide learning. Learning is seen as search through a knowledge space guided by the learning goal. Learning goals determine which parts of prior knowledge are relevant, what knowledge is to be acquired and in which form, how it is to be evaluated and when to stop learning. The importance of planning in this process was also identified ([3]).

In our view the issue of dynamic selection of bias is important in the construction of complex systems. In these tasks we need not only to identify the attributes / features that are potentially relevant, but also identify one or more subproblems (concepts) that constitute the final solution. Typically, it is advantageous to define also some ordering in which (some of) the subproblems (concepts) should be acquired. This problem can be seen as the problem of learning multiple interdependent concepts. In effect, defining the ordering can be regarded as defining the appropriate procedural bias, as it determines how the hypotheses space should be searched. The aim of this talk is to present more details on how these kind of problems could be handled, concentrating mainly on case studies. More details concerning this issue can be found in [1].

References

1. P. Brazdil, C. Giraud-Carrier, R. Vilalta, and C. Soares. *Meta-Learning.* Springer, 2007 (to appear).
2. M. DesJardins and D. Gordon. Evaluation and selection of biases. *Machine Learning*, 20:5–22, 1995.
3. L. Hunter. Planning to learn. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pages 26–34, Hillsdale, NJ., 1990. Lawrence Erlbaum Associates.
4. L. Hunter and A. Ram. The use of explicit goals for knowledge to guide inference and learning. In B. L and C. G, editors, *Proceedings of the Eighth International Workshop on Machine Learning (ML'91)*, pages 265–269, San Francisco, CA, USA, 1991. Morgan Kaufmann.
5. L. Hunter and A. Ram. Goals for learning and understanding. *Applied Intelligence*, 2(1):47–73, July 1992.
6. R. Michalski. *Inferential Theory of Learning: Developing Foundations for Multistrategy Learning*, chapter 1, pages 3–62. Morgan Kaufmann, February 1994.
7. S. M. Weiss and N. Indurkhya. *Predictive Data Mining: A Practical Guide.* Morgan Kaufmann, San Francisco, CA, USA, August 1997.

Meta-Learning Rule Learning Heuristics

Frederik Janssen and Johannes Fürnkranz

TU Darmstadt, Knowledge Engineering Group
Hochschulstraße 10, D-64289 Darmstadt, Germany
[janssen,juffi]@ke.informatik.tu-darmstadt.de

Abstract. The goal of this paper is to investigate to what extent a rule learning heuristic can be learned from experience. Our basic approach is to learn a large number of rules and record their performance on the test set. Subsequently, we train regression algorithms on predicting the test set performance from training set characteristics. We investigate several variations of this basic scenario, including the question whether it is better to predict the performance of the candidate rule itself or of the resulting final rule. Our experiments on a number of independent evaluation sets show that the learned heuristics outperform standard rule learning heuristics. We also analyze their behavior in coverage space.

1 Introduction

It is well-known that learning a classification rule is essentially a search problem [10, 4], where the states are rules and the successor function is a refinement operator that returns all minimal specializations of a rule. The goal is to find a rule that maximizes the predictive performance in a domain. As this performance cannot be directly measured, an evaluation function is used to estimate the quality of a rule. Typically, the same evaluation function is also used as a *search heuristic* that allows a greedy search algorithm to focus on interesting parts of the hypothesis space. The long-term goal of our research is to understand the properties of such search heuristics. The underlying (implicit) assumption is that a rule with high quality will also produce good refinements. There has not been much work on trying to characterize the behavior of good heuristics. Notable exceptions include [9], which proposed *weighted relative accuracy* (WRA) as a novel heuristic, and [6], in which a wide variety of rule evaluation metrics were analyzed and compared by visualizing their behavior in ROC space.

In previous work [7], we adjusted parameters of three heuristics, whose shape was predetermined. The key idea of this work is to meta-learn such a heuristic from experience, without a bias towards existing measures. To this end, we create a large meta data set which we use to learn a function that predicts the performance of a rule on an independent test set. In order to address the issue that a good rule evaluation function does not necessarily coincide with a good search heuristic for finding a rule that optimizes the evaluation function, we also analyze a setting in which the learner attempts to predict the performance of a *complete* rule from its incomplete predecessors.

The rule learner, which is used for generating the meta data and for evaluating the learned heuristics, is described in Section 2, after which we continue with a brief discussion of rule learning heuristics (Section 3). The meta data generation and the experimental setup is described in Section 4. The main results are presented in Section 5.

2 Rule Learning Algorithm

For the purpose of this empirical study, we implemented a simple *Separate-and-conquer* or *Covering* rule learning algorithm [4] within the *Weka* machine learning environment [16]. Both the outer loop (the covering procedure) and the top-down refinement inside the learner are fairly standard. For details about the implementation, see [3, 4].

Separate-and-conquer rule learning can be divided into two main steps: First, a rule is learned from the training data by a greedy search (the *conquer* step). Second, all examples covered by the learned rule are removed from the data set (the *separate* step). Then, the next rule is learned on the remaining examples. Both steps are repeated as long as positive examples are left in the training set. The refinement procedure, which is used inside the conquer step of the algorithm, returns all possible candidate refinements that can be obtained by adding a single condition to the body of the rule. For nominal attributes, conditions test for equality with a domain value, for numerical attributes they use $>$ and $\leq$. The best among all refinements is selected.

Our implementation continues to greedily refine the current rule until no more negative examples are covered. In this case, the search stops and the best rule encountered during the refinement process is added to the theory (which is initialized as the empty theory). Note that this is not necessarily the last rule searched. We use random tie breaking for rules with equal evaluation, and filter out candidate rules that do not cover any positive examples. Rules are added to the theory as long as this increases the accuracy of the theory on the training set (this is the case when the best rule found covers more positive than negative examples).

We do not use any specific pruning technique, but solely rely on the evaluation of the rules by the used rule learning heuristic. Note, however, that this does not mean that we learn an overfitting theory that is complete and consistent on the training data (i.e., a theory that covers all positive and no negative examples), because many heuristics will prefer impure rules with a high coverage over pure rules with a lower coverage.

Multi-class problems are tackled by sorting the classes according to their frequency (least frequent first), and training binary classifiers that discriminate a class from all subsequent classes. The classifiers are then used in this order, which essentially means that a decision list is formed, in which the rules for each class appear in blocks of increasing class frequencies.

Table 1. Rule learning heuristics used in this paper ($\sim$ denotes order equivalency)

$\text{precision} = \frac{p}{p+n} \sim \frac{p-n}{p+n}$	$\text{accuracy} = \frac{p+(N-n)}{P+N} \sim p-n$
$\text{Laplace} = \frac{p+1}{p+n+2}$	$\text{WRA} = \frac{p+n}{P+N}\left(\frac{p}{p+n} - \frac{P}{P+N}\right) \sim \frac{p}{P} - \frac{n}{N}$
$\text{correlation} = \frac{p(N-n)-(P-p)n}{\sqrt{PN(p+n)(P-p+N-n)}}$	

3 Rule Learning Heuristics

Numerous heuristics have been provided for inductive rule learning, a general survey can be found in [4]. Most rule learning heuristics can be seen as functions of the following four arguments:

- P and N: the number of positive/negative examples in the training set
- p and n: the number of positive/negative examples covered by the rule

Examples of heuristics of this type are the commonly used heuristics that are shown in Table 1. *Precision* is known to overfit the data, WRA [14] has a tendency to over-generalize. In [6] it was shown that the m-estimate, which is defined by $\frac{p+m\cdot\frac{P}{P+N}}{p+n+m}$, forms a trade-off between these two extremes.

As P and N are constant for a given learning problem, these heuristics effectively only differ in the way they trade off completeness (maximizing p) and consistency (minimizing n), and may thus be viewed as a function $h(p,n)$. As a consequence, each rule can be considered as a point in coverage space, a variant of ROC space that uses the absolute numbers of true positives and false positives as its axes. The preference bias of different heuristics may then be visualized by plotting the respective heuristic values of the rules on top their locations in coverage space, resulting in a 3-dimensional plot $(p, n, h(p, n))$. A good way to view this graph in two dimensions is to plot the *isometrics* of the learning heuristics, i.e., to show contour lines that connect rules with identical heuristic evaluation values [6]. Another method is to plot both contour lines and the surface of the function which is done in our visualization (cf. Section 5.4).

The goal of our work is to automatically learn a function $\hat{h}(p,n)$, which allows to predict the quality of a learned rule. However, note that most of the functions in Table 1 contain some non-linear dependencies between these values. In order to make the task for the learner easier, we will not only characterize a rule by the values p, n, P, and N, but in addition also use the following parameters as input for the meta-learning phase:

- $tpr = \frac{p}{P}$, the true positive rate of the rule
- $fpr = \frac{n}{N}$, the false positive rate of the rule
- $Prior = \frac{P}{P+N}$, the a priori distribution of positive and negative examples
- $prec = \frac{p}{p+n}$, precision, the fraction of positive examples covered by the rule

Thus, we characterize a rule r by an 8-tuple $<P, N, Prior, p, n, tpr, fpr, prec>$.

Some heuristics use additional components, such as the length of the rule, or the number of positive and negative examples that are covered by the rule's predecessor. We have also performed experiments that include the length or the rule (which can be found in the long version of this paper [8]), but this did not have a noticable effect on the performance. We think that the main goal of using the length of a rule is to indirectly capture the degree of generality of a rule (shorter rules cover more examples), which can be directly measured with p and n.

We will not consider statistics about a rule's predecessor, as our goal is to find a function that allows to evaluate a rule, irrespective of how it has been learned. Including them may result in different evaluations for the same rule, depending on the order in which its conditions have been added to the rule body. An example of such a heuristic is FOIL's information gain. We will also not include it in our performance measures because it actually measures the quality of refinements and not the quality of rules, which means that it cannot be used to select an optimal rule without the use of additional stopping criteria.

4 Meta-Learning Scenario

4.1 Definition of the Meta-Learning Task

We frame the rule learning process as a search problem in the following way: Each (incomplete) rule is a state, and all possible refinements (e.g., all possible conditions that can be added to the rule) are the actions. The rule-learning agent repeatedly has to pick one of the possible refinements according to their expected utility until it has completed the learning of a rule.

In this framework, the problem of meta-learning a rule learning heuristic may be considered as a reinforcement learning problem: After learning a complete theory, the learner receives a reinforcement signal (e.g., the estimated accuracy of the learned theory), which can then be used to adjust the utility function. After a (presumably large) number of learning episodes, the utility function should converge to a heuristic that evaluates a candidate rule with the quality of the *best* rule that can be obtained by refining the candidate rule. However, for practical purposes this scenario appears to be too complex. In [2] a reinforcement learning algorithm was applied on this problem, but with disappointing results.

For this reason, we redefine the problem as a supervised learning task: Each rule is evaluated on a separate test set, in order to get an estimate of its true performance. This information is then used as the target value for rules that are characterized with the eight features discussed in Section 3. We studied both, immediate reward (where rules are trained on their own test set performance) and delayed reward (where rules are trained on the performance of their best refinement; cf. Section 5.3).

4.2 Meta Data Generation

As explained above, we try to model the relation of the rule's statistics measured on the training set and its "true" performance, which is estimated on an independent test set. Therefore, we used the rule learner described above for obtaining the above-mentioned characteristics for each learned rule. These form a training instance in the meta data set. The training signals are the performance parameters of the rule on the test set.

As we want to guide the entire rule learning process, we need to record this information not only for final rules — those that would be used in the final theory — but also for all their predecessors. Therefore all candidate rules which are created during the refinement process are included in the meta data as well. The Algorithm for creating the meta data is described in detail in [8].

It should be noted, that we ignore all rules that do not cover any instance on the test data, because, on the one hand, we do not have training information for such rules (the test precision that we try to model is undefined), and, on the other hand, such rules will not do any harm (they won't have an impact on test set accuracy as they do not classify any example).

To ensure that we obtain a set of rules with varying characteristics, the following parameters were modified:

Datasets: We used 27 datasets with varying characteristics (different number of classes, attributes, instances) from the UCI Repository [12] (for a list see [8]).

5x2 Cross-validation: For each dataset, we performed 5 iterations of a 2-fold cross-validation. 2-fold cross-validation was chosen because in this case the training and test sets have equal size, so that we don't have to account for statistical variance in the precision or coverage estimates. We performed five iterations with different random seeds. Note that our primary interest was to obtain a lot of rules which characterize the connection between training set statistics and the test set precision. Therefore, we collected statistics for all rules of all folds.

Classes: For each dataset and each fold, we generated one dataset for each class, treating this class as the positive one and the union of all the others as the negative class. Rules were learned for each of the resulting two-class datasets.

Heuristics: We ran the rule learner several times on the binary datasets, each time using a different search heuristic (displayed in Table 1). The first four form a representative selection of search heuristics with linear ROC space isometrics [5], while the correlation heuristic has non-linear isometrics. These heuristics represent a large variety of learning biases.

In total, our meta dataset contains $87,380$ examples.

4.3 Regression Methods

We use two standard regression methods for learning functions on the meta data, in their default parametrizations in *Weka* [16]. First, we apply a simple *linear regression* based on the Akaike criterion [1] for model selection. A key advantage of this method is that we obtain a simple, comprehensible form of the learned heuristic function. Note that the learned function is nevertheless non-linear in the basic dimensions p and n because of the non-linear terms that are used as basic features (e.g., $p/(p+n)$). In order to be able to address a wider class of functions, we also employ *multilayer perceptrons* with back-propagation training and sigmoid nodes. We apply various sizes of the hidden layer (1, 5, and 10), and train for one epoch (i.e., we go through the training data once). We have also tried to train the networks with a larger number of epochs, but the results do not further improve.

4.4 Evaluation methods

A straight-forward approach to measure the fit of the learned function to the target values is to estimate its mean absolute error by a 10-fold cross validation on the meta data set.

$$MAE(f, \hat{f}) = \frac{1}{m} \sum_{i=0}^{m} |\hat{f}(i) - f(i)|$$

where m denotes the number of instances, $f(i)$ the actual value, and $\hat{f}(i)$ the predicted value of instance i.

Note, however, that a low prediction error on the meta data set does not necessarily imply that the function works good as heuristic (cf. Table 2). Thus, our primary method for evaluating the learned heuristics is to use these heuristics inside the rule learner. To this end, we evaluate the rule learner on 30 UCI data sets, which have not been used during the training phase (for a list see [8]). Like the 27 data sets on which the rules for the meta data are induced, these 30 sets have varying characteristics to ensure that our method will perform well under a wide variety of conditions. On each dataset, the rule learner with the learned heuristics was evaluated with one iteration of a 10-fold cross validation. The performance over all sets was then averaged. We also evaluated the length of the learned theories in terms of number of conditions.

5 Results

5.1 Predicting Test-Set Precision

We are first interested in how accurately the out-of-sample precision of a rule can be predicted. We train a linear regression model and a neural network on the eight measurements that we use for characterizing a rule (cf. Section 3) using the test set precision as the target function. Table 2 displays results for the Linear Regression and three different neural networks, with different numbers of nodes in the hidden layer. The performances of the four algorithms are quite comparable, with the possible exception of the neural network with 5 nodes in

Table 2. Accuracies and theory complexities for several methods

method	MAE	Accuracy	# conditions
LinearRegression	0.22	77.43%	117.6
MLP (1 node)	0.28	77.81%	121.3
MLP (5 nodes)	0.27	77.37%	1085.8
MLP (10 nodes)	0.27	77.53%	112.7

Table 3. Coefficients of the Linear Regression

P	N	$\frac{P}{P+N}$	p	n	$\frac{p}{P}$	$\frac{n}{N}$	$\frac{p}{p+n}$	constant
0.0001	0.0001	0.7485	-0.0001	-0.0009	0.165	0.0	0.3863	0.0267

the hidden layer, which induced very large theories (over 1000 conditions on average), and also had a somewhat worse performance in predictive accuracy.

Table 3 shows the coefficients of the learned regression model. The most important feature was the *a priori* distribution of the examples in the training data followed by the precision of the rule. Interestingly, while the *tpr* has a non-negligible influence on the result, the *fpr* is practically ignored. Both the current coverage of a rule (p and n) and the total example counts of the data (P and N) have comparably low weights. This is not that surprising if one keeps in mind that the target value is in the range $[0, 1]$, while the absolute values for p and n are in a much higher range. We nevertheless had included them because we believe that in particular for rules with low coverage, the absolute numbers are more important than their relative fractions. A rule that covers only a single example will typically be bad, irrespective of the size of the original dataset.

In order to see whether we can completely ignore the absolute values, we learned another function which only used $\frac{P}{P+N}$, p/P, n/N and $\frac{p}{p+n}$ as input values. The linear regression function trained on this dataset performed insignificantly worse than the one that is computed on the original set (77.43% accuracy vs. 77.20% accuracy). For the neural networks, the performance degradation was somewhat worse.

5.2 Predicting Positive and Negative Coverage

So far we focused on directly predicting the out-of-sample precision of a rule, assuming that this would be a good heuristic for learning a rule set (cf. Section 3). However, this choice was somewhat arbitrary. Ideally, we would like to repeat this experiment with out-of-sample values for all common rule learning heuristics. In order to cut down the number of needed experiments, we decided to directly predict the number of covered positive ($\hat{p}$) and negative ($\hat{n}$) examples. We then can combine the predictions for these values with any standard heuristic h by computing $h(\hat{p}, \hat{n})$ instead of the conventional $h(p, n)$. Note that the heuristic h only gets the predicted coverages ($\hat{p}$ and $\hat{n}$) as new input, all other statistics (e.g., P,N) are still measured on the training set. This is feasible because we designed the experiments so that the training and test set are of equal size, i.e., the values predicted for $\hat{p}$ and $\hat{n}$ are predictions for the number of covered examples on an independent test set of the same size as the training set.

Table 4 compares the performance of various heuristics with measured and predicted coverage values on the 30 test sets. In general, the results are disappointing. For three of the five heuristics, no significant change could be observed, but for WRA and the *Correlation* heuristic, the performance degrades substantially.

Table 4. Accuracy and theory complexity comparison of various heuristics with training-set (p, n) and predicted $(\hat{p}, \hat{n})$ coverages (number of conditions in brackets)

args	Accuracy	Precision	WRA	Laplace	Correlation
(p, n)	75.6% (104.77)	76.22% (129.17)	75.8% (12.13)	76.89% (118.83)	77.57% (47.5)
$(\hat{p}, \hat{n})$	75.39% (110.8)	76.53% (30)	69.89% (29.97)	76.8% (246.8)	58.09% (40.4)

A rather surprising observation is the complexity of the learned theories. For instance, the heuristic *Precision* produces very simple theories when it is used with the out-of-sample predictions, and, by doing so, increases the predictive accuracy. Apparently, the use of the predicted values of $\hat{p}$ and $\hat{n}$ allows to prevent overfitting, because the predicted positive/negative coverages are never exactly 0 and therefore the overfitting problem observed with *Precision* does not occur any more.

In summary, it seems that the predictions of both the linear regression and the neural network are not good enough to yield true coverage values on the test set. A closer look at the predicted values reveals that on the one hand both regression methods predict negative coverages and that on the other hand for the region of low coverages (which is the important one) too optimistic values are predicted (for both the positive and the negative coverage). The acceptable performance is caused by a balancing of the two imprecise predictions (as observed with the two precision-like metrics) or rather by an induced bias which tries to omit the extreme values in the evaluations (which are responsible for overfitting).

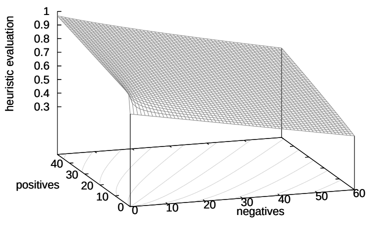
5.3 Predicting the Value of the Final Rule

Rule learning heuristics typically evaluate the quality of the current, incomplete rule, and use this measure for greedily selecting the best candidate for further refinement. However, as discussed in Section 4.1, if we frame the learning problem as a search problem, a good heuristic should not evaluate a candidate rule with its discriminatory power, but with its potential to be refined into a good final rule. Such a utility function could be learned with a reinforcement learning algorithm as described in Section 4.1.

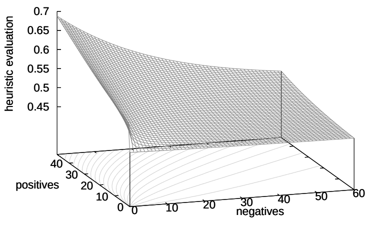
As an alternative, we apply a supervised method which may be viewed as an "offline" version of reinforcement learning. We associate each candidate rule with the precision value of its best refinement encountered during the search. As a consequence, in our approach all candidate rules of one refinement process have the same target value, namely the value of the rule that has eventually been selected. Because of the deletion of all final rules that do not cover any example on the test set, we decided to remove all predecessors of such rules as well. Thus, the new meta data set contains only 77,240 examples in total.

The neural network performs best with an accuracy of 78.37% followed by the Linear Regression which achieves 77.95% on the 30 test data sets. The neural network also learns simpler theories than the Linear Regression (53.97 vs. 95.63 conditions).

Both induced heuristics outperform all of the standard heuristics (cf. Table 4). The Linear Regression was trained on the meta data set that only contains



(a) Linear Regression



(b) Neural Network

Fig. 1. Isometrics of the functions (final rule precision)

the 4 most important features which yield the best model. In terms of theory complexity it seems that about 50 conditions in average are necessary to obtain an accurate classifier. WRA, for example, learns simpler theories (as observed in [14]), but seems to over-generalize. The neural network classifier performs best with the third-smallest theory.

5.4 Isometrics of the Heuristics

To understand the behavior of the learned heuristics, we follow the framework introduced in [6] and analyze their isometrics in ROC or coverage space. Figure 1 shows a 3d-plot of the surface of the learned heuristic in a coverage space with 60x48 examples (the sizes were chosen arbitrarily). The bottom of the graph, shows isometric lines that characterize this surface. The left part of the figure displays the isometrics of the heuristic that was learned by linear regression on the data set that used only the relevant features (see Section 5.1). The right part shows the best-performing neural network (the one that uses only one node in the hidden layer).

Apparently, both functions learn somewhat different heuristics. Although the 3d-surfaces looks fairly similar to each other (except for the non-linear behavior of the neural net and the stronger degradation when covering more negative examples as the linear regression), the isometric lines reveal that the learned heuristics are, in fact, quite different. Those for the linear regression are like a variant of weighted relative accuracy, but with a different cost model (i.e. false negatives are more costly than false positives). The isometrics for the neural net seems to employ a trade-off similar to those of the F-measure. The shift towards the N-axis is reminiscent of the F-measure (for an illustration see [7]), which tries to correct the undesirable property of precision that all rules that cover no negative examples are evaluated equally, irrespective of the number of positive examples that they cover.

However, both heuristics have a non-linear shape of the isometrics in common, which bends the lines towards the N-axis. Effectively, this encodes a bias towards rules that cover a low number of positive examples (compared to regular precision). This seems to be a desirable property for a heuristic that is used in a covering algorithm, where incompleteness (not covering all positive examples) is less severe than inconsistency (covering some negative examples), because incompleteness can be corrected by subsequent rules, whereas inconsistency cannot.

6 Conclusion

The most important result of this work is that we have shown that a rule learning heuristic can be learned that outperforms standard heuristics in terms of predictive accuracy on a collection of databases that were not used in the meta-learning phase. Our first results, which used a few obvious features to predict the out-of-sample precision of the current rule, were already *en par* with the correlation heuristic, which performed best in our experiments (cf. Table 2 and Table 4). Subsequently, we tried to modify several parameters of this basic setup with mixed results. In particular, predicting the positive and negative coverage of a rule on a test set, and using these predicted coverage values inside the heuristics did not prove to be successful. Also, more complex neural network architectures did not seem to be important, linear regression and neural networks with a single node in the hidden layer performed best. On the other hand, a key result of this work is that evaluating a candidate rule by its potential of being refined into a good final rule works better than evaluating the quality of the candidate rule itself. This indicates that a clear separation of rule evaluation metrics (which characterize which rules we should look for) and search heuristics (which guide the process of finding such rules) is important. We intend to further investigate this issue.

A visualization of the learned heuristics in coverage space gave some insight into the general functionalities of the learned heuristics. In comparison to heuristics with linear isometrics, the learned heuristics have non-linear isometrics that implement a particularly strong bias towards rules with a low coverage on negative examples. This makes sense for heuristics that will be used in a covering loop, because incompleteness can be compensated by subsequent rules, whereas inconsistency cannot. Correlation, the standard heuristic that performed best in our experiments, implements a similar bias [6]. Thus, the results of this paper also contribute to our understanding of the desirable behavior of rule-learning heuristics.

Our results may also be viewed in the context of trying to correct overly optimistic training error estimates (resubstitution estimates). In particular, in some of our experiments, we try to directly predict the out-of-sample precision of a rule. This problem has been studied theoretically in [13, 11]. In other works, it has been addressed empirically. For example, empirical data was used in [15] to measure the VC-Dimension of learning machines. In [3] meta data was also created in a quite similar way, and the authors have tried to fit various functions to the data. But the focus there is the analysis of the obtained predictions for out-of-sample precision, which is not the key issue in our experiments.

Acknowledgements
This research was supported by the *German Science Foundation (DFG)* under grant FU 580/2-1.

References

[1] H. Akaike. A new look at the statistical model selection. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[2] S. Burges. Meta-Lernen einer Evaluierungs-Funktion für einen Regel-Lerner, December 2006. Master's Thesis.

[3] J. Fürnkranz. Modeling rule precision. In *Lernen – Wissensentdeckung — Adaptivität. Proceedings of the LWA-04 Workshops*, pp. 147–154, Humboldt-Universität zu Berlin, 2004.

[4] J. Fürnkranz. Separate-and-Conquer Rule Learning. *Artificial Intelligence Review*, 13(1):3–54, February 1999.

[5] J. Fürnkranz and P. A. Flach. An Analysis of Rule Evaluation Metrics. In *Proceedings 20th International Conference on Machine Learning (ICML'03)*, pp. 202–209. AAAI Press, January 2003. ISBN 1-57735-189-4.

[6] J. Fürnkranz and P. A. Flach. ROC 'n' Rule Learning - Towards a Better Understanding of Covering Algorithms. *Machine Learning*, 58(1):39–77, January 2005.

[7] F. Janssen and J. Fürnkranz. On trading off consistency and coverage in inductive rule learning. In K.-D. Althoff and M. Schaaf, editors, *Proceedings of the LWA 2006, Lernen Wissensentdeckung Adaptivität*, pp. 306–313, Hildesheim, Germany, 2006. URL

[8] F. Janssen and J. Fürnkranz. Meta-learning rule learning heuristics. Technical Report. Knowledge Engineering Group, TU Darmstadt. TUD-KE-2007-2, 2007.

[9] N. Lavrač, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pp. 174–185, 1999.

[10] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, 1982.

[11] M. Mozina, J. Demsar, J. Zabkar, and I. Bratko. Why is rule learning optimistic and how to correct it. In *Machine Learning: ECML 2006, 17th European Conference on Machine Learning*, pp. 330–340, 2006.

[12] D. Newman, C. Blake, S. Hettich, and C. Merz. UCI Repository of Machine Learning databases, 1998.

[13] T. Scheffer. Finding association rules that trade support optimally against confidence. In *Principles of Data Mining and Knowledge Discovery, 5th European Conference, PKDD 2001*, pp. 424–435, 2001.

[14] L. Todorovski, P. Flach, and N. Lavrac. Predictive performance of weighted relative accuracy. In *4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000)*, pp. 255–264, September 2000.

[15] V. Vapnik, E. Levin, and Y. L. Cun. Measuring the VC-dimension of a learning machine. *Neural Computation*, 6(5):851–876, 1994.

[16] I. H. Witten and E. Frank. *Data Mining — Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann Publishers, 2nd edition, 2005.

Evolutionary Learning with Cross-Class Knowledge Reuse for Handwritten Character Recognition

Wojciech Jaśkowski, Krzysztof Krawiec, and Bartosz Wieloch

Institute of Computing Science, Poznan University of Technology
Piotrowo 2, 60965 Poznań, Poland
{wjaskowski|kkrawiec|bwieloch}@cs.put.poznan.pl

Abstract. We propose a learning algorithm that reuses knowledge acquired in past learning sessions to improve its performance on a new learning task. The method concerns visual learning and uses genetic programming to represent hypotheses, each of them being a procedure that processes visual primitives derived from the training images. The process of recognition is generative, i.e., a procedure is supposed to restore the shape of the processed object by drawing its reproduction on a separate canvas. This basic method is extended with a knowledge reuse mechanism that allows learners to import genetic material from hypotheses that evolved for the other decision classes (object classes). We compare both methods on a task of handwritten character recognition, and conclude that knowledge reuse leads to significant improvement of classification accuracy and reduces the risk of overfitting.

1 Introduction

Most of contemporary machine learning (ML) algorithms are designed to process *isolated* learning tasks. Usually, a learning algorithm (*inducer*) produces a *classifier* based exclusively on the training data provided for particular learning task. In that process, the inducer relies on fixed inductive bias (priors) that does not change from task to task. There is no way of reusing the knowledge that the inducer could have acquired when inducing classifiers in the past.

This limitation is conspicuously inconsistent with the human way of learning, which is always based on individual's past experience. Priors in human learning come from one's history of dealing with similar tasks. More than that, acquiring new skills in isolation from past experience is impossible for humans. By reusing knowledge, humans can successfully learn in demanding conditions, e.g., when the number of training examples is small or in presence of data inconsistency.

The ability to reuse knowledge would be definitely a virtue for a machine learning system, speeding up the convergence of the learning process, reducing the risk of overfitting, and keeping down the number of training examples required to learn the concept. Some of these benefits have been already demonstrated in related studies on, e.g., multitask learning [1]. However, the last decade did not bring breakthrough in this topic, and knowledge reuse is still listed among the most challenging issues in ML [14].

Incapability of ML systems to reuse knowledge is due to several reasons. Firstly, in the most popular paradigm of inductive learning from examples described by attribute-value pairs, it is difficult to identify universal, or even domain-specific knowledge. Attributes describing examples are highly task-specific, which reduces chances of finding their counterparts in another learning task. Secondly, many knowledge representations used in ML make it difficult to modularize or transfer knowledge. For instance, there is little chance for a fragment of a neural network to be useful in another network taught to solve a different learning task.

Challenged by these limitations, in this paper we exploit the paradigm of genetic programming (GP, [7]) as a vehicle for knowledge reuse. We demonstrate that GP is a convenient platform for this purpose, due to, among others, the symbolic knowledge representation and the ability of abstraction from a specific context. In particular, we use our method presented in [5] that implements knowledge reuse between evolving learners by allowing them to cross over parts of their genetic material. We apply the method to a large-scale task of visual learning (handwritten character recognition) and show that knowledge reuse improves the convergence of the learning process and prevents overfitting.

2 Related Work

Reported research on knowledge reuse concerns mostly knowledge reuse within a *single* learning task, with the exception of multitask learning [1] and meta-learning [19], which mostly concern learning from fixed-length attribute-value representation. In the context of GP, knowledge reuse is often connected with knowledge encapsulation [8,17,2,4], which is however not used in the approach presented here. Among reported approaches, the Case Injected Genetic Algorithm (CIGAR) by Louis *et al.* [11] resembles our contribution the most. In CIGAR, the experience of the system is stored in a form of solutions to problems solved earlier ('cases'). When confronted with a new problem, CIGAR evolves a new population of individuals and injects it periodically with such remembered cases. Experiments demonstrated CIGAR's superiority to standard GA in terms of search convergence. However, CIGAR injects *complete* solutions only, works in a strictly sequential way, and does not involve GP, making it significantly different from our approach.

In this paper, we investigate *visual* learning. As image interpretation is inherently complex, it is difficult to devise a learning method that solves such a task as a whole. Rather than that, most methods proposed so far introduce some learning or adaptation at a particular stage of image processing and analysis, which enables easy interfacing with the remaining components of the recognition system. For instance, training a machine learning classifier on some predefined image features is a typical example of such an approach. In this paper, we propose a learning method that spans the *entire* processing chain, from the input image to the final decision making, and produces a complete recognition system. Former research on such systems is rather scant [18,16,13,10,3].

3 Generative Visual Learning

The approach, originally proposed in [9] and further developed in [6], may be shortly characterized as *generative visual learning*, as our evolving learners aim at *reproducing the input image* using some simpler means. Reproduction takes place on a virtual canvas spanned over the input image. On that canvas, the learner (genetic programming individual) is allowed to perform some elementary *drawing actions* (DAs for short) in response to the input image. In particular, we consider handwritten characters and, to enable learners to restore their shapes, our DAs line sections. Fitness function compares the contents of the canvas to the input image, and rewards individuals that provide high quality of reproduction. Thus, an individual is here awarded not for the final result of decision making only, but for its 'understanding' of the pattern being analyzed.

Such an evaluation method allows us to examine the processing performed by an individual-learner in a more thorough way than in non-generative approach, where individuals are expected to produce a scalar feature or a binary decision in response to the input image. Thanks to that, the risk of overfitting, so immense in learning from high-dimensional image data, becomes significantly smaller.

To reduce the amount of data that has to be processed, our approach abstracts from raster data and relies only on selected salient features found in the input image s. The features correspond to prominent local luminance gradients derived from s using a straightforward procedure described in [5]. For each detected feature, we build a *visual primitive* (VP), described by three scalars called hereafter *attributes*; these include two spatial coordinates of the edge fragment and the local gradient orientation. The complete set of VPs derived from s, denoted in the following by P, is usually several orders of magnitude more compact than the original image s, yet it well preserves the sketch of s.

On the top level, the proposed method uses evolutionary algorithm that maintains a population of visual learners (individuals, solutions), each of them implemented as GP expression. Each learner L is a procedure written in a form of a tree, with nodes representing *functions* that process sets of VPs. The terminal nodes (named *ImageNode*s) fetch the set of primitives P derived from the input image s, and the consecutive internal nodes process the primitives, all the way up to the root node. We use strongly-typed GP (cf. [7]), which implies that two nodes may be connected to each other only if their input/output types match. The following types are used: numerical scalars, sets of VPs, attribute labels, binary arithmetic relations, and aggregators.

The GP functions may be divided into scalar functions, selectors (select some VPs based on their attributes), iterators (process VPs one by one), and grouping operators (group VPs based on their attributes and features, e.g., spatial proximity). Given these operators, an individual-learner L applied to an input image s builds a hierarchy of VP sets derived from s. Each invoked tree node creates a new set of VPs that includes other elements of the hierarchy. In the end, the root node returns a nested VP hierarchy built atop of P, which reflects the processing performed by L for s. A more detailed description of this process, including the full list of GP functions, may be found in [9].

Individual's fitness is based on DAs (drawing actions) that it performs in response to visual primitives P derived from training images $s \in S$. To reconstruct the essential features of the input image s, the learner is allowed to perform DAs that boil down to drawing sections on the output canvas c. To implement that within the GP framework, we introduce an extra GP function called *Draw*. It expects as an argument one VP set T and returns it unchanged, drawing on canvas c sections connecting each pair of VPs from T. Evaluation of L consists in comparing the contents of c to s and assumes that the difference between c and s is proportional to the minimal total cost of bijective assignment of lit pixels of c to lit pixels of s. The total cost is a sum of costs for each pixel assignment. The cost of assignment depends on the distance between pixels: when the distance is less than 5, the cost is 0; maximum cost equals 1 when the distance is greater than 15; between 5 and 15 the cost is a linear function of the distance. For pixels that cannot be assigned (e.g., because there are more lit pixels in c than in s), an additional penalty of value 1 is added to the total cost. In order to compute the minimal total cost of assignment, an effective greedy heuristic was applied.

The (minimized) fitness of L is defined as the total cost of the assignment normalized by the number of lit pixels in $s \in S$, averaged over the entire training set of images S. An ideal learner perfectly restores shapes in all training images and its fitness amounts to 0. The more the canvas c produced by L differs from s, the greater (worse) its fitness value. Thus, fitness function rewards individuals that exactly and completely reproduce as many images from S as possible, therefore promoting discovery of similarities between the training images.

In terms of ML, this generative visual learning (GVL) procedure performs *one-class learning* [15], as it uses training examples from the positive class only and tries to describe it, having no idea about the existence of other decision classes (object classes in case of visual learning). To handle a k-class classification problem, we run in parallel k independent evolutionary processes for n generations, each of them devoted to one object class. The k best individuals obtained from particular runs form the complete multi-class classifier (*recognition system*), ready to recognize new images using a straightforward voting procedure detailed in Section 5.

4 Knowledge Reuse

Given the similar visual nature of learning tasks related to particular decision classes in GVL, we expect them to require some common knowledge. Therefore, running them in isolation may be redundant, as many decision classes may need similar fragments of GP code to, e.g., detect the important features like stroke junctions. For instance, locating the lower end of the shape of letter Y presented in an image may require similar subtree of GP operators as locat-

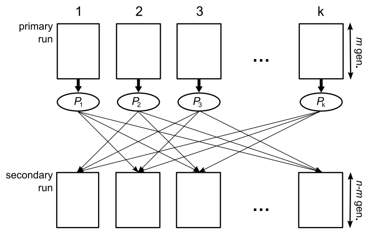


Fig. 1. The architecture of CCKR.

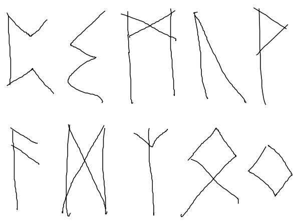


Fig. 2. Examples of handwritten runes.

ing the lower ends of letter X. To exploit such commonalities, we enable GVL to involve *cross-class knowledge reuse* (CCKR) between evolutionary processes devoted to particular classes. For the initial m generations ($m < n$), called hereafter *primary run*, evolution proceeds exactly as in GVL. As the run devoted to i^{th} decision class ($i = 1...k$) reaches the m^{th} generation, we store its population in a *pool* P_i, so that P_i constitutes a snapshot of i^{th} evolutionary run at m^{th} generation. Next, the population is re-initialized (in the same way as the initial population of the primary run), and the evolution continues for the remaining $n - m$ generations, referred to as *secondary run*.

The secondary run slightly differs from the primary one in that it activates an extra *crossbreeding operator* that is allowed to import genetic material from the pools P_i. Crossbreeding works similarly to GP crossover, however, it interbreeds an individual from the current population (a 'native') with an individual from one of the pools P_j, $j \neq i$ (an 'alien'). First, it selects a native parent from the current population using the same selection procedure as crossover. Then, it picks out an alien parent by first randomly choosing one of the pools P_j, $j \neq i$, and then randomly selecting an individual from P_j, disregarding its fitnesses. Finally, crossbreeding randomly selects two nodes N_n and N_a in the native and the alien parent, respectively, and replaces N_n by the subtree rooted in N_a. The modified native parent (offspring) is injected into the subsequent population (provided it meets the constraints optionally imposed on GP trees). Thus, crossbreeding may involve large portions of code as well as small code fragments.

Figure 1 outlines the CCKR approach. Each column composed of primary and secondary run relates to learning one decision class, and arrows depict the transfer of genetic material between them. As GVL requires k runs lasting n generations each, while CCKR involves k runs lasting m generations and k runs lasting $n - m$ generations, the total number of fitness function calls (the effort) is the same. Thus, if we ignore the cost of re-initialization of k populations and the cost of cross-breeding, which are in fact very low compared to overall computation, the time complexity of CCKR is the same as that of GVL on the average (though the actual evolution time may vary due to variability of fitness computation time). As the pools P_is, once created, remain unchanged, the runs do not have to work in parallel, but may be carried out sequentially.

5 The Experiment

In experimental part, we approach a real-world multiclass problem of handwritten character recognition. The task is to recognize letters from the Elder Futhark, the oldest form of the runic alphabet, which consists of the following characters:

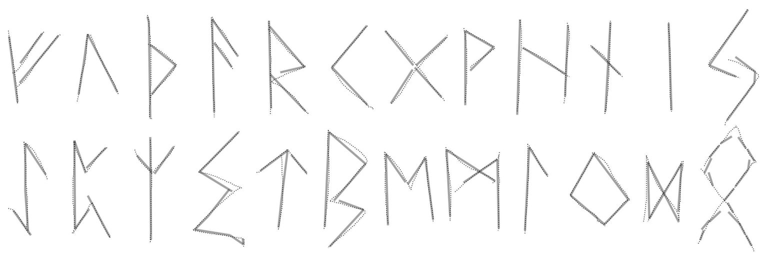
ᚠ ᚢ ᚦ ᚨ ᚱ ᚲ ᚷ ᚹ ᚺ ᚾ ᛁ ᛃ ᛇ ᛈ ᛉ ᛊ ᛏ ᛒ ᛖ ᛗ ᛚ ᛜ ᛞ ᛟ

Elder Futhark letters are written with straight pen strokes only, which makes them a good testbed for our generative recognition approach that uses sections to reconstruct the recognized shapes. Using a TabletPC computer, we prepared a training set containing 240 images (examples, objects) of $k = 24$ runic alphabet characters, each character class represented by 10 examples written by 7 persons (three of them provided two character sets). Figure 2 shows examples of selected handwritten characters.

The purpose of the experiment is to compare CCKR, the method with knowledge reuse, to the basic approach (GVL) that provides us with control results. Technically, we use generational evolutionary algorithm maintaining a population of 10,000 GP individuals for $n = 600$ generations. The initial population is created using Koza's ramped half-and-half operator with ramp from 2 to 6 [7]. We apply tournament selection with tournament of size 5, using individuals' sizes for tie breaking and thus promoting smaller GP trees and alleviating the problem of code bloat. For GVL runs, offspring are created by crossing over selected parent solutions from previous generation (with probability 0.8), or mutating selected solutions (with probability 0.2). For CCKR, the primary run lasts for $m = 300$ generations with the same settings as GVL, while in the secondary run the mutation probability is lowered to 0.1 to yield 0.1 to the crossbreeding operator. The GP tree depth limit is set to 10; the mutation and crossover operations may be repeated up to 5 times if the resulting individuals do not meet this constraint; otherwise, the parent solutions are copied into the subsequent generation. Except for the fitness function implemented for efficiency in C++, the algorithm has been implemented in Java with help of the ECJ package [12]. For evolutionary parameters not mentioned here, ECJ's defaults have been used.

To intensify the search, we split the population into 10 islands and exchange individuals between them every 20^{th} generation starting from the 50^{th} generation. During exchange, each odd-numbered island donates 10% of its well-performing individuals (selected by tournament of size 5) to five even-numbered islands, where they replace the poorly-performing individuals selected using an inverse tournament of the same size. The even-numbered islands donate their representatives to the odd-numbered islands in the same way. The islands should *not* be confused with the boxes depicting evolutionary runs in Fig. 1 – the island model is implemented within *each* evolutionary process independently.

Using these settings, we evolve and compare CCKR recognition systems to GVL recognition systems, with the latter serving as control results. In both cases, this involves running $k = 24$ evolutionary processes, each of them using training examples from one character class for fitness computation. The ensemble


(a)

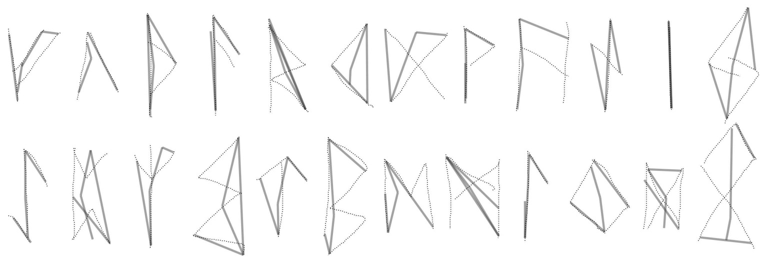

(b)

Fig. 3. Examples of letter reconstructions. The original images are drawn with a thin dotted line. In (a) each letter was reconstructed by a individual taught on a appropriate class, whereas in (b) an attempt was made to reconstruct all shapes using an individual taught on class ᛇ .

of all 24 best-of-run individuals constitute the complete *recognition system.* The recognition system undergoes evaluation on the testing set of characters, which is disjoint with the training set and contains 1440 images, that is 60 images for each character class. The system classifies an example t by computing fitnesses (responses) of all individuals for t and indicating the class associated with the individual that responded in the smallest (the best) value. Such procedure is motivated by an obvious observation, that a learner is taught to perform well on images from one class and its raw (minimized) fitness should be near 0 only for images of this class. For example, in Fig. 3a, each character was reconstructed using individual taught on its coresponding class, so all the reconstructions are good. On the other hand, in Fig. 3b, where each shape was reconstructed using the individual taught on class ᛇ , only the restoration of character ᛇ is correct[1].

Such a *simple recognition system* may be obtained at a relatively low computational expense of k evolutionary runs. Given more runs, recognition accuracy

[1] Though also restoration of ᛁ seems correct, closer examination reveals surplus overlying strokes that will be penalized by the fitness measure.

Table 1. Test-set classification accuracy for different voting methods.

Voting method	simple	vote-2	vote-3	vote-4	vote-5	vote-30
GVL	69.79±1.66	78.50±1.12	82.50±1.02	85.21±0.79	86.66±0.61	91.32
CCKR	81.94±0.89	87.88±0.49	91.19±0.41	92.58±0.31	93.18±0.27	95.56

Table 2. True positive (TP) and false positive (FP) ratios for vote-30 CCKR method.

Letter	ᚠ	ᚢ	ᚦ	ᚨ	ᚱ	ᚲ	ᚷ	ᚹ	ᚺ	ᚾ	ᛁ	ᛃ
TP	90.0%	80.0%	100%	95.0%	98.3%	100%	98.3%	81.7%	100%	93.3%	98.3%	98.3%
FP	0.0%	18.3%	21.7%	1.7%	0.0%	1.7%	3.3%	3.3%	0.0%	6.7%	8.3%	0.0%
Letter	ᛇ	ᛈ	ᛉ	ᛊ	ᛏ	ᛒ	ᛖ	ᛗ	ᛚ	ᛜ	ᛞ	ᛟ
TP	100%	98.3%	93.3%	100%	95.0%	100%	100%	96.7%	80.0%	100%	100%	96.7%
FP	3.3%	1.7%	0.0%	1.7%	3.3%	0.0%	3.3%	0.0%	21.7%	6.7%	0.0%	0.0%

may be further boosted by employing more *voters* per each decision class, as opposed to one voter per class in the above scheme. This is especially appealing in the context of evolutionary computation, as each evolutionary run usually produces a different best-of-run individual, so their fusion may result in synergy. Table 1 presents the test-set classification accuracy of GVL and CCKR for the simple recognition system and the voting method with a different number of voters. The table shows averages with .95 confidence intervals; for vote-30, confidence intervals cannot be provided as, with 30 independent runs for each character class, only one unique vote-30 recognition system can be built. Quite obviously, the more voters, the better the performance. But more importantly, no matter what the number of voters is, CCKR impressively outperforms the corresponding GVL approach. The gap between them narrows when the number of voters increases, but remains significant even for the vote-30 case.

Table 2 presents detailed test set results for the vote-30 CCKR experiment. Nine out of 24 characters were recognized perfectly. Overall, only three characters were recognized in less than 90% of cases: ᚢ , ᚹ , and ᛚ . Basing on the complete confusion matrix (not shown here due to the lack of space), we can conclude that ᚢ is sometimes mistaken for ᛚ , hence both yield high false positive errors. Similarly, the recognition system occasionally incorrectly classifies ᚹ as ᚦ . Since these pairs of letters are very similar to each other and even a human might find them troublesome, this result may be considered appealing.

6 Conclusions

Despite the large number of decision classes (24), low number of training examples per class (10), variability of handwriting styles (7 persons), and, last but not least, one-class learning (learners have no idea what the negative examples look like), the presented approach of GP-based generative visual learning performs surprisingly well, attaining near-to-perfect classification accuracy on the large and diversified test set. This encouraging result should be attributed to the generative trait of the proposed approach, which does not allow the evolving learners to ‘skim’ the training data for *any* discriminating feature, but forces them to fully ‘understand’ the recognized pattern.

The additional mechanism of cross-class knowledge reuse (CCKR) further boosts the recognition accuracy, even when the base approach (GVL) already uses an extensive voting procedure like vote-30. Most of erroneous recognitions concern character classes that are hard to tell apart also for humans. With the difficult character classes excluded (ᚢ and ᛚ , ᚹ and ᚦ), CCKR attains 98.0% recognition accuracy on the test set.

CCKR is straightforward and may be implemented by a relatively simple extension of canonical genetic programming. The method does not increase the computational effort of the learning process, and provides a significant performance improvement at the same computational expense as GVL. Thanks to one-class learning, recognition system may be easily extended by a new decision class by separately evolving an extra learner; the existing components of the recognition system do not have to be modified.

At the current stage, it is difficult to conclude if the results obtained here generalize to other variants of GP-based learning. The conservative answer should be probably negative: many other GP-based learning methods would benefit less from this form of knowledge reuse. For instance, GP expressions operating in the space of attributes in conventional learning from examples, would probably be not beneficiary of CCKR, for the reasons stated earlier in Introduction (low probability of usefulness of GP subexpressions in other learning tasks). However, this tentative conclusion should not restrain us from further investigation of the topic. Quite on the contrary, it may be a useful hint for building GP representations that are susceptible to knowledge reuse. In other words, it would be interesting to pose an inverse problem: instead of trying to devise a knowledge reuse method for a particular knowledge representation, try to define a knowledge representation that makes the knowledge reuse possible.

Acknowledgments

This work was supported by grant DS 91-443. Computations were carried out in Poznań Supercomputing and Networking Center.

References

1. R. Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, 1997.
2. E. Galvan Lopez, R. Poli, and C. A. Coello Coello. Reusing code in genetic programming. In M. K. *et al.*, editor, *Genetic Programming 7th European Conference, EuroGP 2004, Proceedings*, volume 3003 of *LNCS*, pages 359–368. Springer-Verlag, 2004.

3. D. Howard, S. C. Roberts, and C. Ryan. Pragmatic genetic programming strategy for the problem of vehicle detection in airborne reconnaissance. *Pattern Recognition Letters*, 27(11):1275–1288, 2006.
4. W. H. Hsu, S. J. Harmon, E. Rodriguez, and C. Zhong. Empirical comparison of incremental reuse strategies in genetic programming for keep-away soccer. In M. Keijzer, editor, *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, Seattle, Washington, USA, 26 July 2004.
5. W. Jaśkowski, K. Krawiec, and B. Wieloch. Knowledge reuse in genetic programming applied to visual learning. In D. Thierens, editor, *Genetic and Evolutionary Computation Conference GECCO*, pages 1790–1797. Association for Computing Machinery, 2007.
6. W. Jaśkowski, K. Krawiec, and B. Wieloch. Learning and recognition of hand-drawn shapes using generative genetic programming. In M. G. et al., editor, *EvoWorkshops 2007*, volume 4448 of *LNCS*, pages 281–290, Berlin Heidelberg, 2007. Springer-Verlag.
7. J. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1992.
8. J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In T. H. *et al.*, editor, *Proceedings of International Conference on Evolvable Systems: From Biology to Hardware (ICES-96)*, volume 1259 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
9. K. Krawiec. Learning high-level visual concepts using attributed primitives and genetic programming. In F. R., editor, *EvoWorkshops 2006*, LNCS 3907, pages 515–519, Berlin Heidelberg, 2006. Springer-Verlag.
10. K. Krawiec and B. Bhanu. Visual learning by coevolutionary feature synthesis. *IEEE Transactions on System, Man, and Cybernetics – Part B*, 35(3):409–425, June 2005.
11. S. Louis and J. McDonnell. Learning with case-injected genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 8(4):316–328, 2004.
12. S. Luke. ECJ evolutionary computation system, 2002. (http://cs.gmu.edu/eclab/projects/ecj/).
13. M. Maloof, P. Langley, T. Binford, R. Nevatia, and S. Sage. Improved rooftop detection in aerial images with machine learning. *Mach. Learn.*, 53:157–191, 2003.
14. T. M. Mitchell. The discipline of machine learning. Technical Report CMU-ML-06-108, Machine Learning Department, Carnegie Mellon University, July 2006.
15. K. M. W. Moya, M. R. and L. D. Hostetler. One-class classifier networks for target recognition applications. In *Proceedings world congress on neural networks*, pages 797–801, Portland, OR, 1993. International Neural Network Society.
16. M. Rizki, M. Zmuda, and L. Tamburino. Evolving pattern recognition systems. *IEEE Transactions on Evolutionary Computation*, 6:594–609, 2002.
17. S. C. Roberts, D. Howard, and J. R. Koza. Evolving modules in genetic programming by subtree encapsulation. In J. F. M. *et al.*, editor, *Genetic Programming, Proceedings of EuroGP'2001*, volume 2038 of *LNCS*, pages 160–175. Springer-Verlag, 2001.
18. A. Teller and M. Veloso. PADO: A new learning architecture for object recognition. In K. Ikeuchi and M. Veloso, editors, *Symbolic Visual Learning*, pages 77–112. Oxford Press, New York, 1997.
19. R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artif. Intell. Rev.*, 18(2):77–95, 2002.

An Iterative Process for Building Learning Curves and Predicting Relative Performance of Classifiers

Rui Leite and Pavel Brazdil

rleite@liacc.up.pt, pbrazdil@liacc.u.pt
LIAAD-INESC Porto L.A./Faculty of Economics,
University of Porto, Rua de Ceuta, 118-6,
4050-190 Porto, Portugal

Abstract. This paper concerns the problem of predicting the relative performance of classification algorithms. Our approach requires that experiments are conducted on small samples. The information gathered is used to identify the nearest learning curve for which the sampling procedure was fully carried out. This allows the generation of a prediction regarding the relative performance of the algorithms. The method automatically establishes how many samples are needed and their sizes. This is done iteratively by taking into account the results of all previous experiments - both on other datasets and on the new dataset obtained so far. Experimental evaluation has shown that the method achieves better performance than previous approaches.

1 Introduction

The problem of predicting the relative performance of classification algorithms continues to be an issue of both theoretical and practical interest. There are many algorithms that can be used on any given problem. Although the user can make a direct comparison between the considered algorithms for any given problem using a cross-validation evaluation, it is desirable to avoid this, as the computational costs are significant.

It is thus useful to have a main method that helps to determine which algorithms are more likely to lead to the best results on a new problem (dataset). The common approach of many methods is to store previous experimental results on different datasets. The datasets, including the one in question, are characterized using a set of measures. A (meta-)learning method is used to generate a prediction, for instance, in the form of a relative ordering of the algorithms.

Some methods rely on dataset characteristics such as statistical and information-theory measures [5, 3]. However, these measures need to be identified beforehand, which is a non-trivial task.

These difficulties have led some researchers to explore alternative ways to achieve the same goal. Some authors have used simplified versions of the algorithms referred to as *landmarks* [1, 10]. Other researchers have proposed to use the algorithms performance on simplified versions of the data, which are sometimes referred to as *sampling landmarks*. The performance of the algorithms on samples can be used again to estimate their relative performance. The use of previous information about learning curves on representative datasets is essential. Without this, sampling may lead to poor results [9].

Our previous approach [7] used the performance information given by partial learning curves to predict the relative performance on the whole dataset. This method performed better than the one that uses *dataset characteristics*. However, this had one shortcoming. It required the user to choose the sizes of the partial learning curves, *i.e.* the number and sizes of the samples used by the base algorithms.

This choice is not trivial since there is a *tradeoff* between using more information (more samples and larger samples) leading to better predictions and the cost of obtaining this information.

In this paper we describe an improved version of a previously developed method that determines the size of the samples used to create the partial learning curves. The method uses performance information concerning learning curves on other datasets and also involves conducting some experiments on a new dataset.

The planning of these experiments is built up gradually, by taking into account the results of all previous experiments – both on other datasets and on the new dataset obtained so far. This is one important novelty of our approach. Experimental evaluation has shown that the new method can achieve higher performance levels when compared to other approaches.

2 Using Sampling to Predict the Outcome of Learning

The aim is to decide which of the two algorithms (Ap and Aq) is better on a new dataset d. The performance of the algorithms on a set of samples sheds some light to the final results of the learning curves (see Fig. 1). Intuitively this approach should work better if several samples of different sizes are used for each algorithm, letting us perceive the shape of the two learning curves. We assume the existence of available information about how both algorithms perform on different datasets $d_1 \cdots d_n$ for several samples with sizes $s_1 \cdots s_m$. This information defines the relation between the performance of the algorithms on partial learning curves and the relative performance on the whole dataset. The method previously developed [7] and the enhanced version presented here are both based on this intuitive idea. The main difference between the methods is that the new method establishes the sizes of the samples. More samples are introduced if it improves the confidence of the prediction. This method also tries to minimize the costs associated with the training of the algorithms on the chosen samples.

The improved method comprises two functions: function AMDS which uses the performance of the algorithms on given samples; and function SetGen which establishes the sample sizes that should be used by AMDS. Both are described in more detail in the following section.

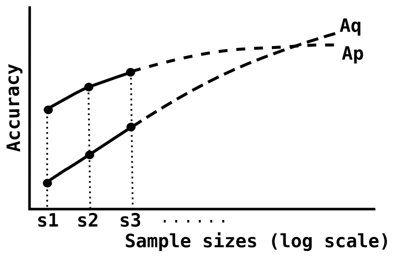


Fig. 1. Partially known learning curves

2.1 Using performance on samples of specified sizes for decision making (ADMS)

The function AMDS accepts a sequence of sample sizes [1] as an input. For example, consider the case shown in Figure 1. The input can be represented as $\{Ap_1$, Ap_2, Ap_3, Aq_1, Aq_2, $Aq_3\}$ corresponding to samples of sizes s_1, s_2, s_3 for both algorithms. Bellow we give an overview of the method.

1. Describe the new dataset d. This step involves the estimation of the algorithms performance when trained on several samples of given sizes.
2. Compute the distances between d and all the other datasets $d_1 \cdots d_n$. Identify the subset of k nearest datasets.
3. Adapt the identified pair of learning curves to the new partial learning curves built for dataset d. There is a pair of curves for each identified dataset.
4. For each pair of adapted curves decide which algorithm is better. This is decided by looking at the relative performance at the end of the curves.
5. Identify the algorithm to use on the new dataset d, by considering the results on all k neighbour datasets.

The reader can find additional details about this method in our previous work [7].

2.2 Generator of Attribute Sequences (SetGen)

We have a method (AMDS) that predicts the relative performance of classification algorithms using the performance of those on different samples. The samples are randomly drawn from the given datasets and can only assume some specific sizes $(s_1, s_2, \cdots, s_m)$. The sample sizes are passed to AMDS as an input (e.g. <s1,s2,s3> related to *Ap* and <s2,s5> related to *Aq*).

The aim of the algorithm SetGen is to determine how many samples should be used and what their sizes should be. This is not solved using a regular feature

[1] The sequences of possible sample sizes are finite (e. g.$\{s_1, s_2, \cdots, s_m\}$) and the sizes grow as a geometric sequence $a_0 \times r^n$. Here we have used sizes 91, 128 etc.

selection method (a forward selection) that try to improve the method accuracy. In our case it also deals with feature costs.

The samples are chosen taking into account the improvement of the prediction confidence. At the same time the system tries to identify those candidate solutions that increase the computational costs by the least amount. The desired sequence of samples is identified using a hill climbing approach.

An overview of the method is presented in Fig.2. It starts with a given sequence of meta-features for the two algorithms (Ap, Aq). Each sequence consists of performance values on a given sequence of samples. The algorithm works in an iterative fashion. In each step it tries to determine how the sequence should be extended. Various alternatives are considered and evaluated. As we have pointed out before, the aim is to improve the confidence of the prediction (which algorithm to use). The best extension is selected and the process is repeated while the confidence of the prediction increases.

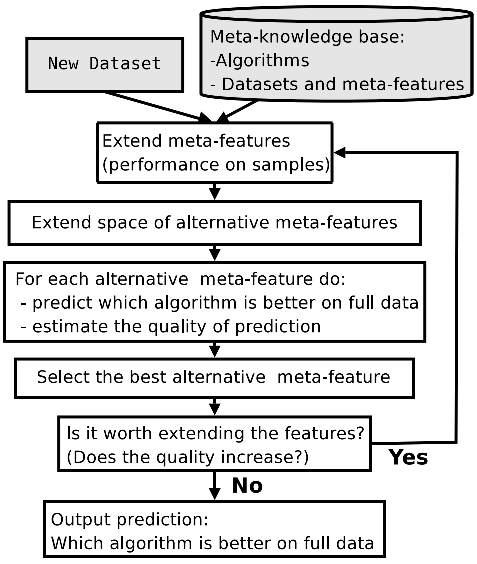

New Dataset
Meta-knowledge base:
-Algorithms
- Datasets and meta-features
Extend meta-features
(performance on samples)
Extend space of alternative meta-features
For each alternative meta-feature do:
- predict which algorithm is better on full data
- estimate the quality of prediction
Select the best alternative meta-feature
Is it worth extending the features?
(Does the quality increase?)
Yes
No
Output prediction:
Which algorithm is better on full data

Fig. 2. Iterative process of characterizing the new dataset and determining which algorithm is better

Fig. 3 shows more details about how the search is conducted. The algorithm starts with an initial set of attributes ($a0$). At each step it expands the current state into its successors. The successor states are the sets of attributes obtained

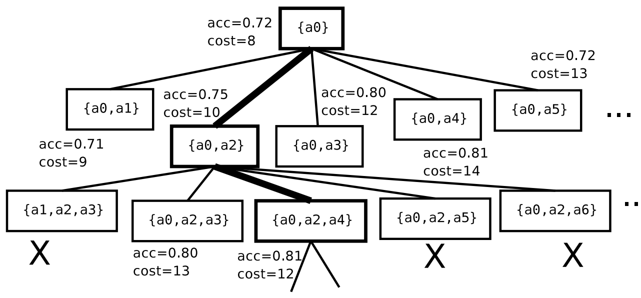


Fig. 3. An example of search in the attribute space

by adding a new attribute to the current set. In the example the successors of $(a0)$ are $(a0, a1)$, $(a0, a2)$ etc. For simplicity we use a_j to represent a *generic attribute*. For each state (represented by the particular set of attributes) the (meta-)accuracy of the method is estimated, as well as the cost of obtaining the attributes.

Once a state is defined the next one is chosen among the successors that improve the meta–accuracy by at least Δ (for instance, the value 0.01). Finally, the one with the minimum cost is chosen. In our example (Fig. 3) the algorithm moves from state $(a0)$ to state $(a0, a2)$. In the next step it chooses the state $(a0, a2, a4)$. This continues until none of the successors improves the accuracy by at least Δ.

Let us clarify the issue of what the generic attributes used here represent. Each particular a_j represents either a particular attribute of algorithm Ap_j (representing the performance of Ap on sample of size s_j), or some attribute of algorithm Aq_k or a combination of both attributes (for instance (Ap_4 and Aq_6)). This means that on each iteration the SetGen method can either insert a new attribute related to one particular algorithm (say Ap) or insert two attributes one related to Ap and another related to Aq. This allow the method to extend either one of the partial learning curves or the two at the same time.

Another aspect needs still to be clarified. When the method attempts to extend a particular state, it verifies whether the extension is *valid*. Valid extensions introduce attributes that are in general *more informative* [2] than those used so far. In Fig. 3 some extensions of states that were considered, but are in fact invalid, are marked by “X”.

Evaluation of Attribute Sequences

[2] For instance, the attribute Ap_3 is considered more informative than Ap_2, as the corresponding sample is larger. Therefore, the system can, add Ap_3 to the set containing Ap_2, but the opposite is not possible.

As described earlier, each state is characterized by two different values, accuracy and computational cost. The algorithm needs to anticipate what would be the accuracy of the method if it used a specific set of attributes (*candSet*). This estimate, evaluated using *EAcc* function, expresses the chances of success of the method on the new dataset. It is calculated using a leave–one–out evaluation on the existing datasets in the meta-knowledge base. Let us assume that it contains information about D datasets. The method is applied to each dataset d_i of D by using the given set of attributes *candSet*. The prediction concerning the algorithm to use is compared with the correct decision stored in the meta-knowledge base. The method succeeds on d_i if its decision on this dataset is the same as the correct decision retrieved.

After repeating this procedure for all D datasets it is necessary to aggregate this information in order to obtain an estimate of the accuracy of the method for the attribute set *candSet* on different datasets. This could be done by calculating an average. However, this would give equal importance to all datasets, without regarding whether they are similar to the dataset in question or not. For this reason the method uses a *weighted average* that corrects this shortcoming. The following equations define the *EAcc* function.

$$EAcc(c) = (1 + \sum_{d_i \in D} [success(d_i, candSet) \times w_i])/2 \tag{1}$$

where D is the set of known a datasets, w_i represents a weight and $success(\ldots)$ is $+1$ (-1) if the method gives the correct (wrong) answer for d_i using the attributes indicated in candSet. The weight is calculated as follows: $w_i = w'_i / \sum_k w'_k$ where $w'_i = 1/(dist(d, d_i) + 1)$.

The computational cost estimate is calculated by using a similar leave–one–out mode. For each dataset d_i the meta–database is used to retrieve the training time spent on evaluating all the attributes in *candSet*. The final value is the weighted average of times for the specified attributes.

$$ECost(candSet) = \sum_{d_i \in D} Cost(d_i, candSet) \times w_i \tag{2}$$

This provides an estimate for the training time needed to compute the same attributes on a new dataset.

3 Empirical Evaluation

The decision problem concerning whether algorithm A_p is better than A_q can be seen as a classification problem which can have three different outcomes: 1 (or -1) if algorithm *Ap* gives significantly better (or worse) results than A_q, or 0 if they are equivalent (not significantly different).

In the experimental study our first aim was to determine the accuracy and the computational cost of our approach (AMDS–SetGen). Additionally we also aimed at comparing these results to a previous method (MDC)[3] which relies on dataset characteristics instead. This method can be breefly described by:

1. Compute the characterization measures for all datasets (including the new one).
2. Compute the distance between the new dataset and the stored ones.
3. Choose the k stored datasets (neighbours) that are "nearest" to the new dataset (according to the distance).
4. Use the algorithm that was most often the best one on the identified nearest datasets.

The predicted class was compared with the true classification determined by a usual cross–validation evaluation procedure on each dataset for the two given algorithms. A statistical test (t–test) was used to compute the statistical significance.

Instead of using the usual accuracy measure, we have used a different measure that is more suited for our classification task with 3 possible outcomes. The errors are called *penalties* and are calculated as follows. If a particular method (e.g. AMDS–SetGen) classifies some case as +1 (or -1), while the true class is 0 (the given algorithms are not significantly different) then, from a practical point of view the method did not fail, because any decision is a good one. Therefore the penalty is 0. However if the method classifies the dataset as 0, while the true class is +1 (or -1) then we consider that the method partially failed. The penalty is 0.5. If the classification is +1 (or -1), while the true class is -1 (or +1), this counts as a complete failure, and the penalty is set to 1. The corresponding accuracy, referred to as *meta–accuracy*, is computed using this formula $1 - \sum_{i \in D} \frac{penalty(i)}{\#D}$.

In this empirical study we have used the following 5 classification algorithms, all implemented within Weka [11] machine learning tools: J48 (C4.5 implemented in weka), JRip - rule set learner (RIPPER [4]), logistic [6], MLP - multi-layer perceptron, and Naive Bayes. Using this setting we get 10 classification problems, one for each pair of algorithms. We have used 40 datasets in the evaluation. Some come from UCI [2], others were used in the project METAL [8]. The evaluation procedure used to estimate the accuracy of the methods is the leave–one–out [3] to on the datasets.

In Figure 4 we present the performance reached by the methods in terms of meta–accuracies. The AMDS–SetGen results have been obtained with the parameter Δ=0.07. The figure also shows the performance of MDC and the default accuracy on each meta-classification problem.

On average with AMDS-SetGen the accuracy improves by 12.2% (from 79.6% to 91.8%) when compared to the default accuracy. Moreover, our method improves the performance by 11% when compared to MDC (that reaches 80.8%), while MDC only improves the performance by 1.2% when compared to the default accuracy. Our method gives better results than MDC on all problems. The

[3] We plan to use a Bootstrap-based error estimator to reduce the variance usually associated with leave-one-out error estimators.

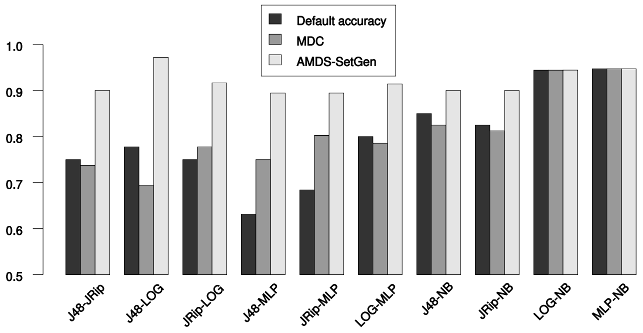


Fig. 4. Result concerning meta–accuracies

largest improvements are observed on the hardest problems where the default accuracy is close to 50%. In such situations our method can attain nearly 25% of improvement.

Regarding computational costs the results of ADMS-SetGen and MDC are compared to the cost of deciding which algorithm is better using cross–validation (CV). To express the run–time savings, Figure 5 shows the ratio between of the time spent by AMDS–SetGen and the time spent by CV. In most cases AMDS–SetGen is much faster than CV (the ratio is on average about 0.2).

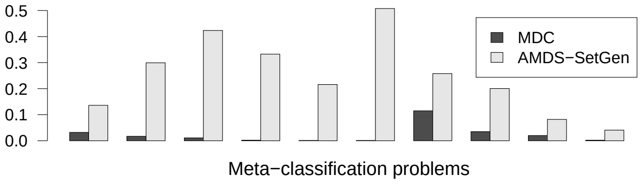


Fig. 5. Results concerning costs

3.1 Further Experimental Results and Discussion

Variation of Δ and its effects: We briefly discuss why we have chosen Δ=0.07. We have conducted further experiments by using the following values for the Δ parameter: 0.2, 0.1, 0.07, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001. If we use small Δ values the meta–accuracy increases, at the same time as the increasing of computational time. The value of Δ = 0.07 seems to be the best compromise, leading, on average, to a meta–accuracy of 0.92 and to a computational time (expressed as a ratio) of 0.2. [4]

AMDS using fixed samples: The user can choose some simple samples like s_j and also combinations like ($s1$,$s2$), ($s1$,$s2$,$s3$) etc. Although the method performs reasonably well the SetGen method is generally more accurate.

A typical case of selected samples: Typically the samples selected by SetGen are not the same for both algorithms and the sample sequences can have gaps. The following example shows the selected samples obtained for J48 *vs* JRip while making the prediction for dataset *musk*: J48 samples=($s1, s5$) JRip samples=($s1, s4$). This is a typical case. Having gaps which happens frequently represents a useful feature as it allows time savings.

Extension to N classifiers: Although this paper focuses on the problem of determining which of the 2 given classification algorithms should be used on a new dataset, it does not represent a serious limitation. If we had N classification algorithms and wanted to find out which one(s) should be used, one could conduct pairwise tests and use the one that wins more often. If the number of alternatives is larger, it is advisable to divide the algorithms into subgroups and establish a winner of each subgroup and then repeat the process. However, this issue exceeds the aims of this paper.

4 Conclusions

In this paper we have described a meta-learning scheme that exploits sampling to determine which of the two given methods is better. The method automatically establishes how many samples are needed and their sizes.

The method uses stored performance information concerning learning curves on other datasets to guide the construction of the sequences of samples to be used on a new dataset. The method involves conducting some experiments on the new dataset. The plan of these experiments is built up gradually, by taking into account the results of all previous experiments - both on other datasets and on the new dataset obtained so far.

The method extend the sequence of samples to improve the results. It tries to determine whether this would improve the confidence of the prediction, while controlling the process and looking for extensions with minimal costs. This is one important novelty of this approach.

Experimental evaluation has shown that the new method achieves good performance. The (meta-)accuracy is on average 91.8% , which is higher than any of the approaches reported before. Besides, the user does not have to be concerned with the problem of determining the sizes of the samples. This represents a significant improvement over previous methods in dealing with the problem of predicting the relative performance of learning algorithms in a systematic manner.

[4] If the Δ is set to 0.05 the meta–accuracy remains almost the same (0.92) but the time increases (ratio 0.3). If on the other hand Δ is adjusted to 0.1, the meta–accuracy decreases to 0.88 and the method becomes faster (ratio 0.16).

References

1. H. Bensussan and C. Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD2000)*, pages 325–330. Springer, 2000.
2. C. Blake and C. Merz. UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/mlrepository.html, 1998.
3. P. Brazdil, C. Soares, and J. Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50:251–277, 2003.
4. W. W. Cohen. Fast effective rule induction. In A. Prieditis and S. Russell, editors, *Proc. of the 12th International Conference on Machine Learning*, pages 115–123, Tahoe City, CA, July 9–12, 1995. Morgan Kaufmann.
5. D. e. C. D. Michie. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.
6. S. le Cessie and J. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.
7. R. Leite and P. Brazdil. Predicting relative performance of classifiers from samples. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 497–503, New York, NY, USA, 2005. ACM Press.
8. Metal project site. http://www.metal-kdd.org/, 1999.
9. C. Perlich, F. Provost, and J. S. Simonoff. Tree induction vs. logistic regression: a learning-curve analysis. *J. Mach. Learn. Res.*, 4:211–255, 2003.
10. B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *Proceedings of the 17th International Conference on Machine Learning (ICML-2000)*, pages 743–750. Stanford, CA., 2000.
11. I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical machine learning tools and techniques with java implementations, 1999.

Towards Automating Goal-driven Learning

Maarten W. van Someren

Department of Human-Computer Studies
University of Amsterdam
Kruislaan 419
1098 VA Amsterdam
The Netherlands
maarten@science.uva.nl

Abstract. A number of computational tools exist to perform parts of the task of constructing knowledge-based systems. These tools come from research and development in Machine Learning and Knowledge Acquisition. More fully automating this process requires a language for specifying the goal of the construction process and knowledge for when and how to invoke a tool. This paper presents a language, LML, for formulating what we call learning goals, in the sense of descriptions of target systems. LML can also be used to express functional models of learning and acquisition tools. These models can be used to select tools to achieve a goal and to prove that the resulting knowledge system satisfies the goal. To demonstrate the use of the language, we present a system that effectively uses these functional models to select combinations of tools for automated construction of knowledge-based systems.

1 Introduction

In this paper we view learning as constructing a system that satisfies certain constraints and we consider the problem of selecting and combining learning systems and applying them to available systems to achieve a learning goal. Most learning systems construct systems that are generalizations of existing systems (we view training examples in combination with a database lookup procedure as a "system"). Some learning systems do not (only) perform generalization but (also) speedup or change of representation language. For many learning tasks, several learning systems need to be combined to achieve a learning goal.

A key problem is he nature of learning goals. Here we define concepts and a language for (a) expressing learning goals, in the form of constraints of a target system to be produced by learning and (b) defining functional models of learning systems. The models of learning systems will be used to select a useful learning system and to infer if the goal of learning has been achieved. Below we also describe a system (GDL for Goal-Driven Learner) that achieves learning goals by selecting knowledge or data and tools, applying the tools and using the results to construct a knowledge system that provably satisfies its learning goal.

2 Models of knowledge systems

A knowledge system can consist of a single “knowledge base” with a single “problem solver” or a compound system (see section 7). Problem data can be entered to the knowledge system and then the problem solver is run on the problem data and the knowledge base to produce a “solution” (or failure). The meta-language LML (Learning Modelling Language) is based on the properties of knowledge systems listed in Table 2. Examples of problem solvers are: decision tree interpreter, prologSolve (acts like Prolog), table lookup and nearest neighbour matcher. We illustrate the properties with a toy knowledge system (Figure 1).

```
tree(thrombosis,
       [yes : tree(pneumonia,
                     [yes : leaf(medication1),
                      no  : leaf(medication2)]),
        no : tree(lungEmbolia,
                      [yes: leaf(medication3),
                       no : leaf(medication1)])])
```

Fig. 1. exampleKS

The values of LML properties are obtained in different ways. The problem solver is recognised by its label. kbGrammar and pGrammar are recognised by language recognisers. pVocabulary, kbVocabulary, kbLexicon, solutions and size are recognised by a parser/recogniser using the kbGrammar. SolutionTime is found by benchmarking: generating problems, solving them and measuring time. Function and minSolutionTimeForFunction are not measured and can only be inferred. Learning goals only involve relative constraints on the function and therefore this need not be computed.

3 Learning Goals

A learning goal is a description of a target knowledge system, represented as constraints on the values of LML properties. Constraints can be relative, referring to one or more other knowledge systems. For example, a constraint on the target system can be *size below 120 (words)* or *size below size system-2* or *size below sum(size(system-2), size(system-3))*.

Learning goals are expressed in a simple constraint language with constraints on sets, grammars and numbers. A special relation is best generalization of. It is useful to distinguish between any subsumption relation between two functions and best generalization of. For example, by joining two knowledge systems we may obtain a new system with a function that subsumes the functions of the input systems but this is different from applying induction to generalize the function of one of the systems.

Property	Meaning	Value in example
kbGrammar	the most specific of a set of predefined grammars that recognises the knowledge base; the primitive elements of a grammar are the lexical categories	decision tree grammar
problem solver	a (predefined) problem solver	decision tree interpreter
kbVocabulary	the set of words in knowledge base	{thrombosis, pneumonia, medication1, medication2, lungEmboli, medication3}
kbLexicon	the words associated with their lexical category	{attribute: {thrombosis, pneumonia, lungEmboli}, value {yes, no}}
pGrammar	the most specific grammar that recognises the problem data	flat-attribute-value
pLexicon	assigns problem vocabulary to grammatical categories	
pVocabulary	set of words that can appear in a problem and can be used by the knowledge system	{thrombosis, pneumonia}
solutions	set of solutions that can be found for at least one problem	{medication1/2/3}
size	number of words in knowledge base	23
function	relation between problems and solutions	-
solutionTime	average time for solving a problem	0.01
minSolutionTime-ForFunction	Boolean that holds if the solution time of a system is minimal (for function)	yes

Table 1. Overview of LML properties

Consider the following example of a learning goal, adapted from an actual practical problem [1]. Several knowledge systems are available: *initial*) built before the current project, *patients* records of patients with their diagnoses, and several human experts. The target system is required to cover the function of *initial* and *patients*. The *patient* records are specific must be generalized. During the initial stage an inventory was made of symptoms and complaints and other data about a patient such as age, sex, conditions under which the complaints began, medical history and of relevant diagnostic categories. For example, the user requires the target system to diagnose cases of thrombosis that share symptoms with *initial* and *patients*. Regulations (for ambulance dispatching) required the system to find solutions in less than 30 seconds. The representation language for the target system is a language based on rules, to make the knowledge base comprehensible. This learning goal can be expressed in LML as follows:

Relations between properties of different knowledge systems (as appear in goals) can be inferred (a) from the LML models of tools and input and output knowledge systems and (b) by applying general knowledge about relations and properties. For example, GDL uses subsumption relations between grammars, symmetry and transitivity of set and numerical relations, knowledge about possible combinations of knowledge base grammars, problem solvers and problem

Find a knowledge system KS3 such that:

1. There is a knowledge system KS1 such that:
   - bestGeneralization of the function of patients
   - problemVocabulary equal to those of patients
   - solutions equal to those of patients
2. and a knowledge system KS2 such that:
   - solutions include thrombosis
   - overlap in pVocabulary with the pVocabulary of KS1
3. and then such that for KS3:
   - solutions include the union of solutions of KS1 and KS2
   - kbGrammar subsumed by hornGrammar
   - problem solver is prologSolve
   - problemVocabulary includes the union of the
   - problemVocabularies of KS1 and KS2
   - function subsumes the function of KS1, KS2 and initial
   - solutionTime below 30 seconds

Table 2. Example learning goal

grammars. GDL also knows that:

for all KS, F: bestGeneralization(F1, KS1) & equal(F1, F2) & function(F2, KS2) → bestGeneralization(F1, KS2)

and

minSolutionTimeForFunction(KS1, F1) & equal(F1, F2) & function(F2, KS2) → minSolutionTimeForFunction(KS1, F2)

4 Functional models of tools

LML properties and relations are also used for functional models of learning tools. Tool models are represented as LML relations between input and output knowledge systems. Table 3 gives an example. The model consists of preconditions on the input data and a model that specifies the form of the knowledge system that is produced by the learner. The most important effect is that the function of the decision tree will be a best generalization of the function of the input but also problem solving will take less time and the size will be reduced. The precondition for applying this tool is: *kbGrammar of InputKS subsumed by attValExamples*. Table 3 gives the functional model.

This model is used in GDL to to infer some properties of the result of applying the tool and to evaluate if applying it to a knowledge system will produce a new knowledge system that is closer to the goal.

kbGrammar	subsumed by decisionTreeGrammar
problem solver	is decisionTreeInterpreter
pGrammar	subsumed by attValExamples
solutions	setequal to solutions of InputKS
kbVocabulary	setequal to kbVocabulary of InputKS
pVocabulary	subset of pVocabulary of InputKS
function	is best generalization of function of InputKS
size	below size of InputKS
solutionTime	below solutionTime of InputKS
solutionTime	minimal for function

Table 3. LML model of a decision tree learner

5 Human experts and knowledge elicitation

Knowledge can be elicited from a human expert. To include this in our model and system we assume that (the knowledge of) a human expert can be modelled like the knowledge of a system, using LML properties. By applying an elicitation tool to a human expert, a knowledge system is obtained. Selection of experts and elicitation tools can be included by constructing LML models of experts and tools.

6 The GDL system

The purpose of building the GDL system is to evaluate if LML can be used to control learning processes. The task of GDL is, given a repository of knowledge systems, a set of LML profiles of experts (and access to them), a learning goal and a collection of tools with their LML models, to construct a knowledge system that satisfies the learning goal.

The implementation of GDL includes a collection of tools. There are tools for generalization (constructing a system with a function that subsumes the function of its input), speedup (reducing solution time) and conversion (changing the grammar). GDL also includes two simple tools for the elicitation, analogous to structured editors for e.g. Repertory Grids. New tools can easily be included but GDL needs an LML model of their function.

Figure 2 summarizes the algorithm.

GDL first constructs LML models of all knowledge systems in the repository. Next, knowledge systems that are relevant for the goal (because they share problem or knowledge base vocabulary) are selected from the repository. If the learning goal refers to words about which no knowledge system is available at all then immediately a relevant expert is located and elicitation about this word is initiated. The relevant knowledge systems are ordered by the number of unsatisfied goal constraints. GDL then selects the best knowledge system and searches for the best applicable (by its preconditions and LML model) tool. If there are no applications of tools that achieve any improvement then GDL can transform

```
GDL(input: Goal, Knowledge systems; output: OutputKS):

1. Find Relevant Systems(input: Solutions, pVocabulary,
      KbVocabulary, Knowledge systems; output: Relevant systems)

2. IF Missing Words(input: Solutions, pVocabulary,
      KbVocabulary, Relevant systems; output: Missing)
   AND available(Expert, Missing)
   THEN elicit(input: Expert, Missing, pVocabulary,
      Solutions; output: ExtraKS)
3.
  3.1 Order(Relevant systems, ExtraKS, OrderedRefs)
  3.2 UNTIL test(ConjGoal, OrderedRefs) succeeds DO:
    3.2.1 select best(OrderedRefs, CurrentBest);
    3.2.2 select(CurrentBest, ConjGoal, MLTool);
    3.2.3 IF applicable(CurrentBest, MLTool) AND
          predicted Improvement(CurrentBest, MLTool,
               ConjGoal)
          THEN (apply(MLtool, CurrentBest, NewKS)
```

Fig. 2. Toplevel algorithm of GDL

a knowledge system into its preferred default kbGrammar, Horn clauses. This trick enables steps that do not directly approximate the goal but that can be useful intermediate steps. GDL does not apply the same operator twice. If there is no (new) step possible and the goal is not achieved, GDL stops with failure.

Consider again the learning goal in Table 3. This goal involves two subgoals that are expressed referring to another knowledge system. Assume that initially GDL has access to seven knowledge systems (see Table 10) and two sources: a human expert on lung diseases (*lung expert*) and one on psychosomatic diseases (*psyc*). For this problem, the knowledge systems *contra indications*, *causal knowledge*, *classification*, *therapy* and the expert *psyc* are not needed.

Figure 3 shows how GDL addresses this learning goal. The boxes are knowledge systems, the ellipses applications of the tool written inside. GDL is applied to each subgoal. If a constraint is not achieved then GDL backtracks to another tool or knowledge system. GDL first constructs LML models of all input knowledge systems in the repository. The first subgoal is then to construct a knowledge system that is a “best generalization” of patients. First the vocabulary is extracted from the subgoal. This vocabulary is used to select relevant knowledge systems. As it happens, all systems in the repository share vocabulary and thus are possibly relevant. GDL now orders the knowledge systems by their distance to the current learning goal. “Patients” itself is of course the knowledge system that is nearest to the goal of being a generalization of patients. GDL then searches for a tool that has not been applied to patients before and that will produce a system that, according to its LML model, is closer to the (sub)goal than patients itself. Using the functional model of tools, GDL predicts that the decision tree learning tool idt will achieve the goal constraint best generalization and applies it. This results in a new knowledge system, a decision tree (ks21 in Figure 3). GDL constructs the LML model of the new ks21 and stores the fact that the function of ks21 is a best generalization of the function of patients. From this GDL infers that the ks21 satisfies the first subgoal.

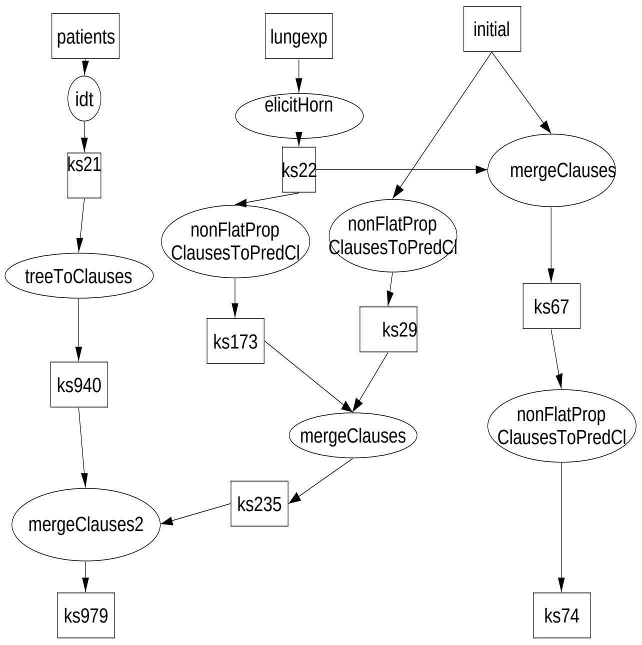


Fig. 3. Example scenario

The next subgoal (b in Table 3) is to construct a knowledge system that includes thrombosis as a solution and that has a problem vocabulary that overlaps with that of patients and initial. GDL selects the relevant knowledge systems from the repository and finds that none of these includes the word thrombosis. An elicitation goal is constructed: a knowledge system that includes thrombosis in its solutions and that has a problem vocabulary that overlaps with that of patients and initial. Only “lung expert” is relevant because this expert can produce knowledge about thrombosis. GDL now searches for an elicitation tool that can extract knowledge from this expert, elicits a set of Horn clauses and adds a “prologSolve” problem solver producing a new knowledge system (ks22 in Figure 3) which directly satisfies the subgoal. The final subgoal (c in Table 3) is to construct a knowledge system that subsumes the functions of the two systems that satisfy the first two subgoals and also the function of initial, and that solves problems within the time limit. GDL orders the relevant knowledge systems (patients, initial, ks21 and ks22) by distance to this subgoal. The best two knowledge systems are initial and ks22. Neither of these satisfies the learning goal. No tool would produce any advance towards the goal and so GDL transforms initial into this form, resulting in ks29. Merging initial with the elicited system (ks22) can be an improvement because it includes thrombosis as a solution. The new system (ks67) is now the best candidate. GDL again converts this new system into its preferred language. The resulting system (ks74) could be merged with some of the systems in its repository but this does not improve it. There is also no other learning tool that would improve it. GDL considers again its next best, ks67, the converted version of initial. This also cannot be improved. The next possibility is to convert the elicited knowledge (ks22) into the preferred language. This gives ks173 which, like ks22 just before, is now the current best system in the repository and so it is considered for further learning. Merging this with ks29 is possible because both have the same grammar and this will improve ks173 (and also ks29) because it subsumes the function of initial and includes thrombosis as solution. The result is ks235.

GDL evaluates the decision tree (ks21) very low because it is relatively far from the goal: it has only part of the solutions, the wrong grammar and problem solver and only part of the function. Because other systems cannot be improved, at this point GDL now converts ks21 to Horn clauses. The result is ks940. GDL then solves the problem by merging ks940 with ks235 and computes its LML model. During the process GDL has stored the facts that the function of ks21 is a best generalization of that of patients, and the function of ks940 is equal to that of ks21. The function of ks29 is equal to that of initial and the function of ks235 subsumes that of ks29 (and of ks173). The function of ks979 subsumes that of ks940 and ks235. From these facts GDL can derive that ks979 satisfies the requirements on the function.

7 Structured or compound knowledge systems

Sofar we considered learning goals about a single knowledge system. Here we extend this to structured, "compound" knowledge systems and goals. A "compound knowledge system" consists of basic knowledge systems (each with its own problem solver), a dataflow structure specifying their input-output relations and a "dataflow problem solver". The latter simply presents the output of one or more components as input to the successors in the dataflow structure. A learning goal may specify a dataflow structure with "subgoals" for the components and (optionally) additional constraints on the compound system as a whole.

For example, suppose that we have the following knowledge systems in the repository: patients (patient records), initial (prototype diagnostic system), causal knowledge (relations data between symptoms, patient properties and possible causes), classification (classifies causes as a diagnosis), therapy (finds therapy for causes) and contra indications (relates data about patients to possible therapies). A compound goal can be a target system with a dataflow structure that connects a component for finding diagnoses of patient symptoms and complaints, a component that uses the diagnoses to find therapies (medication and other interventions) and a component that uses other patient information than symptoms and complaints to check if a therapy is appropriate for the patient. For GDL a decomposition of the learning goal simplifies achieving it - if the decomposition is correct. Because of space restrictions we do not discuss details and examples of learning compound systems.

8 The efficiency of GDL

Is LML control knowledge (the LML models and the control methods included in GDL) effective in controlling learning? Does GDL reduce the branching factor compared to blind search? To obtain an estimate, we performed experiments with GDL on several learning problems. We analyse the effect of three control heuristics: selection of relevant knowledge systems and experts, selecting knowledge systems, preconditions of tools, selection of tools for knowledge systems (by predicted improvement). The experiments consist of following the path through the search space. At each state the number of successors is counted with and without the heuristic filter of predicted improvement. A successor is a combination of knowledge system(s) and an applicable tool. Next we count the number of predicted improvements. One of the combinations with the best prediction is actually applied. This produces another knowledge system for the repository, which results in general in more possible combinations of tools and knowledge systems than before, because the new knowledge system can be input for learning tools. When a subgoal is achieved, the process starts from the beginning with a new (sub)goal (and with a repository that contains the products of previous learning).

For example, for the problem described in section 6 we find that GDL selects 7 knowledge systems as relevant for the first subgoal. The other systems are (correctly) considered irrelevant because they do not share vocabulary with the learning goal or with a relevant system. The first subgoal does not involve any experts. There are 20 tools and some of these take more than one input. This gives a total of 529 possible combinations of input knowledge systems with tools. Of these only 34 combinations (6%) are applicable if the conditions for the tools are applied. The preconditions of the learning tools leave less than 10 percent combinations and the LML model of the effect of the tool leaves about 10 percent of those. In the end about one half percent of the total number of combinations of tool and available knowledge systems and about 10% of the valid candidates (according to preconditions) is predicted to produce an improvement. Response times are relatively independent of the size of the knowledge systems because computation of LML properties and applications of learning operations are performed only once for each knowledge system. This shows that GDL is very effective in controlling search.

9 Discussion

We presented a language, LML, for specifying requirements on a target system in the form of constraints on properties of the target system. The GDL system demonstrates that LML can be used to automatically and efficiently construct systems that provably satisfy a set of constraints. This is achieved by the use of models of knowledge systems and functional models of tools. Experiments with GDL show that the control heuristics (selection of promising knowledge systems and tools, avoiding redundant operations) are indeed effective in guiding the search through the space that is defined by the learning tools and given knowledge systems.

The approach of the MUSKRAT project (e.g. [2, 3, 3]) is very similar to ours, especially the most recent formulations. MUSKRAT uses example problems (with their solutions) to find a knowledge system such that the constraints of this system entail that this system can possibly solve the example problem. From the perspective of LML and GDL, this is a different and perhaps complementary way to address the key problem of describing the (required) function of a knowledge system. Where LML characterises the function of a knowledge system in terms of solutions, problem vocabulary and (subsumption) relations to other systems, MUSKRAT uses more precise descriptions of the function but with a more ad hoc flavour and the need to construct these by hand. An advantage of the MUSKRAT approach is that, unlike LML, MUSKRAT can express different relations between the same problem vocabulary and solutions. On the other hand, LML and GDL use properties of knowledge systems that can be computed automatically (or relations that can be inferred from models of tools) and uses a more principled approach to modelling knowledge systems and tools.

In the context of ML and Data Mining, LML and GDL are comparable to systems like MLT-CONSULTANT (e.g. [4]) and knowledge acquired in the STATLOG project (e.g. [5]). However, these systems are focused on advice about aspects of inductive inference and do not include elicitation, data transformation, speedup learning, operations on data sets or explicit learning goals. By including elicitation and effectively integrating learning tools we apply and extend Michalski's model of inferential learning which distinguishes a wide range of tools but does not use these to define a language for learning goals [6]. On the other hand, LML does not include concepts for more detailed properties of tools and knowledge systems, such as those used in MLT-CONSULTANT, STATLOG and METAL (e.g. [7]. When enough knowledge about the effect of tools becomes available then LML should be extended in this direction.

LML and GDL show how a language for learning goals and functional models of learning tools provides a unified view of various types of ML and KA and how this can be used to automatically control the application of a range of tools to construct knowledge systems from systems that are available in a repository and sources with a known profile of expertise.

References

1. Post, W., Wielinga, B.J., de Hoog, R., Schreiber, G.: Organizational modeling in commonkads: The emergency medical service. IEEE Expert **12**(6) (1997) 46–52
2. White, S., Sleeman, D.H.: A constraint-based approach to the description of competence. In Fensel, D., Studer, R., eds.: EKAW. Volume 1621 of Lecture Notes in Computer Science., Springer (1999) 291–308
3. White, S., Sleeman, D.H.: A constraint-based approach to the description and detection of fitness-for-purpose. Electron. Trans. Artif. Intell. **4**(B) (2000) 155–183
4. Sleeman, D.H., Rissakis, M., Craw, S., Graner, N., Sharma, S.: Consultant-2: pre- and post-processing of machine learning applications. Int. J. Hum.-Comput. Stud. **43**(1) (1995) 43–63
5. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994)
6. Michalski, R.S.: Inferential theory of learning as a conceptual basis for multistrategy learning. Machine Learning **11** (1993) 111–151
7. Kalousis, A., Theoharis, T.: Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection. Intell. Data Anal. **3**(5) (1999) 319–337

Learning to Evaluate Conditional Partial Plans

Sławomir Nowaczyk and Jacek Malec
Department of Computer Science, Lund University
{slawek|jacek}@cs.lth.se

We study agents situated in partially observable environments, who do not have sufficient resources to create conformant (complete) plans. Instead, they create plans which are conditional and partial, execute or simulate them, and learn from experience to evaluate their quality. Our agent employs an incomplete symbolic deduction system based on Active Logic and Situation Calculus for reasoning about actions and their consequences. An Inductive Logic Programming algorithm generalises observations and deduced knowledge so that the agent can choose the best plan for execution.

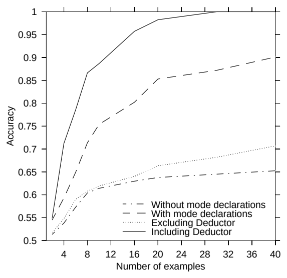
To be fully rational, situated agents need to consciously alternate between reasoning, acting and observing their environment, or even do all those things in parallel. We aim to achieve this by making the agents create short partial plans and execute them, learning more about their surroundings throughout the process. They create several partial plans and reason about usefulness of each one, including what knowledge can it provide. They generalise their past experience to evaluate the likelihood of plans leading to the goal. The plans are conditional (i.e. actions to be taken depend on observations made during execution), which makes quality estimation less situation specific.

The architecture of our agent consists of four main functional modules. Each of them is responsible for a different part of agent's rationality, but the overall intelligence is only achievable by the interactions of them all. The *Deductor* module is the one responsible for classical "reasoning". It uses a logical formalism based on combination of Active Logic [1] and Situation Calculus [2] (as introduced in [3]) to find out consequences of the agent's current beliefs. Based on the domain knowledge and previous observations, it analyses possible actions and predicts the effect of their execution.

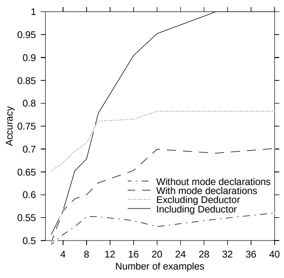
The second module is a *Planner*, which generates partial, conditional plans applicable in the agent's current situation. The third main module, *Actor*, oversees Deductor's reasoning process and evaluates plans the latter has come up with, trying to find out which is the most useful one to perform. For this paper, Actor waits until Deductor terminates and only executes plans after this happens, but in general it is Actor's responsibility to balance acting and deliberation.

Finally, the *Learner* module analyses the agent's past experience and induces rules for estimating quality of plans. Results of learning process are used both by Deductor and by Actor. In particular, since the plans Deductor reasons about are partial, it can be very difficult to estimate whether a particular plan is a step in the right direction or not. Using machine learning techniques is one way in which this could be achieved.

Our initial experiments concerned learning how to detect "bad" plans early, so that Deductor does not need to waste time deliberating about them. We have used the Inductive Logic Programming algorithm called PROGOL [4], since it is among the best known ones. PROGOL is based on the idea of *inverse entailment* and it employs a covering approach similar to FOIL, in order to generate hypothesis consisting of a set of clauses which cover all positive examples and do not cover any negative ones.



(a) Results in Wumpus domain.



(b) Results in Chess domain.

In the first experiment (curve marked "Without mode declarations"), we used as little domain-specific knowledge as possible, in particular we have not provided any *mode declarations* for PROGOL. The goal of mode declarations is to reduce the hypothesis search space by limiting types of predicate arguments. The second curve ("With mode declarations") clearly shows that providing even such a small amount of domain knowledge greatly improves quality of learned hypothesis. It can be also easily seen that the accuracy in the Wumpus domain is significantly higher than the one in the Chess domain. Nevertheless, the learning is still not fully successful. This is due to overfitting and the fact that the search space is too large for PROGOL to handle sufficiently well.

Because of that, we have limited the amount of knowledge used for learning: seemingly, presenting all of the agent's knowledge to the ILP algorithm is not the best idea. As a start, we have decided to use only the initial domain definition and the observations that the agent made in previous situations. The results of learning can be seen on curve marked "Excluding Deductor", so named since they roughly correspond to an agent who does not have a specialised deduction module and uses learning only. Finally, in our fourth and final experiment, we have selected only the most relevant parts of knowledge generated by Deductor and presented them to PROGOL. In the Wumpus case this included $maybeWumpus$, $noWumpus$ and $knowsWumpus$ predicates, while in Chess it included $notProtected$, $distanceTwo$, and $distanceTwo$ predicates. As can be seen from the curve marked "Including Deductor", the agent managed to perfectly identify bad plans from as few as 30 examples chosen at random, in both domains.

It is interesting to note that as few as five *hand-chosen* example plans suffice for PROGOL to learn the correct definition for the Wumpus domain, which opens up interesting possibilities for an agent to *select* learning examples in an intelligent way.

References

1. Purang, K., Purushothaman, D., Traum, D., Andersen, C., Perlis, D.: Practical reasoning and plan execution with active logic. In Bell, J., ed.: Proceedings of the IJCAI-99 Workshop on Practical Reasoning and Rationality. (1999) 30–38
2. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. The MIT Press (2001)
3. Nowaczyk, S.: Partial planning for situated agents based on active logic. In: Workshop on Logics for Resource Bounded Agents, ESSLLI 2006. (2006)
4. Muggleton, S.: Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming **13**(3-4) (1995) 245–286

Author Index