Proceedings of the Twelfth Belgian-Dutch Conference on Machine Learning

Edited by Marco Wiering

institute of information and computing sciences, utrecht university

technical report UU-CS-2002-046

www.cs.uu.nl

Proceedings of the Twelfth Belgian-Dutch Conference on Machine Learning

Edited by

Marco A. Wiering

Intelligent Systems Group
Institute of Information and Computing Sciences
Utrecht University

4 December, 2002
Utrecht University, The Netherlands

Contents

Preface

The Twelfth Belgian-Dutch Conference on Machine Learning (Benelearn'02) has gathered a wide variety of researchers interested in many different topics in machine learning (ML). Decision trees, Bayesian networks, neural networks, and genetic algorithms are used for mostly supervised learning tasks. Multiple authors also propose the use of relational representations (e.g. predicate logic) to strengthen the expressive power of the chosen representation (most often decision trees).

There is also some work on reinforcement learning. Kurt Driessens and Saso Dzeroski describe how guided trials can be used to speed up relational reinforcement learning. Leonid Peshkin and Edwin de Jong describe an approach to enhance cooperation and inductive transfer for multi-agent reinforcement learning.

There is a little work on unsupervised learning; only the paper by Jakob Verbeek, Nikos Vlassis, and Ben Kröse describes a novel unsupervised learning algorithm for non-linear data projection. It is also worth noting that there are no papers on support vector machines, which receive increasing attention at many international machine learning conferences. Finally, Jeroen van Maanen discusses the use of algorithmic information theory to extract a universal learning agent from its interactions with an environment.

We hope that the different topics lead to interesting discussions and we look forward to an interesting and pleasant day.

Marco Wiering
December, 2002

Sponsors:

The workshop was partially sponsored by SIKS (www.siks.nl)
School of Information and Knowledge Systems, the Netherlands.

Invited Talk:

Helge Ritter

Organising Committee:

Program Committee:

Other Contributors:

Institute of Information and Computing Sciences
Utrecht University

Benelearn'02 Home Page

Conference Program

Utrecht, The Netherlands, December 4, 2002

Wednesday 4 December 2002

- **09:30** - Registration and Coffee
- **10:00** - Invited Talk by Prof. Helge Ritter:
  Toward Robots that Learn to Imitate Actions
- **11:00** - Coffee break
- **11.20 - 12.40 Morning Session** Session chair:
- **11:20** - On Using Guidance in Relational Reinforcement Learning
  Kurt Driessens and Saso Dzeroski
- **11:40** - Context-based Policy Search: Transfer of Experience Across Problems
  Leonid Peshkin and Edwin de Jong
- **12:00** - Identifying Mislabeled Training Examples in ILP Classification Problems
  Sofie Verbaeten
- **12:20** - Improving the Efficiency of ILP Systems Using an Incremental Language Level Search
  Rui Camacho
- **12:40 - 13:40** - Lunch
- **13.40 - 15.20 Afternoon Session 1** Session chair:
- **13:40** - Locally Linear Generative Topographic Mapping
  Jakob Verbeek, Nikos Vlassis, and Ben Krose
- **14:00** - Evolving Causal Neural Networks
  Marco Wiering
- **14.20** - Model Growth
  Jeroen van Maanen

- **14:40** - An Approach to Noncommunicative Multiagent Coordination in Continuous Domains
  Jelle Kok, Matthijs Spaan, and Nikos Vlassis

- **15:00** - Non-Universal Suffrage Selection Operators Favor Population Diversity in Genetic Algorithms
  Federico Divina, Maarten Keijzer, and Elena Marchiori

- **15:20 - 15:40** - Coffee break

- **15:40 - 17:40 Afternoon Session 2** Session chair:

- **15:40** - Induction of Supermodels
  Hendrik Blockeel

- **16:00** - Selecting Relevant Features for Splice Site Prediction by Estimation of Distribution Algorithms
  Yvan Saeys, Sven Degroeve, Dirk Aeyels, Yves van de Peer, and Pierre Rouz\'e

- **16:20** - Detecting Orthogonal Class Boundaries in Entropy Behavior
  Floor Verdenius and Maarten van Someren

- **16:40** - Supervised Learning of Bayesian Network Parameters Made Easy
  Hannes Wettig, Peter Grunwald, Teemu Roos, Petri Myllymaki, and Henry Tirri

- **17:00** - Datasize-Based Confidence Measure for a Learning Agent
  Wojciech Jamroga

- **17.20** - Labeling and Splitting Criteria for Monotone Decision Trees
  Jan Bioch and Viara Popova

- **17:40 - 18:40** - Drinks

- **19:30** - Social dinner (organized by Benelearn’02)

Labeling and Splitting Criteria for Monotone Decision Trees

Jan C. Bioch and **Viara Popova**
Dept. of Computer Science
Erasmus University Rotterdam,
P.O. Box 1738, 3000 DR Rotterdam.
{bioch,popova}@few.eur.nl

Abstract

This paper focuses on the problem of monotone decision trees. It addresses the question how to label the leaves of a tree in a way that guarantees the monotonicity of the resulting tree. Two approaches are proposed for that purpose - dynamic and static labeling which are also compared experimentally.

The paper further addresses the problem of splitting criteria in the context of monotone decision trees. Two criteria from the literature are compared experimentally - the entropy criterion and the number of conflicts criterion - in an attempt to find out which one fits better the specifics of the monotone problems and which one better handles monotonicity noise.

Keywords: ordinal classification, monotone decision trees, noise, pruning, labeling, splitting criteria, entropy

1 Introduction

Ordinal classification considers problems that are monotone i.e. if x, y are data points such that $x \leq y$ ($x_i \leq y_i$ for each attribute i) then their labels should satisfy $\lambda(x) \leq \lambda(y)$) and therefore aims at generating classifiers that satisfy this constraint. In practice such problems appear in different domain area. An example can be given with credit rating where if one applicant for credit outperforms another on all criteria then he should be given at least the same chance for being approved. Other examples can be given from the areas of financial management (e.g. bankruptcy prediction, bond rating), marketing, human resources management, etc.

The general methods for generating classifiers cannot guarantee that the monotonicity constraint will be satisfied, therefore different methods are necessary. For example well-known algorithm such as CART and C4.5 are not guaranteed to construct monotone trees. Ordinal classification has been studied in the context of logical analysis of data, decision trees, decision lists, rough sets theory, etc. by a number of authors, e.g. (Crama et al., 1988), (Ben-David, 1995), (Makino et al., 1996), (Bioch and Potharst, 1997), (Greco et al., 1998), (Bioch, 1998), (Bioch and Popova, 2000), (Potharst and Bioch, 2000), (Cao-Van and Baets, 2002), (Bioch and Popova, 2002).

This paper discusses some aspects of the problem in the framework of monotone decision trees (MDT). An extension of the classical decision tree algorithm for dealing with ordinal data was first proposed in (Makino et al., 1996) for 2-class problems. A more general approach applicable to k-class problems is proposed in (Bioch and Potharst, 1997; Potharst and Bioch, 2000). In (Bioch and Popova, 2002) a number of methods were presented for dealing with noise with respect to monotonicity.

The classical decision tree algorithm can be characterized by three rules: a splitting rule which defines how to split a node, a stopping rule defining when to stop growing a branch and turn the current node to a leaf and a labeling rule defining how to label the new leaf. The algorithm for MDT generation presented in (Bioch and Potharst, 1997; Potharst and Bioch, 2000) extends the original algorithm by means of adding one more rule, the update rule, which takes care of preserving the monotonicity property of the tree. The update rule fires as soon as a new node is entered and tries to add two new points to the data set - the minimal and the maximal possible data point for the current node (also called corners). If the corners do not already exist in the data set, they are labeled with the minimal and the maximal possible label respectively for this node and added to the data set.

The MDT algorithm requires a monotone data set. In practical applications, however, noise can cause inconsistencies of the type: data points $x \leq y$ with labels $\lambda(x) > \lambda(y)$. The extension of the algorithm proposed in (Bioch and Popova, 2002) allows the generation of monotone trees from non-monotone data. That is achieved by an alteration of the update rule to not only add the corners to the data set but also, if they are already present but inconsistent, to relabel them with consistent class values. The algorithm is further augmented with methods for pruning in order to limit the tree size while keeping the monotonicity property.

This paper extends some of the methods presented in (Bioch and Popova, 2002) and discusses the issue of splitting criteria in the context of ordinal classification. Section 2 gives a brief introduction to the MDT algorithm. Section 3 presents the extension for dealing with monotonicity noise. Section 4 discusses the problems of pruning and labeling MDT. Two approaches for labeling are presented in subsections 4.1 and 4.2 and compared in 4.3. Section 5 discusses two splitting criteria - the entropy criterion and a more 'consistency-oriented' criterion we refer to as 'number of conflicts'. Section 6 gives the results of the experiments performed on the comparison of the two labeling approaches(subsection 6.1) and on the comparison of the two labeling approaches (subsection 6.2). Section 7 gives some conclusions and some possible directions for future research.

2 Monotone decision trees

Let T be a node of the tree $\mathcal{T}$ generated thus far on the data set $D \subseteq \mathcal{X}$ where $\mathcal{X}$ is the input space. T can be represented as $T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$ where $a(T), b(T)$ are called the lower/upper corners of T respectively. In the following the corners will be denoted by simply a and b when no ambiguity occurs. The original MDT algorithm is given in figure 1. In order to guarantee the monotonicity of the tree, an update procedure is performed on the data set by adding at most 2 new data points with consistent labels. The update procedure is performed every time a node is considered for splitting and the added points are the lower and the upper corners of the node (if they are not already present in the data set). The labels are chosen to be $\lambda_{max}(a)$ and $\lambda_{min}(b)$ respectively, which are defined in the following, where c_{min}/c_{max} are the lowest/highest class in the data set:

$$\downarrow x = \{y \in \mathcal{X} : y \leq x\}$$

$$\uparrow x = \{y \in \mathcal{X} : y \geq x\}$$

$$\downarrow D = \bigcup_{x \in D} \downarrow x$$

$$\uparrow D = \bigcup_{x \in D} \uparrow x$$

$$\lambda_{min}(x) = \begin{cases} \max\{\lambda(y) : \ y \in D \cap \downarrow x\} & \text{if } x \in \uparrow D \\ c_{min} & \text{otherwise.} \end{cases}$$

$$\lambda_{max}(x) = \begin{cases} \min\{\lambda(y) : \ y \in D \cap \uparrow x\} & \text{if } x \in \downarrow D \\ c_{max} & \text{otherwise.} \end{cases}$$

Using this way of labeling we guarantee that the lower corner gets the minimal label possible for the node and the upper corner gets the maximal possible label. At the same time the new points remain consistent with the rest of the data, therefore the monotonicity constraint is not violated.

x	a_1	a_2	a_3	a_4	a_5	a_6	λ
1	0	0	1	1	0	1	0
2	1	1	2	1	3	1	0
3	0	1	1	0	0	1	0
4	2	3	1	3	3	1	1
5	1	0	2	2	3	1	1
6	0	0	0	3	2	2	1
7	2	2	1	1	1	2	1
8	2	4	2	2	2	3	2
9	1	1	2	1	3	2	2
10	3	2	1	0	0	1	2
11	3	2	2	1	2	2	3
12	3	3	4	1	2	2	3
13	4	2	3	3	3	3	3
14	3	3	3	4	1	3	3
15	4	4	2	3	0	1	3

Table 1: The example data set

```
split(node $T$):
    **update($T$)**;
    if $T$ is homogeneous
        label $T$;
    else
        split $T$ into disjoint $T_L$ and $T_R$;
        split($T_L$);
        split($T_R$);

**update(node $T$)**:
    if $a \notin D$
        $\lambda(a) = \lambda_{max}(a)$;
        add $a$ to $D$;
    if $b \notin D$
        $\lambda(b) = \lambda_{min}(b)$;
        add $b$ to $D$;
```

Figure 1: The monotone decision tree algorithm

In order to illustrate how the algorithm works we use the example data set from table 1. In the first step, the root of the tree (containing the whole data set) is considered for splitting. The update rule fires and the corners $a = (0,0,0,0,0,0)$ and $b = (4,4,4,4,3,3)$ are added with labels $\lambda(a) = 0$ and $\lambda(b) = 3$. Further $a_1 > 2$ is chosen for a test of the node and the data set is divided between the two children. Following the left-depth-first strategy we consider the left child for splitting. The corners are $a = (0,0,0,0,0,0)$ and $b = (2,4,4,4,3,3)$ where a is already present and b is added with a label $\lambda(b) = 2$. The algorithm continues with finding the best split,

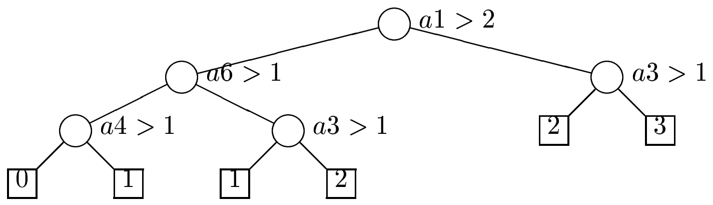


Figure 2: The MDT generated for the example data set

etc. The full monotone tree generated from this data set is given in figure 2.

3 Monotone decision trees from noisy data

The extension of the MDT algorithm for handling noisy data alters the update rule to not only add the corners if they are not present but also to relabel them with consistent labels if they are present. In this way the algorithm tries to repair the inconsistencies caused by monotonicity noise. The new update rule, for a node T with lower/upper corners respectively a and b, is given in figure 3.

```
update(node $T$):
    $l_1 = \lambda_{max}(a)$; $l_2 = \lambda_{min}(b)$;
    if $a \in D$
        relabel $a$: $\lambda(a) = l_1$;
    else
        label $a$: $\lambda(a) = l_1$; add $a$ to $D$;
    if $b \in D$
        relabel $b$: $\lambda(b) = l_2$;
    else
        label $b$: $\lambda(b) = l_2$; add $b$ to $D$;
```

Figure 3: The extended update rule

Theorem: *The MDT algorithm with the extended update rule of figure 3 always generates a monotone tree.*

(See (Bioch and Popova, 2002) for a proof.)

To illustrate the algorithm we introduce monotone inconsistency in the example data set of table 1 - we change the label of data point x_3 from 0 to 1. Thus we introduce an inconsistent pair of data points (x_2,x_3). The output of the algorithm on the new data set is given in figure 4.

Note that a simple criterion for checking the monotonicity of a tree (Potharst and Bioch, 2000) can be defined as follows.

Let $\mathcal{L}$ be the set of leaves of a tree $\mathcal{T}$ and $\mathcal{N}$ be the set of nodes of $\mathcal{T}$. We define a relation on $\mathcal{N}$: for $T, T' \in \mathcal{N}$

$$T \leq T' \Leftrightarrow a(T) \leq b(T').$$

Let $T, T' \in \mathcal{L}$ such that

$$T = \{x \in \mathcal{X} : a(T) \leq x \leq b(T)\}$$

and

$$T' = \{x \in \mathcal{X} : a(T') \leq x \leq b(T')\}.$$

Then the tree is monotone if for any choice of T and T':

$$T \leq T' \Rightarrow \lambda(T) \leq \lambda(T').$$

4 Pruning and labeling rules that guarantee the monotonicity

The original MDT algorithm adds new points to the data set, therefore the set grows and that is one of the reasons why the generated trees tend to be bigger than the ones generated by the classical DT algorithms. The noise with respect to monotonicity can cause further increase of the tree size. One way to correct that is to prune the trees to reduce the size while keeping the misclassification rate as low as required. Pruning can be performed in two ways: pre-pruning which prematurely stops the growth of the tree when a predefined threshold is reached and post-pruning which first grows the full tree and then cuts branches from it until a predefined criterion is fulfilled.

In both situations it is important to apply consistent rules for giving labels to the new leaves in such a way that the resulting tree is monotone. Dynamic labeling refers to giving labels while the tree is being generated, as soon as a node is turned to a leaf. It can be used together with pre-pruning. Static labeling refers to the process of giving consistent labels to the tree that is already generated. It can be used on any trees with non-homogeneous leaves, e.g. trees generated with pre- and post-pruning as well as trees generated with other methods.

This section discusses the two labeling methods.

4.1 Dynamic labeling

One important difference between the static and the dynamic way of labeling is how much of the information in the tree is already available. While in the static case the tree is grown and all the information about the shape of the tree, the corners of the leaves and the labels of these corners is available, in the dynamic case we only have a part of the tree built and

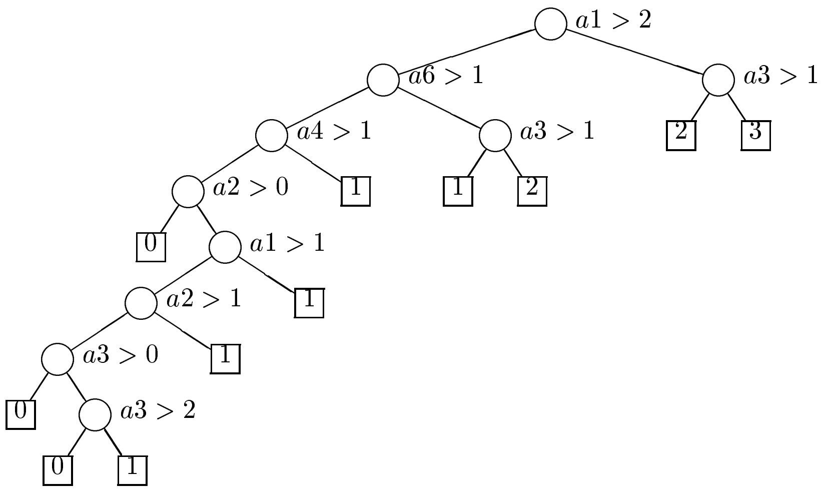


Figure 4: MDT on the non-monotone data set

```
label leaf $T$:
    $\lambda(T) = L(T)$;
    $\lambda(a(T)) = \lambda(T)$;
    $\lambda(b(T)) = \lambda(T)$;
```

Figure 5: The dynamic labeling rule

the new label should be based on partial information. Therefore, for dynamic labeling, an important factor is the search strategy used for building the tree, i.e. which part of the tree is expected to be built already and what kind of information will influence the new labels. In the following we assume the depth-first strategy for growing the tree. First we note an observation that holds for this strategy.

Lemma: *Let $T, T' \in \mathcal{L}(\mathcal{T})$ in the monotone tree $\mathcal{T}$ generated with depth-first strategy. Let $T \leq T'$. Then leaf T is generated before leaf T'.*

(See (Bioch and Popova, 2002) for a proof.)

The general form of the dynamic labeling rule for a non-homogeneous leaf T is given in figure 5 where L is the labeling function. Two possible forms are proposed for the labeling function as follows:

$$L \in \{L_{min}, L_{max}\}$$

$$L_{min}(T) = max\{\lambda(T') | T' \leq T\}$$
$$L_{max}(T) = min\{\lambda(T') | T \leq T'\}$$

Theorem: *Let $\mathcal{T}$ be a tree generated using the extended MDT update rule for a threshold of at least m points in a leaf, $m \geq 1$. Let the leaves be labeled using the dynamic labeling rule of figure 5 where one of the following strategies is applied:*

1. *$L(T) = L_{min}(T), \forall T \in \mathcal{L}$,*
2. *$L(T) = L_{max}(T), \forall T \in \mathcal{L}$.*

Then $\mathcal{T}$ is monotone.

Proof: Let the tree be generated using the left-depth-first strategy. The newly generated leaf to be labeled is denoted by T. If T is the first leaf then the current set of labeled leaves is monotone.

Let T be not the first leaf and let the current set of labeled leaves be monotone. We label $\lambda(T) = L(T)$.

Let $L = L_{min}(T)$. Then

$$\lambda(T) = max\{\lambda(T') | T' \leq T\}.$$

Therefore, for each $T' \leq T$, $\lambda(T') \leq \lambda(T)$. Therefore the state of the tree is still monotone. The proof is analogous for $L = L_{max}$.

In the same way it can be proved for right-depth-first strategy. $\square$

The following observations can help speeding up the computation:

Observation 1: *For the left-depth-first strategy the following holds: $L_{max}(T) = \lambda(b(T))$.*

Observation 2: *For the right-depth-first strategy the following holds: $L_{min}(T) = \lambda(a(T))$.*

The experiments showed that $L_{max}(a)$ tends to favor the lower classes while $L_{min}(b)$ tends to favor the higher classes. Therefore they provide a choice to the decision-maker for a more pessimistic against a more optimistic prediction.

4.2 Static labeling

As mentioned before, static labeling is performed on the fully generated tree, where the information about all the leaves is already available.

The earlier defined relation over the nodes/leaves of a tree is not transitive. We define now a transitive closure relation, denoted by $\trianglelefteq$, in the following way. For $T', T'' \in \mathcal{N}$: $T' \trianglelefteq T'' \Leftrightarrow \exists T_1, T_2, ..., T_m \in \mathcal{N}$ such that $T' \leq T_1 \leq T_2 \leq ... \leq T_m \leq T''$.

In our implementations we used the algorithm of Warshall (Warshall, 1962) for computing the transitive closure.

We define Λ_{min} and Λ_{max} over the set of leaves $\mathcal{L}$ as follows:

$$\Lambda_{min}(T) = max\{\lambda(a(T_1))|T_1 \in \mathcal{L}, T_1 \trianglelefteq T\}$$

$$\Lambda_{max}(T) = min\{\lambda(b(T_2))|T_2 \in \mathcal{L}, T \trianglelefteq T_2\}$$

Then Λ is defined as: $\Lambda \in \{\Lambda_{min}, \Lambda_{max}\}$.

It can be shown that $\Lambda_{min} \leq \Lambda_{max}$ and they are monotone labelings, i.e. for leaves $T_1 \leq T_2$, $\Lambda_{min}(T_1) \leq \Lambda_{min}(T_2)$ and $\Lambda_{max}(T_1) \leq \Lambda_{max}(T_2)$.

Theorem: *Let $\mathcal{T}$ be a tree generated without labeling. Let the leaves be visited following either left-depth-first or right-depth-first order. The leaves are labeled using one of the following strategies:*

1. $\lambda(T) = \Lambda_{min}(T), \forall T \in \mathcal{L}$,
2. $\lambda(T) = \Lambda_{max}(T), \forall T \in \mathcal{L}$.

Then the resulting tree is monotone.

Proof: Let $\Lambda = \Lambda_{min}$ and the order of visiting the leaves is left-depth-first. Let the last leaf labeled is T. If T is the first labeled leaf then the current set of labeled leaves is monotone.

Let T be not the first labeled leaf and let the current set of labeled leaves be monotone.

$\lambda(T) = \Lambda_{min}(T) = max\{\lambda(a(T'))|T' \trianglelefteq T\}$

Let us assume that the resulting set of labeled leaves is no longer monotone. Using left-depth-first means that there are no labeled leaves T' such that $T \trianglelefteq T'$. Therefore $\exists T' \trianglelefteq T, \lambda(T') > \lambda(T)$. Since $\lambda(T) \geq \lambda(a(T'))$, then $\lambda(T') > \lambda(a(T'))$

$$\Rightarrow \exists T'' : T'' \trianglelefteq T', \lambda(T') = \lambda(a(T'))$$

$$\Rightarrow T'' \trianglelefteq T \Rightarrow \lambda(T) \geq \lambda(a(T'')) = \lambda(T')$$

which is a contradiction with the assumption.

The proof is analogous for Λ_{max} and for right-depth-first order. □

4.3 Comparison

In order to get more insight in the performance of the two presented approaches for labeling, experiments were conducted. The main goal was to compare the results from the four possible combinations of settings: dynamic labeling with L_{min}, dynamic labeling with L_{max}, static labeling with Λ_{min} and static labeling with Λ_{max}.

The trees were generated using left-depth-first search strategy with pre-pruning at predefined thresholds for the minimal number of points in a node. Therefore the size of the trees was almost always the same.

The results showed that:

- In general the dynamic labeling produces trees with lower misclassification rate on unseen data. The reason for that is most probably in the fact that the tree changes dynamically and it is possible to generate trees that have different shape while in the static case the shape of the tree is already fixed and the only feature that changes is the labels of the leaves.
- The difference between the two labeling functions for both approaches was not so clear-cut and no conclusion can be made yet on which one is preferable.

Details on the experimental setting and the results are given in section 6.

5 Splitting criteria for MDTs

One of the important rules in building a decision tree is the splitting rule which defines how to split the current node in two branches. For a binary tree a split is a tuple of the type $\langle a_i, v_j\rangle$ where a_i is the attribute to split on and v_j is the cut-off value. There is a lot of research done on finding good criteria for choosing good splits for the classical DT algorithms. One of the most successful approaches is to choose the split which produces the highest decrease in the entropy or the highest information gain. These two notions are usually defined as follows. The entropy of a node T is:

$$Ent(T) = -\sum_{i=1}^{n} p_i log_2 p_i$$

where p_i is the proportion of data points with class i in T for an n-class problem. Then the information gain of an attribute a_i and cut-off value v_j is:

$$Gain(T, a_i, v_j) = Ent(T) - \sum_{k \in \{L,R\}} \frac{|T_k|}{|T|} Ent(T_k)$$

where T_L/T_R are respectively the left and the right child of T when split on $\langle a_i, v_j\rangle$.

However for the specific case of MDT it is not clear which splitting criteria are good. Intuitively they should not only attempt to produce smaller trees but also provide fast decrease in the inconsistencies in the tree. One such criterion was suggested in (Cao-Van and Baets, 2002) although no experimental results were presented on its performance compared to other criteria. The criterion aims at reducing the number of non-monotone pairs of points in the resulting branches. It chooses the split with the least number of inconsistencies/conflicts.

Let the current node T be split into the following non-overlapping subsets:

$$T' = \{a(T') \leq x \leq b(T')\}$$

and

$$T'' = \{a(T'') \leq x \leq b(T'')\}.$$

Let T' be the left branch. Therefore $T' \leq T''$. If for all points $x \in T', y \in T''$ it is true that $\lambda(x) \leq \lambda(y)$ then the split is monotone. There might be however points such that $\lambda(x) > \lambda(y)$. The number of those inconsistent pairs is counted for all possible splits of the current node and the one with the lowest count is chosen.

The experiments that were performed for this paper aim at giving more insight in the performance of the two mentioned criteria in the context of MDT. The two main aspects for comparison are the size and the accuracy of the generated trees. The experimental results point at the following observations.

- On monotone data sets no criterion is systematically better than the other.
- With the increase of monotonicity noise in the data, the entropy criterion tends to generate smaller trees.
- With the increase of monotonicity noise in the data the conflicts criterion generates more accurate trees with lower misclassification rate on unseen data.

Intuitively the reason for the difference is probably in the orientation of the two criteria. The entropy criterion strives at generating smaller trees by reducing the diversity of classes in the leaves as fast as possible. The conflicts criterion, on the other hand, only cares about the consistency/monotonicity of the tree and therefore produces trees that are larger but better fit the 'character' of the data. In this way it handles more successfully monotonicity noise but at the cost of generating bigger trees.

Details about the data used, the performed experiments and the experimental results are given in section 6.

6 Experiments

As it was mentioned in sections 4 and 5, experiments were performed in two directions. The first direction aims at comparing the dynamic and the static approach for labeling with their variations. The results are presented in section 6.1. The other direction of experiments aims at comparing the entropy and the conflicts splitting criteria in the context of monotone decision trees. The experimental setting and the results are presented in section 6.2. In all experiments left-depth-first search strategy was used.

For all experiments, the starting data sets were taken from UCI Machine Learning Repository (Blake and Mertz, 1998). The Nursery data set is a real-world monotone data set which represents applications for a nursery school, contains 12960 instances described by 8 attributes and covers the whole input space. The Cars data set is an artificial set describing cars by their properties and classifying them according to their acceptability for a buyer. The original data was not strictly monotone and for that reason one of the values of one attribute was removed. The resulting set was monotone. It contains 1153 instances described by 6 attributes. Since both data sets cover the whole input space, they were also used as test sets in the experiments.

For the experiments random samples of size 200 points were drawn from both data sets. Monotone inconsistencies were introduces in the data in the following way: a pair of comparable data points (such that either $x \leq y$ or $x \geq y$) from different classes was chosen at random and the labels were switched. That results in one or more inconsistent pairs. The procedure can be repeated to introduce more noise.

Since both data sets cover the whole input space it was possible to test the misclassification rate on the whole data instead of using separate test samples.

6.1 Comparison of the dynamic and the static labeling

For the comparison of the two labeling approaches, four samples were used - one from the cars data set and three from the nursery data. For each the algorithms were applied for the thresholds of minimum 5, 10, 15 and 20 points in a node. For the static case the leaves were left unlabeled and the labeling was performed at the end. Eight different combinations of settings were tested: entropy versus conflicts splitting criteria, L_{min} versus L_{max} and Λ_{min} versus Λ_{max}. The results about the splitting criteria are discussed in section 6.2 and for the purpose of the current discussion the results will the averaged over the two criteria.

As a test set, the full data sets were used as they cover the whole input space (1153 points for the cars data set and 12960 points for the nursery data set). The number of misclassified points per sample is given in table 2. The first column contains the value of the threshold. Columns 2 and 3 give the results for the tree labeled with static labeling for Λ_{min} and Λ_{max} respectively Columns 4 and 5 give the analogous results for dynamic labeling and L_{min} and L_{max} respectively.

As it was mentioned before, the size of the trees was the same for the static and the dynamic case (per sample and threshold). Having this in mind, it can be seen that the number of points misclassified by the trees which was labeled dynamically is systematically lower than (often more than twice as low as) that of the trees labeled statically.

On the other hand no definite answer can be given to the question which labeling function to use: L_{min} versus L_{max} and Λ_{min} versus Λ_{max}. In the static case the choice of labeling function made hardly any

	static		dynamic	
thr	Λ_{min}	Λ_{max}	L_{min}	L_{max}
5	205	205	93	285
10	350	350	156	407
15	408	408	217	425
20	408	408	219	434
5	2495	2495	1708	1915
10	4012	4012	2172	2154
15	4530	3524	2395	2292
20	4950	4950	2512	2424
5	2436	2436	1821	1694
10	3380	3380	2486	1892
15	4653	4653	2587	2404
20	5055	5055	2967	2404
5	4152	4152	2097	3351
10	5402	5402	2489	3509
15	6892	6892	2983	4219
20	7398	7403	3130	4555

Table 2: Experimental data on the labeling approaches

difference in the results while in the dynamic case for some samples L_{min} is better and for the other L_{max} gives better results.

These results are also confirmed by the experiments performed by a master's student (van Eikeren, 2002) working under the guidance of the authors of this paper.

6.2 Comparison of the entropy and the conflicts splitting criteria

The experiments were designed to answer two main questions in the comparison of the two splitting criteria:

- Which criterion performs better on monotone data by means of generating smaller and/or more accurate monotone trees?
- Which criterion handles better monotonicity noise/inconsistencies by means of generating smaller and/or more accurate monotone trees?

In order to give some insight on these questions the experiments were conducted in two different settings. In the first part 10 monotone samples were drawn from the nursery data set. MDTs were generated from each of them using the two criteria. The results - the number of misclassified points over the full data sets and the number of tree nodes - are given in table 3. It can be seen that the performance of the two criteria is different on the different samples and no definite conclusion can be made on which one fits better monotone problems.

For the second part of the experiments, 4 samples were chosen - 1 from the cars data set and 3 from the nursery data set (rows 2, 3 and 7 from table 3). For each data set 6 noisy versions (5 for the cars data) were generated by switching the labels of 1 to 7 pairs of points. For each such series, MDTs were generated using the entropy and the conflicts criteria.

entropy		num conflicts	
miscl	nodes	miscl	nodes
1612	321	1512	233
1516	95	1522	211
1520	87	1752	131
1645	85	1458	247
1478	143	1330	159
1354	151	1429	783
1277	163	1718	195
1266	107	1232	149
1087	401	1672	241
1504	343	1622	327

Table 3: Experimental data on the splitting criteria for monotone samples

The results per series are presented in table 4. The first column contains the number of non-monotone pairs of points in the set. Columns 2 and 3 give the number of misclassified points on the full data set and the number of nodes for the tree generated with the entropy criterion, while columns 4 and 5 give the respective information for the number of conflicts criterion.

It can be seen that with the increase of inconsistencies the number of conflicts criterion gives systematically better misclassification rate while producing bigger trees than the entropy criterion.

7 Conclusions and further research

This paper presents two approaches for labeling decision trees in a way that guarantees the monotonicity of the resulting trees. For each approach two possible labeling functions are proposed and their performance was compared experimentally. The results indicate that dynamic labeling approach produces more accurate trees than the static labeling for trees of the same size.

The experiments used the left-depth-first search strategy for generating the trees. The obvious next step in this direction of research is to investigate whether the results also apply for the right-depth-first strategy.

The paper further investigates the performance of two splitting criteria from the literature - the classical entropy criterion and the more 'monotonicity-oriented' number of conflicts criterion. Although for monotone samples the performance of none of the criteria is systematically better, the experimental results confirm the intuition that with the increase of monotonicity noise the entropy criterion produces smaller trees while the conflicts criterion generates more accurate trees.

incons	entropy		num conflicts	
pairs	miscl	nodes	miscl	nodes
0	52	53	47	83
19	90	293	74	171
55	196	311	103	373
86	218	421	172	501
111	225	469	176	521
118	224	473	186	609
0	1516	95	1522	211
2	1500	115	1520	303
21	2326	561	1789	397
28	2496	593	1773	583
31	2366	605	1737	589
40	2432	963	1763	639
42	2586	1787	1831	1069
0	1520	87	1752	131
2	1512	157	1812	125
7	1673	747	1850	317
39	3052	3655	2692	3497
64	3607	3719	3003	3829
66	3685	3353	3327	4339
69	3798	3751	3322	4437
0	1277	163	1718	195
39	3035	2093	3607	3945
59	3518	3755	4283	5645
74	4577	2969	4369	5737
78	4603	3137	4553	5863
81	4547	3125	4504	5887
83	4547	3135	4466	5861

Table 4: Experimental data on the splitting criteria on non-monotone samples

These results point at one potential direction for further research - is it possible to combine the good properties of the two criteria in a new criterion that both strives at generating smaller trees and reducing the inconsistencies in the nodes in order to produce trees that fit better monotone data and better handle monotonicity noise.

References

A. Ben-David. 1995. Monotonicity maintenance in information-theoretic machine learning algorithms. *Machine Learning*, 19:29–43.

J. C. Bioch and V. Popova. 2000. Rough sets and ordinal classification. *LNAI, Proceedings of the 11th Int. Conf. on Algorithmic Learning Theory (ALT'2000), Sydney, Springer-Verlag*, pages 291–305.

J. C. Bioch and V. Popova. 2002. Monotone classification and noisy data. Technical Report ERS-2002-53-LIS, Dept. of Computer Science, Erasmus University Rotterdam, http://www.erim.nl.

J. C. Bioch and R. Potharst. 1997. Decision trees for monotone classification. *K. van Marcke and W. Daelmans (eds), Proceedings of the Dutch Artificial Conference on Artificial Intelligence (NAIC'97), Antwerp*, pages 361–369.

J. C. Bioch. 1998. Dualization, decision lists and identification of monotone discrete functions. *Annals of Mathematics and Artificial Intelligence*, 24:69–91.

C. L. Blake and C. J. Mertz. 1998. Uci repository of machine learning databases. Irvine, CA: University of California, Department of Information and Computer Science [http://www.ics.uci.edu/ mlearn/ MLRepository.html].

K. Cao-Van and B. De Baets. 2002. Growing decision trees in an ordinal setting. *submitted to International Journal of Intelligent Systems.*

Y. Crama, P. L. Hammer, and T. Ibaraki. 1988. Cause-effect relationships and partially defined boolean functions. *Annals of Operations Research*, 16:299–326.

S. Greco, B. Matarazzo, and R. Slowinski. 1998. A new rough set approach to evaluation of bankruptcy risk. *C. Zopounidis (ed.), Operational Tools in the Management of Financial Risks, Kluwer, Dordrecht*, pages 121–136.

K. Makino, T. Suda, K. Yano, and T. Ibaraki. 1996. Data analysis by positive decision trees. *Proceedings of International Symposium on Cooperative Database Systems for Advanced Applications (CODAS), Kyoto*, pages 282–289.

R. Potharst and J. C. Bioch. 2000. Decision trees for ordinal classification. *Intelligent Data Analysis*, 4:1–15.

W. van Eikeren. 2002. Monotone decision trees and stacked generalization. Master's thesis, Erasmus University Rorrerdam.

S. Warshall. 1962. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12.

Induction of Supermodels

Hendrik Blockeel
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
hendrik.blockeel@cs.kuleuven.ac.be

Abstract

It is rarely the case in machine learning and data mining projects that only a single inductive process is run. Often, experiments are performed with a number of different algorithms, different settings for a single algorithm, different feature sets, and so on. While most research focuses on algorithms that induce a single model, given this practice it would be useful to look for algorithms that induce a kind of modelgenerator or "super"-model, from which multiple concrete models can efficiently be generated (this is a kind of partial evaluation). Thus one can avoid executing computationally heavy induction processes many times in very similar contexts. The process is similar to partial evaluation. This paper discusses some possibilities and challenges for research on supermodels.

1 Introduction

A lot of machine learning research is devoted to induction of models (theories, hypotheses) from data. In many cases the user is interested in predictive models, which can be used to predict properties of new instances (for instance, their class, in the case of classification).

In the context of predictive induction there is often a clear quality criterion, such as predictive accuracy, expected misclassification cost, mean squared error, etc. The user is interested in finding a predictive model from a set of data that performs as well as possible according to the chosen criterion.

Now one problem with machine learning algorithms is that their performance is often influenced by a set of parameters. In many cases it is unclear which parameter values will yield the best model, so the user just runs the algorithm several times in a row, varying the parameters in search for optimal parameter values, and stopping when no further improvements to the model are found.

This approach can be automated to some extent. One then wraps a script around a machine learning algorithm; all the script does is execute a loop that consists of adapting parameter values, running the learning algorithm, record the performance of the resulting model, and based on this performance exit the loop or re-iterate. Such an approach is called a wrapper approach (Kohavi, 1995).

Wrapper approaches have as main advantage that they tend to yield close-to-optimal models even when the result of the induction process heavily depends on the parameter values, and this without the user having to spend any effort tuning these values. A potential danger is that they are more prone to overfitting than building a single model with fixed parameters, but as long as the model validation is done carefully this is not that much of a problem.

A more serious problem with wrappers is that they strongly increase the time needed for the induction process. Even if a single run of the machine learning algorithm takes very little time, a wrapper may easily run that algorithm hundreds of times (note that if for instance tenfold cross-validation is used to determine the expected performance of a model built with certain parameter values, the algorithm already needs to be run ten times in each single iteration), and a more extensive search for parameter values may require many more runs. Obviously such approaches do not scale very well to large data sets, unless specific adaptations are made to the algorithms.

It seems, however, that these problems of computational complexity can be avoided in some cases. The observation that lies at the basis of this statement, is that in all these runs of the same algorithm on the same data set, many computations are exactly the same. One could ask whether it would not be possible to induce a kind of data structure that is independent of specific parameter settings, but from which specific models can be derived efficiently by filling in the values for these settings. We call such a structure a *supermodel*. We will see in the remainder of this text that this is indeed possible. In fact, certain already existing algorithms can be viewed as instances of this approach. The main contribution of this paper is that it puts these approaches into a new perspective and presents arguments for doing more research in this direction.

The remainder of this paper is structured as follows. In Section 2 we describe in more detail the principle of induction of supermodels. In Section 3 we relate the approach to existing approaches in machine learning and data mining. We list a number of opportunities for further research in Section 4 and discuss the expected advantages and current challenges of the supermodel approach in Section 5. We summarize by giving a generic approach to construction of supermodel algorithms in Section 6, then we conclude in Section 7.

2 Induction of Supermodels: Principle

The principle of induction of supermodels can be described as follows. Consider a machine learning or data mining algorithm that from a data set D induces a model M. The behaviour of the algorithm is influenced by certain parameters P. Thus, the algorithm can be described as a function that maps a data set and a set of parameter values onto a model. We call this function *induce*. In pseudo-code we could write the induction process as

```
$M$ := induce($D$, $P$)
```

A supermodel should have the property that it can be induced from the data without the parameters P, and that a specific model can be deduced from it by filling in values for P. Hence we write

```
$SM$ := induce_sm($D$)
$M$ := deduce($SM$,$P$)
```

Note the similarity of the above process to partial evaluation (Jones et al., 1993). The theoretical existence of supermodels follows immediately from the *S-m-n* theorem (Kleene, 1952). The practical computability of SM is of course another issue.

Assuming P is not a single parameter but a set of parameters, the above description of induction of supermodels can be made more flexible by distinguishing between that subset of the parameters that can be postponed to the deduce phase (i.e. which do not take part in the partial evaluation), and those that cannot. If we call these subsets P_1 and P_2 respectively, then the normal induction process is described as

```
$M$ := induce($D$, $P_1$, $P_2$)
```

and is equivalent to the following two-step process of inducing a supermodel and deriving a model from it:

```
$SM$ := induce_sm($D$, $P_1$)
$M$ := deduce($SM$,$P_2$)
```

Now the practical computability of SM is less of a concern. One simply defines P_1 and P_2 in such a way that the supermodel is computable. Obviously the usefulness of the technique increases with the size of P_2.

Now what does the two-step process buy us? As mentioned previously, the idea is that conceptually we wish to run the induction algorithm many times with different parameter values. If we wish to change only values for P_2, then in the two step process we need only run the *deduce* function multiple times. Note that this function is independent of the data set, so the complexity issues related to the data set size and the search through parameter space are decoupled. More specifically, if p combinations of parameter values are tried out during the parameter search, the complexity of both *induce* and *induce_sm* is $O(f(N))$ with N the size of the data set, and the complexity of deducing a single model from the supermodel is c, then the complexity of the whole process becomes $O(f(N) + pc)$ instead of $O(pf(N))$.

Of course, in order to obtain real gains from the two-step process for reasonably sized data sets and parameter spaces, the computational complexity of *induce_sm* should not be excessively higher than that of *induce* and the complexity of *deduce* should be low.

3 Concrete Examples

We have already mentioned the relationship of our technique with partial evaluation. In fact there exists a lot of work in data mining (more than in machine learning, possibly because the issue of scalability towards very large data sets has received more attention in that domain) that is related to our principle of supermodel induction.

This work can be considered to go back to Apriori (Agrawal et al., 1993). Apriori is an algorithm for finding association rules. These are rules of the form $A_1 \ldots A_n \rightarrow B$ where A_i and B are boolean attributes (called *items*) with semantics "if attributes $A_1, \ldots, A_n$ have the value *true*, then B is probably *true* too". Such rules are said to hold with a certain confidence (proportion of cases where B is true, out of all cases where $A_1, \ldots, A_n$ are true) and support (proportion of cases in the whole database where $A_1, \ldots, A_n, B$ are all true). The task is to find the set of all association rules with support and confidence above certain thresholds *min_support* and *min_confidence*.

Finding such a set is a non-trivial task. The approach of Apriori is to compute *a priori* some statistics on frequent itemsets and store these statistics in a database, so that afterwards association rules can efficiently be derived from them. In our terminology, the set of itemset counts is a supermodel that is dependent on a parameter *min_support2*, and a single model is then a set of association rules with parameters *min_support* ($\geq$ *min_support2*) and *min_confidence* that contains all association rules with support at least *min_support* and confidence at least *min_confidence*. Variants include approaches where syntactical constraints on the rules are added to either P_1 (in order to make the itemset discovery process more efficient, e.g., Garofalakis et al., 1999) or P_2, e.g., query languages for data mining (De Raedt, 2000).

Imielinski and Mannila's (1996) seminal paper on inductive databases argues for considering the results of data mining as data objects in themselves that can be stored in a database and consecutively queried. This can be seen as a generalization of the Apriori approach. Nevertheless, most work that has followed up on that paper seems to have focused on the association rule context, although the ideas have a much broader application potential.

The notion of *sufficient statistics* is also relevant. The idea is that from a data set a kind of summary is produced that contains sufficient information to perform a certain type of computations afterwards. Moore and Lee (1998) describe the use of AD-trees for storing sufficient statistics for building trees, rules, bayesian nets, ... (This approach is quite similar to counting frequent itemsets, but has certain advantages over the latter). In a related vein, recently a number of approaches have been proposed where a data set is summarized by a probabilistic model that makes it possible to approximate the joint probability distribution; we mention Pavlov et al. (2001) who perform a comparative study of several such approaches, and Getoor et al. (2002) who present a similar approach for multiple relations in a relational database.

In the context of decision trees, recent work on efficient cross-validation (Blockeel and Struyf, 2002) can be seen as presenting instances of the supermodel approach proposed here. In the cross-validation context a structure is built that contains information on the different trees resulting from the different folds. Also some of Blockeel and Struyf's (2001) Frankenstein classifiers can be considered supermodels: multiple trees are derived from a single tree that each are optimal under different conditions, more specifically, in different areas of ROC space (Provost and Fawcett, 1998).

While all this work implicitly or explicitly promotes an approach similar to the one we propose here, it would appear that the idea in its full generality remains little explored. For instance, if we consider a spectrum of supermodels, with supermodels close to the data (e.g., probabilistic models) on the one side, and supermodels close to the final models (e.g., cross-validation forests) on the other side, there is a large gap in between these two extremes.

4 Opportunities

We have mentioned that approaches similar to our supermodel approach exist mainly in the "association rules" context. There are clearly many other opportunities. Let us have a closer look at decision tree induction.

Decision tree induction (Quinlan, 1993; Breiman et al., 1984) typically makes use of a heuristic such as information gain (ratio) that can be computed entirely from class distributions, and these can themselves be computed from attribute-value counts.[1] Suppose for instance that in a certain node three levels below the root all examples with values $[x_1, x_2, x_3]$ for attributes $[X_1, X_2, X_3]$ are gathered, and we need to compute the information gain of attribute X_4. This information gain can be computed from the joint frequency distribution of $[X_4, Y]$ within the current node. Assuming that for all values x_{4i} and y_j for the attribute X_4 and target attribute Y the frequency of the tuple $[x_1, x_2, x_3, x_{4i}, y_j]$ has been computed in advance, the information gain can be computed from these frequencies.

In general, if attribute-value frequencies are computed up to level N using an Apriori-like approach, then the first $N-1$ levels of a decision tree can be computed from these without ever looking at the data. Of course counting the attribute-value combinations may take more work than inducing a single decision tree, but if several trees may have to be built (e.g., for different target attributes), or if the sets have been computed anyway (e.g., with the aim of deriving association rules), then this approach may pay off.

There remain some possible problems with this approach. The Apriori algorithm typically encounters complexity problems when it is run with a very low support threshold, so that many combinations are "frequent". Increasing the support threshold would means that the quality of certain kinds of splits can only be estimated. More specifically, if the treshold is t, then an exact count of the number of instances belonging to a certain class is only available if there are at least t of them. If for only one class that number is missing, it can still be computed from the other numbers. Otherwise, a pessimistic estimate of entropy can be made by assuming classes for which no information is available to be equally frequent. It is unclear how much such estimates would influence the quality of the induced tree.

Thus, apart from a number of issues that are not settled yet, it seems safe to say that frequent itemset (or attribute-value) counts deserve consideration as a possible supermodel for decision trees. This is even more the case for AD-trees

We have already mentioned the work on efficient cross-validation,where a kind of forest structures are induced from which the specific decision trees obtained in different folds of a cross-validation are efficiently computable. These are also supermodels for trees. For induction of such forests one needs to know the partitioning of the data used for the cross-validation. In fact, if we assume that the data are extended with a single attribute indicating for each example in which fold it is left out of the training set, and perform the attribute-value counting with this attribute included, then all necessary information for inducing such forests is available. In other words, attribute-value counts can also be considered supermodels for these forests.

A maximal, unpruned tree can be seen as a supermodel for all pruned versions of it. In those cases where the stopping and pruning methods are monotonically influenced by a single numerical parameter, it suffices to add to each node of the maximal tree a single threshold that indicates for what values of the parameter the subtree starting in that node would be collapsed to a leaf. This means parameters of the stopping and pruning criteria are good candidates to belong to the P_2 set of parameters.

It is clear from this that we can define a *hierarchy of supermodels* for trees. Based on the above the hierarchy could look as follows. At the highest level we have attribute-value counts, which are closest to the data and from which a very general set of models could be computed. Next is a forest of unpruned trees that for a given cross-validation partitioning summarizes the unpruned trees that would be obtained during the cross-validation. The different unpruned trees are at the next level, and at the lowest level we obtain concrete pruned trees.

In our pseudocode scheme, the process

[1]These are similar to itemset counts as computed by Apriori, except that the "items" are now specific values for an attribute.

roughly looks like this:

```
$Counts$ := induce_sm($D$, $min\_support$)
$Forest$ := deduce$_1$($Counts$, $various\_param$)
$UTree$ := deduce$_2$($Forest$, $fold$)
$Tree$ := deduce$_3$($UTree$,$pruning\_threshold$)
```

where *various_param* contains all parameters not mentioned elsewhere, such as which attribute is the target attribute, possibly the heuristic to be used, ...

Clearly, in those cases where multiple cross-validations are to be performed with varying pruning parameter and target attributes, a lot of efficiency can be gained by following the hierarchical supermodel approach. Note that what we have presented here is only one possible hierarchy, not necessarily the most useful one.

We end our discussion of supermodel-based tree induction here. It will be clear by now that a more complete discussion of this approach will require a much more extensive investigation than is the scope of the current paper.

Besides decision tree induction, there are of course other techniques that can benefit from the supermodel approach. First of all, note that rule induction is governed by largely the same principles as decision tree induction: it typically follows a greedy search, estimating rule quality based on class distributions in subsets of examples. The remarks given for decision tree induction apply also in this context.

Genetic algorithms are computationally burdened by the need to perform very many evaluations of hypotheses. These evaluations may be sped up significantly if statistical information such as the one available in attribute-value counts can be used.

Probabilistic approaches heavily rely on frequency distributions; again attribute-value counts are a reasonable supermodel to start from. Other supermodels include those discussed in (Moore and Lee, 1998; Pavlov et al., 2001; Getoor et al., 2002). On the other hand, for models such as neural networks it is much less clear what a supermodel should look like.

5 Promises and Problems

We briefly list a number of advantages that can be expected from the supermodel approach, as well as a number of current challenges that we face before we can implement the approach.

5.1 Promises

An important advantage of the supermodel approach is that it will make *wrapper approaches* much more feasible. In fact, if the *deduce* function is sufficiently efficient, then an extensive search through model space becomes possible, even in those cases where the size of the data set precludes an extensive search through the model space. Note that if n models are to be investigated, the total time needed for inducing these models is nT_{induce} for the normal approach and $T_{induce\_sm} + nT_{deduce}$ for the supermodel approach, so that a speedup close to n or T_{induce}/T_{deduce} (whichever is smallest) can be obtained. Both numbers may be quite large in practice.

We do not present experimental results in this paper, but some existing results support our claim that significant efficiency gains are possible with the supermodel approach. For instance, Blockeel and Struyf (2002) report speedup factors of about 3 for bagging and close to n for certain n-fold cross-validation experiments, with n up to 20. For wrapper approaches even better speedups can be expected, as n (the number of different models considered) is typically larger there. Moore and Lee (1998) report speedup factors of over 4000 in specific cases, by using AD-trees.

Another advantage of the supermodel approach is that it offers more *flexibility*, as full details on the models to induce need not be available in advance. Most of the computational effort can happen in advance and when full details are available, the concrete models can quickly be obtained. This is nicely illustrated by the Apriori approach: one does not need to know which kind of association rules one is interested in to compute itemset counts. Once the user expresses interest in association rules with item A in the head and a certain minimal confidence, these can very quickly be computed.

5.2 Problems

It is currently unclear in many cases to what extent the supermodel approach can be followed. Potential problems can be identified with respect to inducing supermodels, representing them, and deducing concrete models from them. In many cases a concrete view on how this should be done does not exist yet.

Let us first focus on representation issues. A supermodel contains information on many different concrete models. A trivial representation of a supermodel is a set of couples (P, M) where P is a combination of parameter values and M is the model that the supermodel is instantiated to if P is filled in. It is obvious that when millions of combinations of parameter values exist, this representation is infeasible because of memory limitations. On the other hand, the *deduce* function would become a simple lookup.

Supermodels should be designed in such a way that they concisely summarize properties of many different models. In the mentioned work on efficient decision tree cross-validation, for instance, the forests obtained are not impressive but acceptable. In the worst case, they take about as much space as the different trees themselves. This is typically not a problem in practice as the number of cross-validation folds is typically in the order of 10.

Attribute-value counts may also consume a large amount of memory; as mentioned before the "minimal support" parameter may have to be relatively high for them to be manageable. Note that the option of storing this information in a database on disk alleviates the problem of memory-intensive representations somewhat. The same could of course be done for any supermodel.

As supermodels contain information for many different models, their induction may also require much more computational effort. For instance, computing attribute-value counts requires more work than building a single decision tree. On the other hand, in the case of decision tree cross-validation, the supermodel can in some cases be induced with barely any overhead over induction of a single model (Blockeel and Struyf, 2002).

Obviously the amount of overhead caused by the supermodel approach that is considered acceptable will depend on how many specific models will be derived from it later, and on how efficiently this deductive derivation can be performed. In the supermodel hierarchy for decision trees discussed above, all deductive steps are more efficient than the single inductive step, although $deduce_1$ is computationally heavier than the other deductive steps.

6 A Supermodel Building Methodology

The following could be considered a generic approach to building algorithms for the induction of supermodels:

```
Given an algorithm A:
- list its parameters P
- study which ones can easily be postponed
  to the deductive phase (P2); P1 = P - P2
- design a representation for supermodels
  based on P1
- write an algorithm that builds supermodels
  using parameters P1
- write an algorithm that deduces models
  from supermodels using P2
```

The choice of P_1 and P_2 will take into account the considerations mentioned above. In general, a trade-off has to be found between the complexity of the inductive and deductive steps and the feasibility of the supermodel representation. There may be multiple "optimal" points for splitting P into $P1$ and $P2$, in which case it may be worthwhile to consider a supermodel hierarchy.

Clearly the major opportunities for research into supermodels are in the investigation of which part of a set of parameters can be postponed, and how easily this can be done, taking into consideration the issues mentioned above.

7 Conclusions

This paper introduces the concept of induction of supermodels as a general approach to machine learning and data mining. The approach is related to ideas such as inductive databases, construction of sufficient statistics, etc. and encompasses several existing approaches as specific instances.

Induction of supermodels is very promising in the sense that it may significantly redefine our views of popular techniques such as wrapper approaches. It would enable one to perform a search where each node in the search space is generated efficiently and deductively, instead of inductively and at high computational cost. Because of this it may strongly reduce the need for manual parameter tuning by users of machine learning or data mining algorithms. Even though users might be guided by some intuition about which parameter values are promising, the additional efficiency brought about by the supermodel approach may enable such extensive searches that a brute force search for good parameter values becomes feasible.

The idea of supermodel induction is currently still quite immature. Although specific examples of the approach exist and convincingly show the potential of the approach, in many other cases it is not clear what a supermodel could look like. A few suggestions are given in this paper for the case of decision trees. In general there is a certain trade-off between representation complexity and the complexity of the inductive and deductive steps, and some kind of optimum should be searched for. Clearly, there are ample opportunities for further research on this. Finally, an intriguing question in this respect is how existing results in partial evaluation might help with the discovery of good supermodels.

Acknowledgements

The author is a post-doctoral fellow of the Fund of Scientific Research of Flanders, Belgium.

References

R. Agrawal, T. Imielinski, and A. Swami. 1993. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 207–216, Washington, D.C., USA, May. ACM.

H. Blockeel and J. Struyf. 2001. Frankenstein classifiers: Some experiments. In V. Hoste and G. De Pauw, editors, *Proceedings of the Eleventh Belgian-Dutch Conference on Machine Learning*, pages 5–12.

H. Blockeel and J. Struyf. 2002. Efficient algorithms for decision tree cross-validation. *Journal of Machine Learning Research.* In press.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. 1984. *Classification and Regression Trees.* Wadsworth, Belmont.

L. De Raedt. 2000. A logical database mining query language. In J. Cussens and A. Frisch, editors, *Proceedings of the Tenth International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 78–92. Springer-Verlag.

M. N. Garofalakis, R. Rastogi, and K. Shim. 1999. SPIRIT: Sequential pattern mining with regular expression constraints. In *The VLDB Journal*, pages 223–234.

L. Getoor, D. Koller, and B. Taskar. 2002. Statistical models for relational data. In *Proc. of the KDD-02 Workshop on Multi-Relational Data Mining.*

T. Imielinski and H. Mannila. 1996. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64.

N.D. Jones, C.K. Gomard, and P. Sestoft. 1993. *Partial Evaluation and Automatic Program Generation.* Prentice Hall.

S. Kleene. 1952. *Introduction to metamathematics.* D. van Nostrand, Princeton, New Jersey.

Ron Kohavi. 1995. *Wrappers for Performance Enhancement and Oblivious Decision Graphs.* Ph.D. thesis, Computer Science department.

A. W. Moore and M. S. Lee. 1998. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91.

D. Pavlov, H. Mannila, and P. Smyth. 2001. Beyond independence: Probabilistic models for query approximation on binary transaction data. ICS TR-01-09, Information and Computer Science Department, UC Irvine.

F. Provost and T. Fawcett. 1998. Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 43–48. AAAI Press.

J. R. Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann series in machine learning. Morgan Kaufmann.

Improving the efficiency of ILP systems using an Incremental Language Level Search

Rui Camacho
LIACC, Rua do Campo Alegre, 823, 4150 Porto, Portugal
FEUP, Rua Dr Roberto Frias, 4200-465 Porto, Portugal
rcamacho@fe.up.pt
`http://www.fe.up.pt/~rcamacho`

Abstract

We propose and evaluate a technique to improve the efficiency of an ILP system. The technique avoids the generation of useless hypotheses. It defines a language bias coupled with a search strategy and is called Incremental Language Level Search (ILLS). The techniques have been encoded in the ILP system IndLog. The proposal leads to substantial efficiency improvements in a set of ILP datasets referenced on the literature.

1 Introduction

Inductive Logic Programming (ILP) has achieved considerable success in a wide range of domains. It is recognised however that efficiency is a major obstacle to the use of ILP systems in applications requiring large amounts of data. Relational Data Mining applications are an example where efficiency is an important issue. In this paper we address the problem of efficiency in ILP systems by proposing a technique to improve it. Our proposal is called Incremental Language Level Search (ILLS) and is a language bias coupled with a search strategy. It is a correct language bias as defined by De Raedt (De Raedt, 1992). ILLS imposes a partition (and a meta-order) in the subsumption lattice (the search space of hypothesis). Language levels are defined based on the repetitions of predicate symbols in the clauses and since in most of the target theories of common applications the repetitions of literals in the clauses is very small or non - existent the most likely clauses are investigated first. A very useful admissible pruning rule is a corollary of this technique. Our proposal is a general technique that can be implemented in any ILP system to improve performance.

A typical ILP system carries out a search through an ordered hypothesis space. During the search hypotheses are generated and their quality estimated against the given examples. Improvements in efficiency may be obtained in avoiding to generate useless hypothesis or/and improving their evaluation.

Avoiding to generate useless hypotheses may be achieved with the specification of language bias limiting therefore the size of the search space (Nédellec et al., 1996). One my consider a procedure where there is a sequence of "space boundaries" that are increased if the system does not find an acceptable hypothesis within the current space. This technique is generally called *shift of bias* (De Raedt, 1992). Our ILLS proposal is a an example of a *shift of bias* technique. Another approach considers the study of refinement operators that allow to efficiently navigate through a hypothesis space (van der Laag and Nienhuys-Cheng, 1998).

The problem of efficient testing of candidate hypotheses has been tackled by two lines of research: improving sequential execution of an ILP system and; improving hypothesis evaluation by a parallel or a distributed execution. We first consider the sequential execution.

Sebag and Rouveirol (Srinivasan, 1999; Sebag and Rouveirol, 1997) (and later Botta *et al.* (Botta et al., 1999)) suggest the use of stochastic matching to speedup hypothesis evaluation. These approaches reduce the evaluation effort at the cost of being correct only with high probability. Another approach is proposed by Costa *et al.*. They carried out a study on exact transformations of queries when evaluating hypotheses (see (Santos Costa et al., 2000) and (Costa et al., 2002 to appear)). In (Santos Costa et al., 2000), the authors illustrated that query execution was a very high percentage of total running time. Some exact transformations on the generated hypotheses were proposed that make them more efficient to execute on a Prolog engine.

There is a great number of common literals in a clause and its refinements and even with the refinements of its refinements. A *query pack* technique (Blockeel et al., 2002) takes advantage of the great number of shared literals among a clause and its refinements. Queries with a great lot of common computations are grouped in sets called *query packs* that are executed in a way to avoid redundant computations.

Lazy evaluation of examples (Camacho, 2000) as used in IndLog produces considerable speedups. The technique consists in avoiding or postponing the evaluation of each hypothesis against all the examples. Two kinds of lazy evaluation of examples is distinguished: negatives and positives. The rationale supporting lazy evaluation of negative examples is as follows. The purpose of evaluating a hypothesis on the negative examples is to determine if it is consistent, that is to find out if it covers less than "*noise* number" of negative examples. If the hypothesis covers more than "*noise* number" of negative examples it has to be refined (it is too general). In lazy evaluation of negative examples the system stops as soon as the clause covers "*noise* number" + 1 negative examples since that is enough to determine the lack of consistency of the clause. In the lazy evaluation of positives the system stops the evaluation of the clause as soon as it covers 1 more than the best positive cover so far. That number is enough to avoid pruning it away. The complete positive coverage is done only for consistent clauses. In most applications the negative examples overwhelm the positives and lazy evaluation of negatives may be very useful. These techniques prevent the use of negatives or positives counts in the heuristic functions but in most cases the speedups of using these techniques overcome the use of more powerful heuristics.

One last line of research followed by Dehasp and De Raedt (Dehaspe and De Raedt, 1995) and by Ohawada *et al.* (Ohwada et al., 2000), and enphasised by David Page (Page, 2000), is the parallel or distributed execution of an ILP system. It is also a very promising direction to overcome the efficiency bottleneck. Yu Wang (Wang, 2000) claim to have achieved super-linear speedup in their implementation. Graham *et al.* (Graham et al., 2000) report a linear speedup in their implementation. Work in this line of research has been essentially in distributing parts of the search space among the processors. One alternative would have a master slave architecture where each slave gets just one set of the examples partition. The master then sends each hypothesis to the slaves for evaluation on the subsets of the examples. This approach makes Data Mining applications having millions of examples amenable for ILP systems since each slave processor would process "just" a subset of the examples, requiring therefore less memory resources. A challenging approach would be to have a parallel execution of an ILP engine on top of a parallel Prolog engine. In some datasets it is the large number of examples that requires parallelization of the ILP engine whereas in some others evaluating each hypothesis is hard. In the later case a parallel Prolog engine could be very useful. Combing the workload of the ILP and Prolog engine is a very hard problem.

Most of the techniques we just referred are applicable in a parallel or distributed execution setting and therefore substantial improvements on efficiency may be gained through the combination of the results of all of these lines of research.

The evaluation of our proposal was done using the *IndLog* system (Camacho, 2000). *IndLog* is an example of an empirical ILP system, according to the classification proposed by De Raedt (1992). It is based on the *Mode Directed Inverse Entailment* Muggleton (1995). *IndLog* can handle non-ground background knowledge, can use nondeterminate predicates, uses a strongly typed language and makes use of explicit bias declarations such as mode, type and determination declarations. It has admissible pruning strategies and can handle numerical and imperfect data. To handle numerical data IndLog has the possibility of lazy evaluation of literals (Srinivasan and Camacho, 1999) and performs a user-defined cost search. To handle large amounts of data IndLog may perform lazy evaluation of the positive and negative examples. For a complete description of *IndLog* refer to (Camacho, 2000).

The structure of the rest of the paper is as follows. In Section 2 we present the Incremental Language Level Search strategy. The experiments that empirically evaluate our proposal are presented in Section 3. The last section draws the conclusions.

2 Incremental Language Level Search

IndLog uses a particular strategy of searching through the hypothesis space that is based on a structural organization imposed on this space. We now present the hypothesis space organization by language levels and then describe the search procedure that takes advantage of such layered organization.

Let $\mathcal{D}$ be the set of definite clauses and $\mathcal{L}_i$ a set of clauses of level i. Consider a partition[1] of $\mathcal{D} = \bigcup_{i=0}^{\infty} \mathcal{L}_i$. Each subset $\mathcal{L}_i$ is called a *language level* and is defined as:

Definition 2.1 Language level

$\mathcal{L}_i$ = {*clause* | *maximum number of occurrences of a predicate symbol in the body of clause is i*}

A clause belongs to the language level i if it has a predicate symbol P that occurs i times in the body of that clause. No other predicate symbol Q ($\neq$ P) of that clause occurs more than i times.

The maximum number of occurrences of predicate symbols in the body of the clauses determines to which partition the clause belongs. The language $\mathcal{L}_0$ is composed of definite clauses with just the head literal. The language $\mathcal{L}_1$ is composed by definite clauses whose literals in the body have no repeated predicate symbols. The language $\mathcal{L}_2$ will contain clauses whose literals in the body have a maximum number of occurrences of the same predicate symbol of two. Table 1 shows some examples of clauses classified according to language level. The language level definition is applicable to both recursive and non-recursive clauses. Clauses belonging to $\mathcal{L}_i$ will have a minimum length of i+1 (including the head literal) and a maximum of $p * i + 1$[2].

illegal(A, B, C, D, E, D)	$\in$	$\mathcal{L}_0$
multiplication(A,B,C) :- dec(A,D), multiplication(D,B,E), plus(E,B,C)	$\in$	$\mathcal{L}_1$
qSort(X,Y) :- part(X,Z,W), qSort(Z,Z1), qSort(W,W1), app(Z1, W1, Y)	$\in$	$\mathcal{L}_2$
fold(imglobuli, A) :- l(A,B), i(50,B,173), a(A,C), i(0,C,1), b(A,D), i(7,D,10)	$\in$	$\mathcal{L}_3$
p(X) :- b(X), a(X,U), a(X,Y), a(X,Z), a(X,W), b(X)	$\in$	$\mathcal{L}_4$

Table 1: Examples of clauses belonging to different language levels.

One very important property of the partitioning by language level is that all clauses in language $\mathcal{L}_{i+1}$ are subsumed by at least one clause in language $\mathcal{L}_i$ as stated in Theorem 2.1. The subsumption relation among language levels is illustrated in Figure 1.

Theorem 2.1 *Let C_j and C_k be clauses and i an integer ($i \geq 0$).*

$$\forall_{C_j} \exists_{C_k} : (C_j \in \mathcal{L}_{i+1}) \wedge (C_k \in \mathcal{L}_i) \rightarrow C_k \preceq C_j$$

Any clause belonging to language level $\mathcal{L}_{i+1}$ is θ-subsumed by at least one clause in language level $\mathcal{L}_i$.

Proof. We prove that for each clause C_{i+1} in $\mathcal{L}_{i+1}$ there is at least one clause C_i in $\mathcal{L}_i$ whose literals are a subset of C_{i+1}. Then we prove that there is a substitution that completes the subsumption relation ($C_i \preceq C_{i+1}$).

If C_{i+1} is in $\mathcal{L}_{i+1}$ then it has at least one predicate symbol that occurs i+1 times in its body. We will call it *p*. Since in each language level *i* all clauses have in their body a maximum of i occurrences of at least one predicate symbol, then necessarily $\mathcal{L}_i$ has a clause (C_i) with just *i* literals in the body having the predicate symbol *p*. Assuming that we found a correct

[1]The ∞ sign in the upper limit of the union represents a value limit smaller than infinity since definite clauses, by definition, have a finite number of literals.

[2]Where p is the maximum number of predicate symbols usable in the body of a clause.

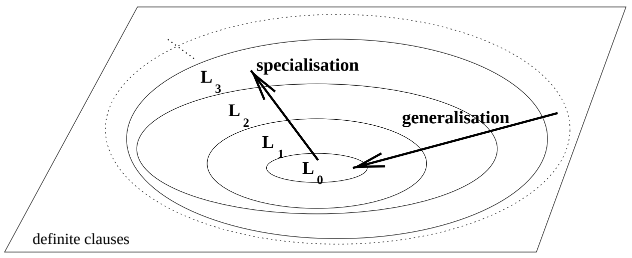


Figure 1: Partition of the set of definite clauses by Language Level.

substitution then clause C_i is a subset of C_{i+1}. To prove the subsumption relation we have to find the assumed substitution. The empty substitution is enough to establish the subsumption relation. We just have to choose C_i such that the variables in its body literals are renaming of the literals p of C_{i+1}. With this choice, all body literals of C_i unify with the literals p of C_{i+1} and therefore $C_i\theta \subseteq C_{i+1}$.
■

The suggested partition by language level applies directly to the subsumption lattice that is traversed by the induction algorithm when constructing the hypothesis clause. The subsumption lattice is subdivided into sub-lattices, each corresponding to a particular language level. The search algorithm using the Incremental Language Level Search is summarised in Algorithm 2.1. As can be seen it involves, at each step of the main cycle, a call to a "traditional" hypothesis induction procedure. At each cycle the "traditional" hypothesis induction procedure may only generate clauses within the current language level. After each cycle the language level is increased. The cycle terminates when one of three conditions is met: i) search constraints C are no longer satisfied; ii) there has been no improvement in coverage during current level or; iii) there are no more clauses to refine because they are already refined or pruned away. If the quality of a hypothesis is its coverage[3] then we may use Theorem 2.2 to establish a stopping condition. The search may be terminated after a language level has been search through and no hypothesis was found that would bring improvements when compared to the hypotheses found in previous levels. Search constraints C typically include the maximum number of clauses constructed and the maximum length of the constructed clauses.

Theorem 2.2 *If there is no hypothesis at language level i that subsumes (covers) more positive examples than the best of positive coverage in levels j (< i), then there will be no better clauses at levels k > i.*

Proof. The proof follows from Theorem 2.1 and from the observation that a clause subsumed by another cannot cover more positive examples than the clause which subsumes it.
■

Algorithm 2.1
Incremental Language Level Search (ILLS)

```
input:
    C                      /* search constraints */
    D                      /* a dataset */
    h = □                  /* the empty clause */
    i = 0                  /* language level counter */
    SearchStrategy         /* a search strategy for the
                              refinement step */
    language = $\mathcal{L}_0$       /* initial language is $\mathcal{L}_0$ */
    improvement = true     /* flag */

output: h                  /* best consistent hypothesis */

  while improvement and satisfying C do

                           /* induce hypotheses */
    h' = genHypothesis(SearchStrategy,language,C,D)

                           /* update */
    if (h = h') then improvement = false
    h = bestOf(h,h')       /* update best hypothesis */
    i = i+1
    language = $\mathcal{L}_i$       /* update language level */

  endwhile
```

Language level is not directly related to i-determinacy[4] as can be seen by the following examples: p(X) :- q(X,Y), r(Y,Z) belongs to level 1 and has i-depth of 2; p(X) :- q(X,Y), q(Y,Z) belongs to level 2 and has i-depth of 2.

An advantage of the search by language levels is that the most probable sub-lattices are searched first. In most of the target theories of common applications the repetitions of predicate symbols in the clauses is very small or non - existent. Support for this statement can

[3]As is common in most ILP systems.

[4]First introduced by Muggleton and Feng (Muggleton and Feng, 1990).

be found in the books by Stephen Muggleton (Muggleton, 1992) and the one by Nada Lavrač (Lavrač and Džeroski, 1994). Those books present some theories induced by ILP systems for *real world* applications. With the exception of some biochemical applications where the atoms positions are used, the repetition of predicate symbols in the body of the clauses rarely surpasses 2.

Varšek and Urbančič (1993) used a language restriction that specifies that only one "attribute = value" test is allowed to appear in the body of the induced rules. This restriction is equivalent to specifying a language level limit of one.

The shift of language bias was already described by De Raedt (De Raedt, 1992). However in ILLS the different languages defined are organised in a subsumption relationship order that as several advantages as the one of Theorem 2.1. In the work by De Raedt there is no such concern with the different languages defined.

3 The experiments

To empirically evaluate our proposals we run IndLog (Camacho, 2000) on several well known datasets. The experiments aim at estimating the efficiency gains when adopting the ILLS. By efficiency gain we mean a reduction in the number of useless hypotheses generated during search. This measure is machine and implementation independent. For each dataset we used in the experiments the final set of induced clause(s) are the same with and without ILLS. Therefore the accuracy of the induced theories does not change by adopting the proposed technique.

Settings

The IndLog V1.0 system was used in the experiments. The datasets used in the experiments are characterised in Table 2 and were downloaded from the Oxford[5] and York [6] Universities Machine Learning repositories.

To evaluate ILLS we run IndLog imposing a limit on the number of clauses constructed during search[7]. The *IndLog* parameter settings used in the experiments are shown in Table A of Appendix A. We then measure the percentage of constructed clauses that are above the Language Level of the target theory of each dataset. This number represents useless clauses that would never been constructed if ILLS were active. In some datasets (like mutagenesis or carcinogenesis) where we do not know the target theory, we tested several values for the maximum language level and for the maximum clause length. Table 3 shows the results of our experiments.

Dataset	positive examples	negative examples	background predicates
carcinogenesis	162	136	38
choline	663	663	32
cyclic	2	2	2
member	20	6	2
mesh	2272	223	29
multiplication	9	15	3
mutagenesis	114	57	18
pyrimidines	1394	1394	244
suramin	7	4	2
train	5	5	10

Table 2: Characterisation of the datasets used in the experiments.

The results indicate that there can be a lot to be gained if the language level of the target theory is 1. The values shown in Table 3 for language level 1 may reach 70 % (multiplication dataset) useless clauses that could be avoided. The ILLS is also recommended if the language level of the target theory is 1 (or small), clause length is large (greater than 4) and the target predicate is determined by a small number of predicates. If there is a small number of predicate symbols for the body literals and the system has to assemble a long clause it is more likely that repetitions will occur and the ILLS may be useful.

It may also be noticed that the gain of using ILLS decreases quite significantly when the target theory is in language level 2 (see the choline and the mesh datasets).

4 Conclusions

In this paper we proposed a general technique that improves the efficiency of an ILP system.

[5]URL:www.comlab.ox.ac.uk/oucl/groups/machlearn/
[6]URL: www.cs.york.ac.uk/mlg/index.html

[7]*IndLog*'s *nodes* parameter.

dataset	max length	max L.L.	useless clauses (%)
carcinogenesis	4	1	29.6
carcinogenesis	6	1	27.6
choline	4	1	9.9
choline	4	2	0.2
choline	6	1	10.2
choline	6	2	0.3
cyclic	3	1	23.8
mutagenesis	4	1	29.4
pyrimidines	4	1	9.6
pyrimidines	6	1	9.9
suramin	4	1	51.1
member	2	1	26.2
mesh	6	2	1.5
multiplication	4	1	70.7
train	4	1	17.4

Table 3: Percentage of useless clauses generated by IndLog when not using ILLS. The parameter settings are shown in Table A of Appendix A.

It is called Incremental Language Level Search (ILLS) and avoids the generation of useless hypotheses. The ILLS is a language bias coupled with a search strategy that imposes a partition (and a meta-order) on the subsumption lattice.

ILLS achieved significant reductions (up to 70.7 %) in the number of useless clauses generated by the induction algorithm.

Acknowledgments

We thank Universidade do Porto and Fundação para a Ciência e Tecnologia that fund our research. We thank the reviewers for their helpful comments.

References

Hendrik Blockeel, Luc Dehaspe, Bart Demoen, , Gerda Janssens, Jan Ramon, and Henk Vandecasteele. 2002. Improving teh efficicency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166.

M. Botta, A. Giordana, L. Saitta, and M. Sebag. 1999. relational learning: hard problems and phase transitions. In *Proc. of the 6th Congress AI*IA, LNAI 1792*, pages 178–189. Springer-Verlag.

R. Camacho. 2000. *Inducing Models of Human Control Skills using Machine Learning Algorithms.* Ph.D. thesis, Department of Electrical Engineering and Computation, Universidade do Porto.

Vítor Costa, Ashwin Srinivasan, Rui Camacho, Hendrik Blockeel, Bart Demoen, , Gerda Janssens, Jan Struyf, Henk Vandecasteele, and Wim Van Laer. 2002 (to appear). Query transformations for improving the efficiency of ilp systems. *Journal of Machine Learning Research.*

Luc De Raedt. 1992. *Interactive Theory Revision: An Inductive Logic Programming Approach.* Academic Press, London, UK.

L. Dehaspe and L. De Raedt. 1995. Parallel inductive logic programming. In *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases.*

J. Graham, David Page, and A. Will. 2000. Parallel inductive logic programming. In *Proceedings of teh Systems, Man and Cybernetics Conference (SMC-2000).*

N. Lavrač and S. Džeroski. 1994. *Inductive Logic Programming: Techniques and Applications.* Ellis Horwood.

S. Muggleton and C. Feng. 1990. Efficient induction of logic programs. In *Proceedings of the 1st Conference on Algorithmic Learning Theory*, pages 368–381. Ohmsma, Tokyo, Japan.

S. Muggleton, editor. 1992. *Inductive Logic Programming.* Academic Press.

S. Muggleton. 1995. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286.

C. Nédellec, C. Rouveirol, H. Adé, F. Bergadano, and B. Tausend. 1996. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, pages 82–103. IOS Press.

Hayato Ohwada, Hiroyuki Nishiyama, and Fumio Mizoguchi. 2000. Concurrent execution of optimal hypothesis search for inverse entailment. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 165–173. Springer-Verlag.

David Page. 2000. ILP: Just do it. In

J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 3–18. Springer-Verlag.

Vítor Santos Costa, Ashwin Srinivasan, and Rui Camacho. 2000. A note on two simple transformations for improving the efficiency of an ILP system. In J. Cussens and A. Frisch, editors, *Proceedings of the 10th International Conference on Inductive Logic Programming*, volume 1866 of *Lecture Notes in Artificial Intelligence*, pages 225–242. Springer-Verlag.

M. Sebag and C. Rouveirol. 1997. Tractable induction and classification in first-order logic via stochastic matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann.

A. Srinivasan and R.C. Camacho. 1999. Numerical reasoning with an ILP program capable of lazy evaluation and customised search. *Journal of Logic Programming*, 40(2,3):185–214.

A. Srinivasan. 1999. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123.

Patrick van der Laag and Shan-Hwei Nienhuys-Cheng. 1998. Completeness and properness of refinement operators. *Journal of Logic Programming*, 34(3):201–226, March.

Alen Varšek, Tanja Urbančič, and Bogdan Filipič. 1993. Genetic algorithms in controller design and tuning. *IEEE Transactions on Systems, Man and Cybernetics*, 23(5):1330–1339.

Yu Wang. 2000. Parallel inductive logic in data mining. Technical report, Department of Computing and Information Science, Queen's University, Kingston, Canada.

A IndLog Settings

Table A shows the settings of the most relevant *IndLog* parameters. Parameter *nodes* specifies the maximum number of the subsumption lattice nodes visited during the inductive search. *Noise* specifies the maximum number of negative examples a clause may cover in order to be accepted. *Min cover* specifies the minimum number of positives examples a clause has to cover in order to be accepted (avoids the generation of clauses with a very poor coverage, possibly overfitting the data). *i-depth* specifies the maximum depth of a literal with respect to the head literal of that clause. A literal that has an input argument that is connected to the head is said to be at i-depth 1. If an input argument of a literal is connected to an output argument of a literal of i-depth j then that literal is at i-depth of $j+1$.

dataset	max nodes	noise	i-depth	min cover
carcinogenesis	1000	15	4	5
choline	200	10	4	20
cyclic	100	0	2	1
mutagenesis	1000	5	2	10
pyrimidines	200	20	4	50
suramin	200	0	4	1
member	100	0	1	1
mesh	1000	5	5	10
multiplication	1000	0	3	1
train	1000	0	2	1

Table 4: IndLog parameter settings used in the experiments. The heuristic was set to length (breadth-first search).

Non-Universal Suffrage Selection Operators Favor Population Diversity in Genetic Algorithms

Federico Divina, Maarten Keijzer, Elena Marchiori
Vrije Universiteit
Amsterdam
{*divina,mkeijzer,elena*}*@cs.vu.nl*

Abstract

State-of-the-art concept learning systems based on genetic algorithms evolve a redundant population of individuals, where an individual is a partial solution that covers some instances of the learning set. In this context, it is fundamental that the population be diverse and that as many instances as possible be covered. The universal suffrage selection (US) operator is a powerful selection mechanism that addresses these two requirements. In this paper we compare experimentally the US operator with two variants of this operator that incorporate information on the hardness of the instances to be covered during the evolutionary process. The results of the experiments indicate that a strong search bias on non-covered instances is most beneficial for promoting population diversity, while maintaining the crucial property of US selection of favoring coverage of many instances.

1 Introduction

Learning from examples can be viewed as a search problem in the space of all possible hypotheses (Mitchell, 1982). Given a description language used to express possible hypotheses, a background knowledge, a set of positive examples, and a set of negative examples, one has to find a hypothesis which covers all positive examples and none of the negative ones (cf. (Kubat et al., 1998; Mitchell, 1997)). This problem is NP-hard even if the language to represent hypotheses is propositional logic.

Powerful learners like FOIL (Quinlan, 1990) and Progol (Muggleton, 1995) adopt a progressive coverage approach. Starting from a training set containing all positive and negative examples, a (if-then) rule is constructed which covers some of the positive examples; the covered positive examples are removed from the training set and the learner continues with the search for the next rule. When the process terminates (after a maximum number of iterations or when all positive examples have been covered), the resulting set of rules is reviewed and redundant rules are eliminated.

Concept learners based on genetic algorithms (GA) act on more rules at the same time. Systems like GIL (Janikow, 1993), GLPS (Leung and Wong, 1995) and STEPS (Kennedy and Giraud-Carrier, 1999) use an encoding where a chromosome represents a set of rules. Other recent state-of-the-art GA based systems like SIA01 (Augier et al., 1995), REGAL (Giordana and Neri, 1996), G-NET (Anglano et al., 1998), DOGMA (Hekanaho, 1998), and ECL (Divina and Marchiori, 2002) evolve a population representing a redundant set of partial solutions, where a chromosome encodes a rule (typically a Horn clause). Both approaches present advantages and drawbacks. Encoding a whole hypothesis in each chromosome allows an easier control of the genetic search but introduces a large redundancy, that can lead to populations hard to manage and to individuals of enormous size. Encoding one clause in each chromosome allows for co-operation and competition between different clauses, but requires the use of suitable strategies for promoting population diversity and coverage of many positive examples.

REGAL and its descendant G-NET address these issues with a selection operator specifically designed to act on a redundant population of rules, called universal suffrage selection (US) operator (Giordana and Neri, 1996). A positive example is randomly selected, and a roulette wheel on the set of chromosomes covering that example is performed in order to select a fit chromosome (in case the example is not covered by any individual of the population, a new individual is created that covers that example). The selection operator is called 'universal suffrage' because examples can be viewed as voters, chromosomes as candidates, and all examples have the same right (probability) of vote. The US operator does not distinguish between examples that are harder to cover, and this can favor the emergence of so called super-individuals that cover many (easy) examples. In REGAL and G-NET this problem is tackled by using a distributed architecture of nodes where each node evolves a single population acting on a different set of examples, and the nodes co-evolve in order to favor the formation of different species. In (Divina and Marchiori, 2002) a less involved approach is implemented in the system ECL, which acts on a single population and modifies the US selection operator in order to take into account the hardness of examples. Each example gets a weight linearly proportional to the number of rules that cover that example. Weights are updated during the evolutionary process, and examples are selected by performing a roulette wheel with sectors proportional to the inverse of their weights.

That paper investigates the performance of the ECL system as a whole and does not address an interesting question: how do US and modified US selection operators affect population diversity and coverage? In this paper we tackle this issue and analyze experimentally how population diversity and coverage vary when the US selection is made less and less 'universal suffrage' by giving more power to examples that are harder to cover. More specifically we introduce another variant of the US operator, where the weight associated to an example is exponentially proportional to the number of rules covering that example, and we compare experimentally the three US based selection operators. Experiments on an artificial dataset and on real life datasets show that giving more power to examples that are harder to cover favors population diversity and maintains the coverage of many examples.

2 US Selection Operator and Variants

Selecting individuals from the population is a fundamental operation in a genetic algorithm, because new individuals are generated from them. We will here describe the US selection operator and two variants of it, called Weighted US (WUS) and Exponential Weighted US (EWUS) selection operators. In the next section we will discuss some aspects related to each operator and the reasons why we introduced the variants of the US selection operator.

2.1 US Selection Operator

This selection operator was first introduced in (Giordana and Neri, 1996). The basic idea behind this operator is that individuals are candidates to be elected, and positive examples are the voters. The US selection operator operates in two steps:

1. randomly select n examples from the positive examples set;
2. for each selected positive example e_i, $1 \leq i \leq n$, let $Cov(e_i)$ be the set of individuals covering e_i. If $Cov(e_i) = \emptyset$ then call a seed procedure for creating an individual that covers e_i. If $Cov(e_i) \neq \emptyset$, choose one individual from $Cov(e_i)$ with a roulette wheel mechanism, where the sector associated to an individual $x \in Cov(e_i)$ is proportional to the ratio between the fitness of x and the sum of the fitness of all individuals occurring in $Cov(e_i)$.

In this way, each positive example has the same voting power, i.e. has the same probability of being selected in the first step of the US selection operator.

The US selection operator was used within a distributed system. In this system various genetic nodes perform a GA on some training examples. A supervisor process assigns the training example sets to each node, changing these sets periodically. The supervisor can, in this way, shift the focus of the genetic search, performed by each genetic node, toward specific regions of the hypothesis space.

2.2 WUS Selection Operator

A first variation of the US selection operator is represented by the Weighted US (WUS) selection operator (Divina and Marchiori, 2002). The difference between this operator and the US selection operator lies in the first step of the selection. Examples have different voting power, i.e. examples are no longer chosen at random. A weight is associated to each example, where smaller weights are associated to examples harder to cover. More precisely the weight for an example e_i is equal to:

$$w_i = \frac{|Cov(e_i)|}{|Pop|} \tag{1}$$

$1 \leq i \leq P$, being $|Pop|$ the population size and P the number of positive examples. If the population is empty, then each example has the same weight. The weight of each example is adjusted at each iteration. Examples are now chosen with a roulette wheel mechanism, where the sector associated to each example is inversely proportional to its weight. So the less individuals cover an example, the more chances that example has of being selected in the first step of the selection.

2.3 EWUS Selection Operator

The WUS selection operator selects with higher probability examples that are harder to cover, by means of the weights assigned to each example. The Exponential Weighted US (EWUS) selection operator continues this line, with the difference that now the weight w_i of an example e_i, $1 \leq i \leq P$, is given by the following formula:

$$w_i = \frac{e^{-|Cov(e_i)|}}{\sum_{j=1}^{P} e^{-|Cov(e_j)|}} \tag{2}$$

Examples are still selected with a roulette wheel mechanism, where the sector associated to each example is proportional to its weight. In this way the selection pressure toward examples harder to cover will be much higher than in the WUS selection operator.

In both the WUS and the EWUS operators, the second step of the selection is the same as the one performed by the US operator. At the end of each iteration the weights are updated.

3 Discussion

With the introduction of the two weighted variants of the US selection operator we wanted to achieve the following aims:

- Good coverage of positive examples.
- Diversity in the population.

One would usually like to have the final hypothesis found by the GA to cover as many positive examples as possible and as few negative examples as possible. The variants of the US selection operator deal with the coverage of positive examples, while the fitness function deals also with the second aspect besides the coverage of positive examples. By having the learning system focusing more and more on difficult example to cover, it is easier to have each positive example covered by at least one individual.

Maintaining a good diversity in the population would also lead to a good coverage of examples. Generally it is a good idea to have the population spread across the hypothesis space, so that all the areas of the hypothesis space can be searched. There are various way of doing that, e.g. with distance measures, (Burke et al., 2002).

The US selection operator considers all the examples as equally important. In this way the search can focus mainly in the areas where the concentration of positive examples is higher, and seldom touches less inhabited areas of the hypothesis space. With the weighted variants of the US operator, examples difficult to cover will have higher chance of being selected. In this way, if we have many individuals covering the same subset of training examples, those individuals will be seldom selected for reproduction, because that area of the hypothesis space is already crowded. Instead we will select, or create, individuals that occupy, or will occupy, less exploited regions.

Formally, let E_x^+ be the set of positive examples covered by the individual x. Then the probability that an individual x has of being selected can be written as:

$$P(x) = \sum_{e_i \in E_x^+} P(x|e_i) \cdot P(e_i)$$

where $P(x|e_i)$, the probability of x being selected conditioned to the selection of example e_i in E_x^+, is equal to

$$\frac{f(x)}{\sum_{x \in Cov(e_i)} f(x)}$$

with $f(x)$ the fitness of individual x. The three US based selection operators induce different probability $P(e_i)$ of selection of example e_i.

In the US selection operator $P(e_i) = \frac{1}{P}$. No distinction is made among positive examples, so individuals with a high recall are favored in this scheme, because they cover many positive examples. A possible effect of this could be the presence of super individuals, that will dominate the population and will be often selected, leading to a low diversity in the population. Also some examples could stay uncovered, because not selected and difficult to cover.

In the WUS selection operator $P(e_i)$ is proportional to $\frac{1}{w_i+1}$, with w_i defined in equation 2.2. In this way WUS tries to favor not only examples that are uncovered, but in general it favors examples that are difficult to cover. Individuals with a high recall will still have more changes of being selected. However the system will focus more and more on examples harder to cover. In this way it is more probable that at the end of the evolution process all positive examples are covered. The weights given to examples by the WUS operator increase linearly with each individual that cover the example. This means that the chances that an example covered by one individual is selected are almost the same as the ones of an uncovered example. The EWUS selection operator deals with this problem, by choosing $P(e_i) = w_i$, with w_i defined in equation 2.3, which changes exponentially. If there are uncovered examples, then the probabilities that they have of being selected with the EWUS operator are very high, and also examples covered by few individuals are likely to be chosen.

In this way a good coverage of positive examples is encouraged, leading also to a good diversity in the population.

Individuals with an high recall will be selected more often only if there are not multiple copies of them, because in this case the examples that they cover will have an high coverage rate, and if these individuals do not cover only easy examples.

4 Experimental Setup

In order to substantiate the beneficial properties of the variants of the US operator stated above, we perform experiments on both artificially generated as well as real life datasets. The genetic algorithm used in the experiments is based on ECL, and is summarized in pseudo-code in figure 1. A detail description of the system can be found in (Divina and Marchiori, 2002). Here we will give a general explanation of the main features used in the algorithm.

```
ALGORITHM ECL
Sel = positive_examples
repeat
    Select partial Background Knowledge
    Population = Initial_pop
    while (not terminate) do
        Select n chromosomes using Sel
        for each selected chromosome chrm
            Mutate chrm
            Optimize chrm
            Insert chrm in Population
        end for
    end while
    Store Population in Final_Population
    Sel = Sel - { positive examples
        covered by clauses in Population }
until max_iter is reached
Extract final theory from Population
```

Figure 1: The overall learning algorithm ECL

In the repeat statement of the algorithm a `Final_population` is iteratively built. At each iteration a part of the background knowledge is chosen by means of a simple stochastic sampling mechanism. This partial background knowledge will be used for building and evaluating individuals inside each iteration of the algorithm.

A `Population` is evolved by the repeated application of selection, mutation and optimization. At each iteration n individuals are selected. Individuals are selected with one of the three selection operators described in section 2.

Each individual undergoes a mutation and an optimization phase. The mutation consists in the application of one of the four generalization/specialization operators. A clause can be generalized either by deleting a predicate from its body or by turning a constant into a variable. With the dual operations a clause can be specialized. After that an individual has been mutated, the optimization phase is performed. This phase consists of the repeated application of one of the four mutation operators, until the fitness of the individual increases (in this case the last mutation is retracted), or a maximum number of optimization steps has been reached.

Each mutation operator has a degree of greediness. In order to make a mutation, a number of mutation possibilities is considered and the one leading to the best improvement is applied.

The system does not make use of any crossover operator. Experiments with a simple crossover operator, which uniformly swaps atoms of the body of the two clauses, have been conducted. However the obtained results did not justify its use.

The so modified individuals are then inserted in the population. If the population is not full then the individuals are simply inserted. If the population has reached its maximum size, then n tournaments are made among the individuals in the population and the resulting n worst individuals are substituted by the new individuals.

At the end of the run a logic program is extracted from the final population. The theory has to cover as many positive examples as possible, and as few negative ones as possible. This problem can be translated into an instance of the weighted set covering problem as follow. Each individual is a column with positive weight equal to the fitness of the individual. The rows are positive and negative examples. The columns relative to each positive example are the clauses that cover that example, while the columns relative to the negative examples are the clauses that do not cover that example. A fast heuristic algorithm (Marchiori and Steenbeek, 2000) is applied to this problem instance to find a "best" theory.

The fitness of an individual x is given by the inverse of its accuracy:

$$fitness(x) = \frac{1}{Acc(x)} = \frac{P+N}{pos_x+(N-n_x)}$$

In the above formula P and N are respectively the total number of positive and negative examples in the training set, while p_x and n_x are the number of positive and negative examples covered by the individual x. We take the inverse of the accuracy, because ECL was originally designed to minimize a fitness function.

ECL uses a high level representation similar to the one used by SIA01, where a rule $p(X,Y) \leftarrow r(X,Z), q(Y,a)$. is described by a list of the form:

$$\boxed{p,X,Y}\ ,\ \boxed{r,X,Z}\ ,\boxed{q,Y,a}$$

In this way it is easy to represent rules of variable length, and to access and process atoms contained in the rule.

	Toy	Mutagenesis	Vote
pop_size	30	30	30
mut_rate	1	1	1
n	10	10	10
max_gen	1	1	1
max_iter	8	10	10

Table 1: Parameter settings for the various datasets. All the experiments were done using the whole background knowledge.

The parameter settings used in the experiments are given in table 1. These values have been obtained after a few experiments on the training sets.

5 An Artificially Generated Example

In order to assess the validity of each selection operator a dataset was built. The dataset is made of five hundred positive examples and five hundred negative examples. Each example can be described by three attributes q, p and r, that can assume respectively the values $\{a, b\}$, $\{c, d, e\}$ and $\{f, g\}$.

Attr.	Pos	Neg
a,c	0.50	0.05
a,d	0.30	0.05
a,e	0.05	0.50
b,c	0.05	0.20
b,d	0.05	0.10
b,e	0.05	0.10
f	0.70	0.30
g	0.30	0.70

Table 2: Probabilities of having a particular combination of attributes/values describing an example, given a positive (Pos) or a negative (Neg) example in the dataset.

The dataset was generated with the probabilities given in table 2. For instance, the probability of having a positive example described by the attribute q with value a and by the attribute p with value c is 0.5, while the probability of having a negative example described by the same pair of attributes values is 0.05.

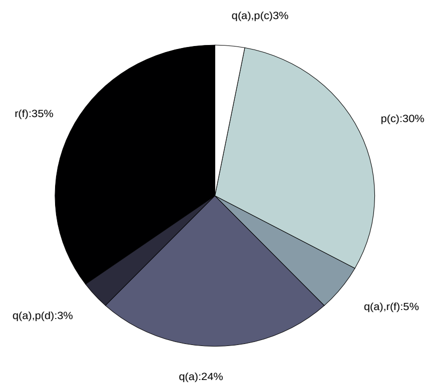


Figure 2: Distribution of different rules in the final population obtained with the use of the US selection operator.

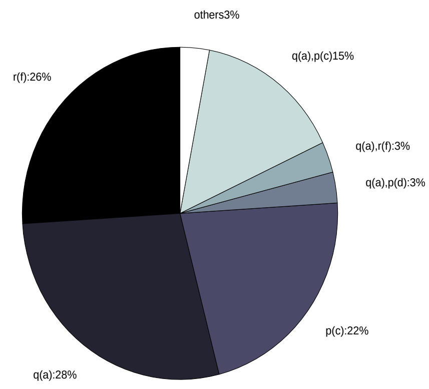


Figure 3: Distribution of different rules in the final population obtained with the use of the WUS selection operator.

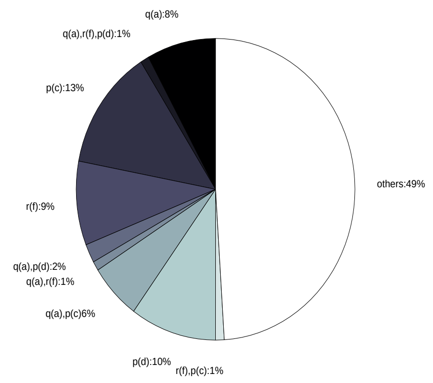


Figure 4: Distribution of different rules in the final population obtained with the use of the EWUS selection operator.

Figures 2, 3 and 4 show the percentage of different kind of individuals in the final population after that the system was run for five times for each operator. The difference between the US and the EWUS selection operator is evident here. In the population obtained with the US selection operator, there are three kind of rules that dominate the population, namely $r(X,f), q(X,a)$ and $p(X,c)$. Instead, in the population obtained with the EWUS operator the distribution of different kind of rules is more homogeneous. There are not individuals that dominate the population. Also the WUS operator shows better results regarding the diversity in the population. There are still individuals that dominate the population, but the distribution of different kind of rules is more even.

	US	WUS	EWUS
Unc	13.5 (3.53)	13 (0.01)	0.67 (0.58)
Avg	19.80 (0.24)	17.20 (3.11)	9.00 (0.23)

Table 3: The first row of the table shows how many positive examples were still uncovered after the GA ended. The second row shows the average of the dimension of the coverage sets, which is how many individuals cover a positive example.

Table 3 shows another result in which we were interested: how the positive examples are covered by population evolved by the system using the three selection operators. Again, results obtained with the EWUS operator are better than the ones obtained with the other two operators. Practically with the use of the EWUS operator, all the positive examples are covered, while with the other two variants thirteen positive examples are still uncovered after the GA has ended. This is due to the more variety in the population evolved by the system with the use of the EWUS selection operator. The second row of table 3 also indicates that the population obtained with the EWUS operator is more spread through the hypothesis space then the ones obtained with the other two operators. For this result the WUS operator performed a little bit better than the US operator, however these results are not comparable with the ones obtained by the EWUS operator.

In the previous results, the new created in-

Clause	US	WUS	EWUS
f	0.62 (0.06)	0.58 (0.04)	0.15 (0.01)
a	0 (0)	0.01 (0.01)	0.11 (0.01)
c	0 (0)	0.02 (0.01)	0.03 (0.04)
a,d	0 (0)	0.01 (0.01)	0.04 (0.03)
a,f	0.1 (0.05)	0.14 (0.11)	0.06 (0.05)
a,c	0.16 (0.05)	0.14 (0.08)	0.07 (0.04)
d	0 (0)	0 (0)	0.02 (0.01)
others	0.12 (0.03)	0.11 (0.03)	0.52 (0.05)

Table 4: Diversity in the final populations evolved using the three selection operators with a different replacement strategy.

dividuals replaced the worst individuals in the whole population. Table 4 shows results where new individuals replace the worst individuals among those individuals that cover the example selected in the first step of the selection process. It can be seen that also in this setting the diversity obtained by the EWUS operator is higher than the one obtained by the other two operators. With this replacement strategy the system was not able to find some rule, e.g $r(X, f), p(X, c)$, regardless of the selection operator used.

Table 5 shows the results regarding the coverage. Also here the EWUS performed better than the other two operators, the higher diversity obtained led to a better coverage of the positive examples.

	US	WUS	EWUS
Unc	56 (24.04)	24.67 (3.68)	3.67 (0.47)
Avg	16.48 (11.11)	17.69 (10.45)	9.77 (3.96)

Table 5: Number of positive examples uncovered and average dimension of covering sets for the alternative substitution method.

6 Experiments

After having validated the effectiveness of the EWUS selection operator in the previous section, we will test again the three operators on two well known problems. For these problems the evaluation method used is ten-fold cross validation.

The first problem is represented by the mutagenesis dataset (Debnath et al., 1991). This dataset comes from the field of organic chemistry, and concerns the problem of learning the mutagenic activity of nitroaromatic compounds. The concept to learn is expressed by the predicate $active(C)$, which states that compound C has mutagenic activity. There are 188 compounds, of which 125 are active and 63 are inactive. Five runs were performed on this dataset, with different random seeds.

	US	WUS	EWUS
Div	8 (0.82)	9.34 (3.2)	21 (1.63)
Unc	10 (2.45)	7.67 (3.09)	0.33 (0.47)
Avg	17.36 (7.85)	15.33 (7.35)	6.97 (2.81)
Acc	0.85 (0.08)	0.86 (0.08)	0.89 (0.06)

Table 6: The first row of the table shows how many different rules are present in the final population obtained with the three selection operators on the mutagenesis dataset. The second row shows the number of uncovered positive examples at the end of the evolution. The third row shows the average dimension of the coverage sets. The fourth row shows the accuracy of the solutions found. The results are averages over five runs with standard deviation reported between brackets.

Table 6 shows the results obtained on the mutagenesis dataset by the three variants of the selection operators. The use of the EWUS operator led to a population with a much higher diversity than the two population evolved with the use of the other two selection operators. This is also reflected in the coverage of positive examples. Also the accuracy obtained by the system with the EWUS operator is better than the other two.

	US	WUS	EWUS
Div	8.67 (1.47)	13.5 (1.50)	13 (1.12)
Unc	2.33 (0.47)	0.80 (0.75)	0 (0)
Avg	21.13 (7.42)	20.50 (8.38)	22.87 (5.16)
Acc	0.92 (0.05)	0.93 (0.06)	0.94 (0.04)

Table 7: Results for the vote dataset. Diversity, number of uncovered examples, average dimension of covering sets and accuracy are shown.The results are averages over four runs with standard deviation given between brackets.

A second problem is represented by the vote dataset (Blake and Merz, 1998). This dataset contains votes for each of the U.S. House of Representatives Congressmen on the sixteen key votes. The problem is learning a concept for distinguishing between democratic and republican congressmen. The dataset consists of 435 instances, of which 267 are examples of democrats and 168 are republicans.

The results obtained on the vote dataset are shown in table 7, after having run the system for four times with different random seeds. In this case the EWUS and the WUS operator performed substantially in the same way. Both operator helped the system to evolve a population with higher diversity than the population obtained with the US operator. The average dimension of the covering sets is practically the same. It must be said, however, that for the US operator this figure was influenced by the number of uncovered examples. Also in this case the EWUS operator led to a better accuracy.

7 Conclusion

In this paper we considered variants of the US selection operator which incorporate information on the difficulty of the examples to be covered. We analyzed experimentally the effect of these operators on the diversity in the evolved population as well as on the coverage of positive examples. The results suggest that 'less' universal selection schemes are more effective for promoting diversity while maintaining the key property of the US selection operator of covering many positive examples.

References

C. Anglano, A. Giordana, G. Lo Bello, and L. Saitta. 1998. An experimental evaluation of coevolutive concept learning. In *Proc. 15th International Conf. on Machine Learning*, pages 19–27. Morgan Kaufmann, San Francisco, CA.

S. Augier, G. Venturini, and Y. Kodratoff. 1995. Learning first order logic rules with a genetic algorithm. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *The First International Conference on Knowledge Discovery and Data Mining*, pages 21–26, Montreal, Canada, 20-21. AAAI Press.

C.L. Blake and C.J. Merz. 1998. UCI repository of machine learning databases.

R. Burke, S. Gustafson, and G. Kendall. 2002. A survey and analysis of diversity measures in genetic programming. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 716–723, New York, 9-13 July. Morgan Kaufmann Publishers.

A.K. Debnath, R.L. Lopez de Compadre, G. Debnath, A.J. Schusterman, and C. Hansch. 1991. Structure-Activity Relationship of Mutagenic Aromatic and Heteroaromatic Nitro Compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of Medical Chemistry*, 34(2):786–797.

F. Divina and E. Marchiori. 2002. Evolutionary concept learning. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 343–350, New York, 9-13 July. Morgan Kaufmann Publishers.

A. Giordana and F. Neri. 1996. Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416.

J. Hekanaho. 1998. DOGMA: a GA based relational learner. In D. Page, editor, *Proceedings of the 8th International Conference on Inductive Logic Programming*, LNAI 1446, pages 205–214. Springer Verlag.

C.Z. Janikow. 1993. A knowledge intensive genetic algorithm for supervised learning. *Machine Learning*, 13:198–228.

C. J. Kennedy and C. Giraud-Carrier. 1999. A depth controlling strategy for strongly typed evolutionary programming. In *GECCO 1999: Proceedings of the First Annual Conference*, pages 1–6. Morgan Kauffman, July.

M. Kubat, I. Bratko, and R.S. Michalski. 1998. A review of Machine Learning Methods. In R.S. Michalski, I. Bratko, and M. Kubat, editors, *Machine Learning and Data Mining*. John Wiley and Sons Ltd., Chichester.

K.S. Leung and M.L. Wong. 1995. Genetic logic programming and applications. *IEEE Expert*, 10(5):68–76.

E. Marchiori and A. Steenbeek. 2000. An Evolutionary Algorithm for Large Scale Set Covering Problems with Application to Airline Crew Scheduling. In *Real World Applications of Evolutionary Computing. Springer*, pages 367–381. Springer-Verlag.

T.M. Mitchell. 1982. Generalization as search. *Artificial Intelligence*, 18:203–226.

T.M. Mitchell. 1997. *Machine Learning*. Series in Computer Science. McGraw-Hill.

S. Muggleton. 1995. Inverse entailment and PROGOL. *New Generation Computing*, 13:245–286.

J.R. Quinlan. 1990. Learning logical definition from relations. *Machine Learning*, 5:239–266.

On Using Guidance in Relational Reinforcement Learning

Kurt Driessens
Department of Computer Science
K.U.Leuven
Belgium
kurt.driessens@cs.kuleuven.ac.be

Sašo Džeroski
Department of Intelligent Systems
Jožef Stefan Institute
Slovenia
saso.dzeroski@ijs.si

Abstract

Reinforcement learning, and Q-learning in particular, encounter two major problems when dealing with large state spaces. First, learning the Q-function in tabular form may be infeasible because of the excessive amount of memory needed to store the table and because the Q-function only converges after each state has been visited multiple times. Second, rewards in the state space may be so sparse that with random exploration they will only be discovered extremely slowly. The first problem is often solved by learning a generalization of the encountered examples (e.g., using a neural net or decision tree). Relational reinforcement learning (RRL) is such an approach; it makes Q-learning feasible in structural domains by incorporating a relational learner into Q-learning. To solve the second problem a use of "reasonable policies" to provide guidance has been suggested. In this paper we investigate the best ways to provide guidance in two different domains.

1 Introduction

Q-learning (Watkins, 1989) is a form of reinforcement learning where the optimal policy is learned implicitly in the form of a Q-function, which takes a state-action pair as input and outputs the quality of the action in that state. The optimal action in a given state is the action with the largest Q-value.

One of the main limitations of standard Q-learning is related to the number of different state-action pairs that may exist. The Q-function can in principle be represented as a table with one entry for each state-action pair. When states and actions are characterized by parameters, the number of such pairs grows combinatorially in the number of parameters and thus can easily become very large, making it infeasible to represent the Q-function in tabular form, let alone learn it accurately (convergence of the Q-function only happens after each state-action pair has been visited many times). This problem is typically solved by integrating into the Q-learning algorithm an inductive learner, which learns a function that generalizes over given state-action pairs. Thus reasonable estimates of the Q-value of a state-action pair can be made without ever having visited it. Examples include neural networks (Bertsekas and Tsitsiklis, 1996), nearest neighbour methods (Smart and Kaelbling, 2000) and regression trees (Chapman and Kaelbling, 1991).

A relational learner is employed by Džeroski et al. in(Džeroski et al., 1998), hence the name "relational reinforcement learning" or RRL. RRL uses first order representations for states and actions, and learns a first order regression tree (Kramer, 1996; Blockeel et al., 1998) that maps these structural descriptions onto real numbers. The use of first order representations gives RRL a broader application domain than classical Q-learning approaches. Examples of such relatively complex applications are described in more detail in this paper, include learning to solve simple planning tasks in a blocks world, or learning to play certain computer games (such as Tetris).

When dealing with large state spaces, as is usually the case in structured domains, another problem encountered by Q-learning is that rewards may be distributed very sparsely throughout this space. In previous work (Driessens and Džeroski, 2002), we have suggested to supply guidance to the learning system at the start of the learning process to provide it with enough knowledge to explore the rest of the state-space on its own. In this work we will try to explore other ways of supplying guidance to RRL.

The remainder of the paper is structured as follows. In Section 2 we discuss different ways in which guidance can be incorporated in a Q-learning approach. In Section 3, we present a number of structural domains and discuss specific difficulties that are encountered when solving tasks in these domain. In Section 4, we present experimental results in these domains, comparing different ways of providing guidance to RRL and we conclude in Section 5. For related work we would like to refer to previous work (Driessens and Džeroski, 2002).

2 The Approach

2.1 The RRL-TG System

We make use of the RRL-TG algorithm as described in (Driessens et al., 2001). The RRL system is a Q-learning system that uses a first order representation for the encountered states, actions and the resulting Q-function. RRL starts with running a normal episode just like a standard reinforcement learning algorithm but uses this episode to generate a set of examples for tree induction. At the end of each episode, a first order regression tree builder TG is supplied with the encountered (*state*, *action*, *q-value*)-triplets and incrementally builds a first order regression tree that represents the Q-function. In the next episode, the q-values predicted by the generated tree are used to calculate the q-values for the next set of examples.

2.2 On Providing Guidance

Although random policies can have a hard time reaching sparsely spread rewards in a large world, it is often possible to reach these rewards in a reasonable number of steps by using "reasonable" policies. While optimal policies are certainly "reasonable", non-optimal policies are often easy (or easier) to implement or generate than optimal ones. One obvious candidate for an often non-optimal but reasonable controller would be a human expert.

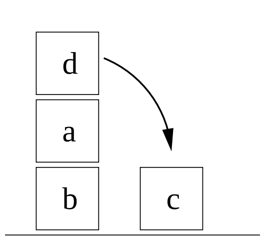
To integrate the guidance that these reasonable policies can supply with our relational reinforcement learning system, we use the given policy to generate a number of behaviour traces, together with the rewards that the actions in such a trace would receive. In case of a human controller, one can log the normal operation of a system together with the corresponding rewards. The generated traces are translated to state-action-qvalue triplets and then presented to the RRL algorithm. The generalisation algorithm responsible for the generation of the Q-function in RRL can use the state-action-qvalue triplets generated this way to build a partial Q-function.

In earlier work (Driessens and Džeroski, 2002), we supplied all of the guidance (traces) in the beginning of the learning experiment. Although this generally turned out to help RRL reach higher levels of performance, we also noticed the occurrence of overgeneralization in the regression algorithm TG. We propose in this work to counter this overgeneralization-effect caused by the supply of optimal state-action combinations only, by interleaving the guided traces with the exploration traces. We will compare different rates at which the guidance is supplied.

In the guided traces, the actions are chosen by a pre-defined (reasonable) policy and are only observed by RRL. In the exploration traces, actions are chosen autonomously by RRL, using the current estimate of the Q-function and the Boltzmann statistics (Kaelbling et al., 1996). For the generalization engine TG, there is no difference between the two types of traces. It will simply transform each encountered (*state*, *action*, *q-value*) triplet into one training example to learn from. A more formal explanation of this approach can be found in (Driessens and Džeroski, 2002).

We will test the different suggestions on two example applications: the Blocks World and the computer game Tetris. While we will use an optimal policy to provide guidance in the Blocks World, it is very difficult — and maybe impossible — to construct an optimal policy for Tetris. We will therefore compare guidance given by two different — but still fairly simple — policies, of which one performs considerably better than the other.

In the case of the Blocks World we will try to go one step further in the kind of guidance we provide. Since we test the performance of the strategy generated by RRL at regular time intervals, we can tell RRL in which cases it failed. Next time RRL asks for some guidance, we can allow it to ask for help in one of the cases that it



```
clear(d).
clear(c).
on(d,a).
on(a,b).
on(b,floor).
on(c,floor).
move(d,c).
```

Figure 1: Example state and action in the blocks-world.

could not solve. We have named this approach "active" because of the similarity of this technique to "active learning" strategies in classification tasks (Cohn et al., 1994).

3 The Domains

3.1 The Blocks World

We use the blocks world with 10 blocks as our first testing environment. Blocks can be on the floor or can be stacked on each other. We represent the states and actions in the blocks-world as shown in Figure 1 (Prolog notation). The available actions are $move(x,y)$ where $x \neq y$, x is a block and y is a block or the floor. There is no limit on the number of blocks that can be on the floor. In addition, we supply the RRL algorithm with the number of blocks, the number of stacks and the following background predicates: *equal*/2, *above*/2, *height*/2 and *difference*/3 (an ordinary subtraction of two numerical values).

With 10 blocks, the blocks world becomes large enough to illustrate the effect of guidance in Q-learning, while remaining simple enough to explore different guidance strategies. With 10 blocks, there are close to 59 million possible states in the blocks world.

We study the goal of putting one specified block on top of another specified block. Please note that the use of RRL permits the use of variables in the description of the goal and the Q-function and thus allows us to learn to stack any two given blocks, without having to restart the learning process when changing the names of the blocks. In a blocks world with 10 blocks, there are 1.5 million states that satisfy a specific $on(A,B)$ goal. A reward of 1 is given in case a goal-state is reached in the optimal number of steps; the episode ends with a reward of 0 if it is not. This limits the number of steps allowed per episode.

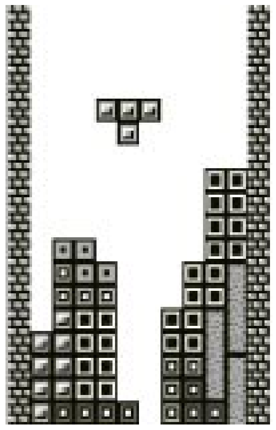


Figure 2: The TETRIS Game.

3.2 The Tetris Game

Tetris[1] is a well known puzzle-video game played on a two-dimensional grid. Differently shaped blocks fall from the top of the game field and fill up the grid. The object of the game is to keep the blocks from piling up to the top of the game field. To do this, one can move the dropping blocks right and left or rotate them as they fall. When one horizontal row is completely filled, that line disappears and the player scores points. When the blocks pile up to the top of the game field, the game ends.

In the tests presented, we only looked at the strategic part of the game, i.e., given the shape of the dropping and the next block, one has to decide on the optimal orientation and location of the block in the game-field. Using low level actions — turn, move left or move right — to reach such a subgoal is rather trivial and can easily be learned by RRL. However, the dropping of the block is instantaneous. It is impossible in our setup to slide a block into place sideways. We represent the full state of the Tetris game, the type of the next dropping block included. We allow RRL to use the following predicates (among others): $blockwidth/2$, $blockheight/2$, $rowSmaller/2$, $topBlock/2$, $holeDepth/2$, $holeCovered/1$, $fits/2$, $increasesHeight/2$, $fillsRow/2$ and

[1]Tetris was invented by Alexey Pazhitnov and is owned by The Tetris Company and Blue Planet Software.

$fillsDouble/2$. The system gets a reward after each action of 1 point for each (with that action) deleted line.

3.3 Comments

Both domains have large state spaces and are hard to solve with ordinary Q-learning. Although some of these problems can be tackled by using RRL, Q-learning in both of them has been shown to benefit from the added guidance (Driessens and Džeroski, 2002).

In Tetris, a good policy can supply a reward every 4 or 5 steps, which is quite frequent. However, a single bad action can have long lasting effects, thereby hiding obvious rewards from the learning algorithm. This, together with the fact that Tetris is a stochastic game (the shape of the next falling block is unknown) makes it a very hard application for Q-learning. For the Tetris game, an optimal policiy is hard — and may be impossible — to construct.

4 The Experiments

We expect that spreading the presented guidance will have two influences on the learning performance. In terms of learning speed, we expect the guidance to help the Q-learner to discover high yielding policies earlier in the learning experiment. Through the early discovery of important states and actions and the early availability of these state-action pairs to the generalization engine, we also expect that it is possible for the Q-learner to reach a higher level of performance — i.e., a higher average reward when testing the current strategy — in the available time.

To test these effects, we compare RRL with different guidance frequencies in the following setup: First we run RRL in its natural form and in the form where we supply all the guidance in the beginning of the learning episode (Driessens and Džeroski, 2002). We give it the possibility to train for a certain number of episodes; at regular time intervals we extract the learned policy from RRL and test it on a number of randomly generated test problems. For RRL with guidance, we replace a given number of episodes with traces from a hand-coded policy. We test the result in the same manner as the original RRL. Note that no matter what frequency we use to supply RRL with guidance, the total amount of guided traces will be the same in all cases.

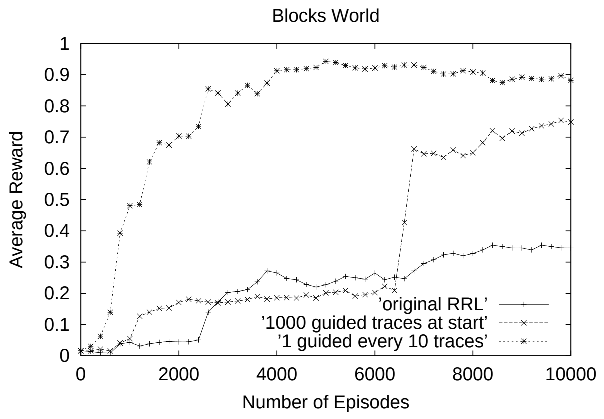


Figure 3: The Learning Curves for the Blocks World.

4.1 The Blocks World

As is clearly shown in Figure 3 the spreading of the guidance helps in terms of both learning speed and level of performance. Providing an equal amount of guidance in the beginning of learning gives a (relatively) comparable increase in level of performance, but not in terms of learning speed. This can be explained by the interaction of the TG-algorithm with the fact that only optimal traces are presented. When we supply RRL with only optimal traces, overgeneralization occurs. The generalization engine never encounters a non-optimal action and therefore, never learns to distinguish optimal from non-optimal actions. It will create a Q-tree that separates states which are at different distances from the goal-state and later, during exploration, expand this tree to account for optimal and non-optimal actions in these states. These trees are usually larger than they should be. RRL is often able to generalize over non-optimal actions in states that are close to the goal and optimal actions in states that are a little further from the goal in one leaf of its tree.

By spreading the guidance and supplying the optimal traces not all at once but interleaved with random — at least in the beginning of the learning process — traces avoids this overgeneralization. Of course, this problem (and as a consequence the solution as well) is strongly related to model building generaliz-

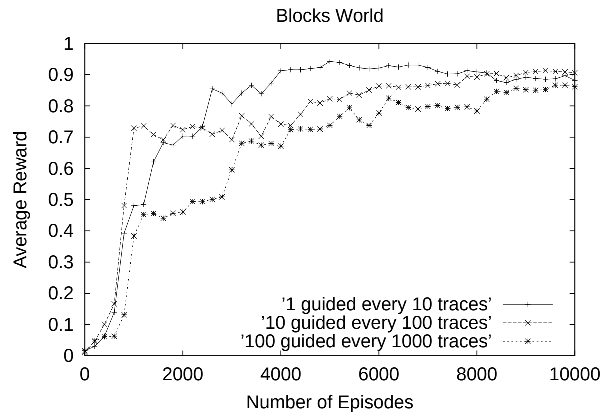


Figure 4: Comparing different Guidance Frequencies.

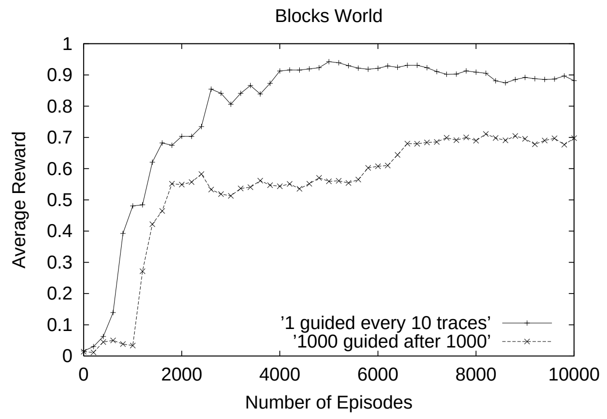


Figure 5: Comparing Spread Guidance to Delayed Batch Guidance.

ation algorithms such as tree-builders or rule-based systems. Other generalization techniques such as instance-based approaches might not suffer from this problem.

Figure 4 shows the influence of different frequencies used to provide guidance. Note that in all cases, an equal amount of guidance was used. Intuitively, it seems best to spread the available guidance as thin as possible and the performed experiments do not show any negative results of doing so. However, spreading out the guidance when there is only a small amount available (e.g. 1 guided trace every 10 000 episodes) might prevent the guidance from having any effect.

As a specific solution to the sparsity of guidance for the RRL-Tg-algorithm we could let RRL ask for guidance episodes whenever it is bound to make a permanent decision. (In the case of the tree-learner Tg this would be when it is about to make a new split in a leaf.) Even when this guidance is not case specific, it could be used to check whether a reasonable policy does not contradict the proposed split. Alternatively, one might decide to store (some of) the guided traces and re-use them: at present, all traces are forgotten once a split of the TG tree has been made.

When looking for a more general solution, one could try to provide a larger batch of guidance after RRL has had some time to try and explore the state-space on its own. This is related to a human teaching strategy, where providing the student with the perfect strategy at the start of learning is less effective than providing the student with that strategy after he or she has had some time to explore the systems behavior. Although Figure 5 shows inferior results for this approach when compared to the spread out guidance, this is probably due to the large size of the presented batch. Note also that learning here takes place faster than when all guidance is provided at the beginning.

4.2 The Tetris Game

The guidance strategy used in the Tetris experiment was very simple and included only the following rules:

1. Take an action that creates no new holes and does not increase the current height of the wall in the playing field.
2. If no action of type 1 can be found, take an action that does not increase the current height of the wall in the playing field.
3. If no action of type 1 or 2 can be taken, take an action that does not create a new hole.
4. If no action of type 1, 2 or 3 exists, take a random action.

This strategy scores an average of 6.3 lines per game.

Due to the lack of sufficient time to run more experiments, the shown graphs are performance curves of single learning experiments and as such show some erratic behavior. We will have better results for the final version.

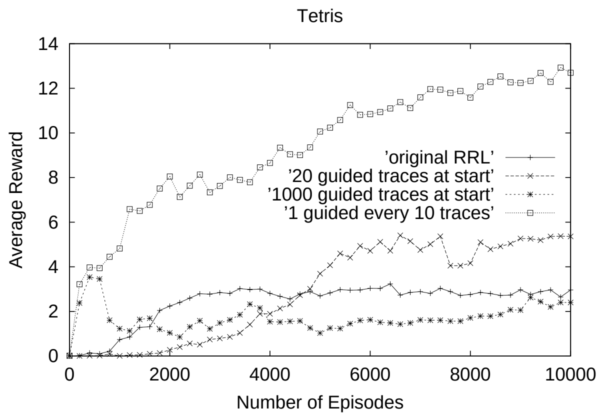


Figure 6: The Learning Curves for Tetris.

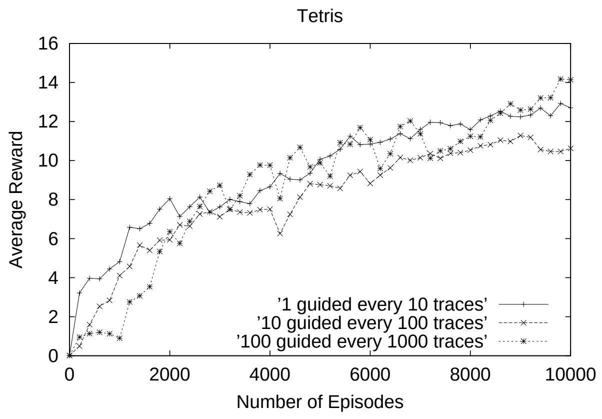


Figure 7: Comparing different Guidance Frequencies.

Figure 6 shows the improvement of RRL with guidance. Since the problem of overgeneralization is more apparent in the Tetris experiments — the experiment with 1000 starting traces does not perform better than the original RRL — we included an experiment where we gave RRL only 20 guided traces at the start of the experiment, to show that RRL does benefit from early guidance as well. However, the improvement received from spreading the guidance is even more apparent than in the Blocks World. Also note that the average number of lines deleted by RRL rises above 12 per game while the strategy used for guidance only reaches 6.3 lines per game.

Figure 7 shows the comparison of guidance frequencies. As we already noticed in the blocks-world experiments, although providing a lot of guided traces in the beginning of the experiment will slow down the progress made by the RRL-system during the initial stages of the experiment, there is little to no difference between the performances later on in the experiment.

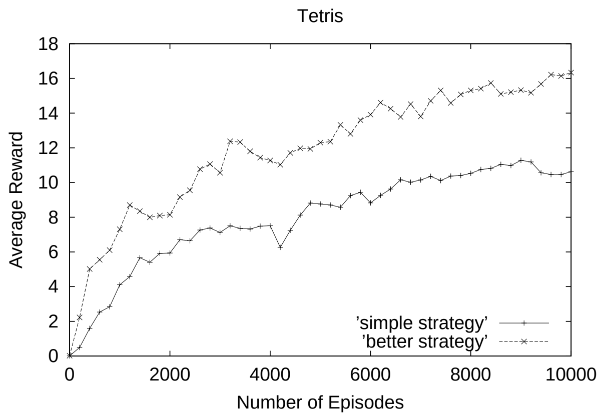


Figure 8: Comparing different Guiding Policies.

To test the influence of the performance of the policy used for guidance, we designed another (simple) strategy for Tetris. With the addition of a few more rules that tested the number of deleted lines a block would cause, we got the performance of the guidance strategy up to 16.7 lines per game on average. The results of using the two different strategies are shown in Figure 8. The graph shows that there is a significant increase in performance for using the better policy to guide RRL. This shows that the exploration insensitivity of table-based Q-learning (Kaelbling et al., 1996) does not carry over to Q-learning with generalization. However, one should notice that although the "guidance strategy" improved by approximatly 10 lines per game, the improvement of the resulting strategy learned by RRL is smaller.

4.3 "Active" Guidance

As stated at the end of Section 2, in the Blocks World where each episode is started from a randomly generated starting position, we can let RRL know in which cases it failed to reach the goal state, and give RRL the opportunity to ask for guided traces starting from some of these states. This will allow RRL to explore parts of the state space where it does not yet have enough knowledge and supply the TG-algorithm with examples which are not yet correctly predicted.

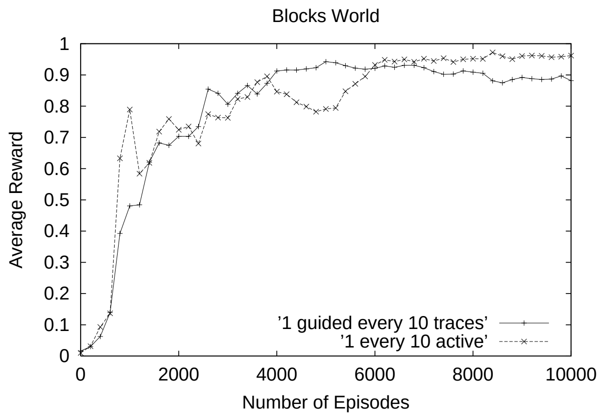


Figure 9: The Learning Curves for the Blocks World.

Figure 9 shows the results of this active guidance. Although there is little difference between the performance of the two approaches in the beginning of the learning experiments, the active guidance succeeds in pushing RRL to better performance at the end of the learning experiment. The percentage of cases where RRL does not reach the goal state is reduced from 11% to 3.9%. This is completely consistent with what one would expect. In the beginning, both systems will see more than enough new examples to both be able to increase the accuracy of their Q-function. However, when both the algorithms have a large part of the state-space covered by their Q-function, the specific examples provided by the active guidance allow for the Q-function to be extended to cover the outer reaches of the state-space.

Although this idea of "active" guidance seems very attractive both intuitively and in practice, it is not easy to extend this approach to applications with stochastic actions or a fixed starting state such as the Tetris game where the next block to be dropped is chosen randomly and the starting state is always an empty playing field. For the Tetris game one could imagine remembering the entire sequence of blocks and asking the guidance strategy for a game with the given sequence, however, given the large difference in the state provided by only a few different actions, we anticipate the effect of this approach to be very small. Another step towards active guidance in stochastic environments would be to keep track of actions (and states) with a large negative effect. For example in the Tetris game we could notice a large increase of the height of the wall of the playing field. We could then use these remembered states to ask for guidance. However, this approach requires not only a large amount of administration inside the learning system but also needs some a priori indication of bad and good results of actions.

5 Conclusions

In this paper, we explored several ways of including guidance into reinforcement learning. Although we have used relational reinforcement learning (RRL) in our experiments, we assume that the results carry over to all forms of Q-learning that use regression to approximate the Q-function. We have shown that spreading this guidance during the entire learning eposide can have a large influence on both the speed of learning and the overall level of performance that is obtained.

According to the results we obtained the approach does not suffer when the guidance is spread out thinly. We feel that spreading out the guidance as much as possible would yield the best results, provided there is enough guidance available to not lose its influence to noise elimination.

The influence of the performance of the strategy used for guidance seems small. Assuming the use of a "reasonable" policy, i.e. a policy that is able to discover the rewards in the state-space at a reasonable rate, the policy learned by RRL depends little on the performance of the policy used for guidance. The RRL-system is able to improve on the performance of the guidance policy.

The use of "active" guidance seems to help improve the performance of RRL even more in the case of deterministic applications. The specific examples provided by this active guidance helps the Q-function to cover a larger area of the state-space by allowing the learning algorithm to zoom in on unknown areas of the state-space. However, it seems hard to expand this approach to stochastic applications while maintaining the simplicity and elegance of the original idea.

Possible directions for further work include tighter integration of the use of guidance and the generalization engine used (RRL-TG), on one hand, and the investigation of the effects of using guidance with other types of learning algorithms (e.g., instance-based) within Q-learning, on the other hand.

References

Bertsekas and Tsitsiklis. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

H. Blockeel, L. De Raedt, and J. Ramon. 1998. Top-down induction of clustering trees. pages 55–63.

D. Chapman and L. P. Kaelbling. 1991. Input generalization in delayed reinforcement learning: An algorithm and performance comparisions. pages 726–731.

D. Cohn, L. Atlas, and R. Ladner. 1994. Improving generalization with active learning. 15:201–221.

K. Driessens and S. Džeroski. 2002. Integrating experimentation and guidance in relational reinforcement learning. In C. Sammut and A. Hoffmann, editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 115–122. Morgan Kaufmann Publishers, Inc.

K. Driessens, J. Ramon, and H Blockeel. 2001. Speeding up relational reinforcement learning through the use of an incremental first order decision tree learner. In *Proceedings of the 13th European Conference on Machine Learning*, pages 97–108. Springer-Verlag.

S. Džeroski, L. De Raedt, and H Blockeel. 1998. Relational reinforcement learning. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 136–143. Morgan Kaufmann.

L. Kaelbling, M. Littman, and A. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Stefan Kramer. 1996. Structural regression trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 812–819, Cambridge/Menlo Park. AAAI Press/MIT Press.

W. D. Smart and L. P. Kaelbling. 2000. Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th International Conference on Machine Learning*, pages 903–910. Morgan Kaufmann.

Christopher Watkins. 1989. *Learning from Delayed Rewards*. Ph.D. thesis, King's College, Cambridge.

Datasize-Based Confidence Measure for a Learning Agent

Wojciech Jamroga
Parlevink Group
University of Twente, Netherlands
jamroga@cs.utwente.nl

Abstract

In this paper a confidence measure is considered for an agent who tries to keep a probabilistic model of her environment of action. The measure is meant to capture only one factor of the agent's doubt – namely, the issue whether the agent has been able to collect a sufficient number of observations. In this case stability of the agent's current knowledge may give some clue about the trust she can put in the model – indeed, some researchers from the field of probability theory suggest that such confidence should be based on the variance of the model (over time).

In this paper two different measures are proposed, both based on aggregate variance of the estimator provided by the learning process. The way the measures work is investigated through some simple experiments with simulated software agents. It turns out that an agent can benefit from using such measures as means for 'self-reflection'. The simulations suggest that the agent's confidence should reflect the deviation of her knowledge from the reality. They also show that it can be sometimes captured using very simple methods: a measure proposed by Wang is tested in this context, and it works seldom worse than the variance-based measures, although it seems completely ad hoc and not well suited for this particular setting of experiments at the first sight.

`Keywords:` multiagent systems, confidence measure, uncertainty, machine learning, user modeling

1 Introduction

There are roughly two possible sources of doubt for a learning agent. First, the agent may have collected too little data. For instance, when the agent starts interaction with a completely new user, her knowledge about the user is virtually none. However, the knowledge is utilized in the same way by most algorithms, regardless of the number of learning steps that have been taken so far.

Next, the environment might have changed considerably, so the data do not reflect its current shape.

The knowledge produced by a learning algorithm is often no more than a working hypothesis. It's necessary for the agent that she can make her decisions; however, trusting the knowledge blindly implies some additional assumptions which are not true in most real-life situations. It's good for the agent to have some measure of uncertainty in her own knowledge – to minimize the risk of a decision, especially in the case when she has several alternative models to choose among or combine.

This paper is focused on the first source of the agent's uncertainty: how much confidence can she have in her knowledge when there is not enough data to support it? The problem is analyzed in a very simple setting: the agent is assumed to be a 1-level agent – i.e. an agent that models other agents as stochastic agents (Sen and Weiss, 1999) – and the users are 0-level agents with probabilistic policies. The reinforcement is known beforehand for every decision of the agent, given a response from the user, and the domain of action is stateless (or at least the agent's perception doesn't let her distinguish between different states of the environment). The agent tries to estimate the actual policy of the user calculating a frequency distribution, which can be further used to find the decision with the maximal expected reward. The aim of the confidence is to represent meta-(un)certainty about the agent's knowledge, so when she has several alternative models available she can choose among them or combine their output. Thus, the actual confidence values should range from 0 (complete distrust) to 1 (full confidence).

2 The Learning Method: Counting with Decay

Assume an autonomous software agent A who interacts with some other agent B (for example, A may be an agent representing a bank and B may be a potential customer – a user of an Internet banking service). The interaction with the 'user' is sequential and it consists of subsequent turns: first A chooses to proceed with an action $a*$ from a finite set $ActA$, then B replies with some $b* \in ActB$, then A does $a' \in ActA$ and so on. Let $p_B(b|a) \equiv p_B(a,b)$ denote the current probability of agent B choosing action b as a response to A's action a. A tries to estimate the policy with a relative frequency distribution $\hat{p}_B$:

$$\hat{p}(b|a) \quad \leftarrow \quad \begin{cases} \frac{\hat{p}(b|a)N(a)\cdot\lambda+1}{N(a)\cdot\lambda+1} & a=a^*, b=b^* \\ \frac{\hat{p}(b|a)N(a)\cdot\lambda}{N(a)\cdot\lambda+1} & a=a^*, b\neq b^* \\ \hat{p}(b|a) & \text{else} \end{cases} \tag{1}$$

$$N(a) \quad \leftarrow \quad \begin{cases} N(a)\cdot\lambda+1 & a=a^* \\ N(a) & \text{else} \end{cases} \tag{2}$$

where $\lambda \in [0,1]$ is the decay rate implementing the way A 'forgets' older observations in favor of the more recent ones to model users that may change their preferences dynamically (Kumar, 1998) (Koychev, 2000) (Koychev, 2001). $N(a)$ represents the data size after collecting n observations. Since the older observations are used only partially (the first one with weight λ^{n-1}, the second: λ^{n-2} etc.), the real quantity of data we use is

$$N(a) = \sum_{i=1}^{n} \lambda^{n-i} = \begin{cases} \frac{1-\lambda^n}{1-\lambda} & \text{for } 0 < \lambda < 1 \\ n & \text{for } \lambda = 1 \end{cases}$$

The nil distribution $\mathbf{0}(b|a) = 0$ is used as the initial one. If the decay rate is allowed to vary then $N(a) = \sum_{i=1}^{n} \prod_{j=i+1}^{n} \lambda_j$, where $\lambda_1, ..., \lambda_n$ denote the actual decay rates at the moments when the subsequent observations and updates were made.

Note that $\hat{p}(b|a)$ is basically a sample mean of a Bernoulli variable $Resp(b|a)$, although it's a *mean with decay*:

$$\hat{p}_n(b|a) = M_{\lambda_{1..n}}(Resp_{i=1,...,n}(b|a)), \quad (3)$$

$$\text{where} \quad Resp(b|a) = \begin{cases} 1 & \text{if } b \text{ is the user's response to } a \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

$$M_{\lambda_{1..n}}(X_{i=1,...,n}) = \frac{\sum_{i=1}^{n} (\prod_{j=i+1}^{n} \lambda_j) X_i}{\sum_{i=1}^{n} \prod_{j=i+1}^{n} \lambda_j} \quad (5)$$

Note also that for $\lambda = 1$ we obtain an ordinary frequency distribution with no temporal decay.
M_λ has some standard properties of a mean (the proofs are straightforward):

$$M_{\lambda_{1..n}}(X + Y) = M_{\lambda_{1..n}}(X) + M_{\lambda=1..n}(Y) \quad (6)$$

$$M_{\lambda_{1..n}}(aX) = a M_{\lambda_{1..n}}(X) \quad (7)$$

$$\sum_b M_{\lambda_{1..n}}(p_{i=1..n}(b)) = 1 \text{ if } p_i \text{ are probability functions} \quad (8)$$

3 Self-Confidence with Insufficient Data

It is often assumed that the (un)certainty an agent can have about her knowledge is nothing but a meta-probability or meta-likelihood – cf. (Draper, 1995) for instance. On the other hand, there are researchers who argue against it (Kyburg, 1988; Wang, 2001). This seems to reflect the assumption that the meta-uncertainty should refer to the usability of the model. Indeed, meta-probability isn't very useful in this case: even if we know for sure that the model is slightly different from the reality (in consequence, its meta-probability is exactly 0), it *does* matter whether it's close to the real situation or not (Wang, 2001). This is also the perspective adopted in this paper. In this respect, some authors propose approaches based on some notion of error or fitting obtained through a posterior verification of the model (Hochreiter and Mozer, 2001; Spiegelhalter et al., 1998; Marshall and Spiegelhalter, 1999). However, the disconfidence studied here is *a priori* not *a posteriori* by definition – therefore any posterior reasoning can do no good here. In consequence, purely practical solutions may be very useful and work surprisingly well in particular situations (Kumar, 1998; Wang, 2001).

It has been suggested that, when the model is a probability distribution, the agent's self-confidence may be defined using the variance of the distribution treated as a random quantity itself (Pearl, 1987; Kyburg, 1988). Thus, the confidence measures being proposed and studied in this paper are based on the notion of aggregate variance of the estimator provided by the learning process.

3.1 Binding the Variance of Sampling

Handbooks on statistics like (Berry and Lindgren, 1996) suggest a way to determine whether an amount of data is enough to estimate the population mean EX with a sample mean $\bar{X}$: we assume some acceptable error level ε and as soon as the sampling deviation (standard deviation, for instance) gets below this value: $\sigma(\bar{X}) \leq \varepsilon$, we feel satisfied with the estimation itself. Since the real deviation value is usually hard to obtain, an upper bound or an estimation can be used instead.

If we want the 'satisfaction measure' to be continuous, it seems natural that the satisfaction is full 1 when the condition holds for $\varepsilon = 0$, and it decreases towards 0 as the dispersion grows. It is proposed here that the confidence for a frequency distribution $\hat{p}(\cdot|a)$ can be somehow proportional to $1 - \sum_b disp(b|a)$, and the variance $var(\hat{p}(b|a))$ is used to express the dispersion $disp(b|a)$. The reason for choosing the variance is that $0 \leq \sum_b var(\hat{p}(b|a)) \leq 1$ in our case, while the same is not true for the standard deviation σ as well as the mean deviation $m.a.d.$

We assume that the old observations are appropriate only partially with respect to the (cumulative) data decay encountered so far. Let $n \geq 1$ be an arbitrary number. By the properties of the variance and given that $Resp_1(b|a), ..., Resp_n(b|a)$ represent a random sampling of the user's responses:

$$\begin{aligned} var(\hat{p}_n(b|a)) &= var\Big(M_\lambda(Resp_{i=1..n}(b|a))\Big) = \\ &= var\Big(\frac{\sum_{i=1}^{n} Resp_i(b|a)\lambda^{n-i}}{\sum_{i=1}^{n} \lambda^{n-i}}\Big) = \\ &= \frac{\sum_{i=1}^{n} var(Resp_i(b|a))\lambda^{2(n-i)}}{(\sum_{i=1}^{n} \lambda^{n-i})^2} \end{aligned}$$

$var(Resp_i(b|a))$ is a population variance at the moment when the ith observation was made. If $p_i(b|a)$ was the real probability of user responding with action b at that particular moment, then:

$$\begin{aligned} var(Resp_i(b|a)) &= p_i(b|a) - p_i^2(b|a) \\ \sum_b var(\hat{p}_n(b|a)) &= \frac{\sum_{i=1}^{n} \lambda^{2(n-i)} \Big(\sum_b p_i(b|a) - \sum_b p_i^2(b|a)\Big)}{(\sum_{i=1}^{n} \lambda^{n-i})^2} \end{aligned}$$

$\sum_b p_i^2(b|a)$ is minimal for the uniform distribution $p_i(b|a) = \frac{1}{|ActB|}$, so:

$$\sum_b var(\hat{p}_n(b|a)) \leq \frac{\sum_{i=1}^{n} \lambda^{2(n-i)}}{(\sum_{i=1}^{n} \lambda^{n-i})^2}(1 - \frac{1}{|ActB|}) \quad (9)$$

Let

$$dispb(a) = \frac{\sum_{i=1}^{n} \lambda^{2(n-i)}}{(\sum_{i=1}^{n} \lambda^{n-i})^2}(1 - \frac{1}{|ActB|}) \quad (10)$$

$$Cbound(a) = 1 - dispb(a) \quad (11)$$

Now the confidence is never higher than it *can* be – the agent is playing it safe:

$$Cbound(a) \leq 1 - \sum_b var(\hat{p}_n(b|a)) \quad (12)$$

Note also that $dispb(a)$ is a decreasing function of λ for $\lambda \in (0, 1]$, so its value is always between $(1 - \frac{1}{|ActB|})/n$ (the value for $\lambda = 1$), and $1 - \frac{1}{|ActB|}$ (which is $\lim_{\lambda \to 0} dispb(a)$). Thus also

$$0 \leq \frac{1}{|ActB|} \leq Cbound(a) \leq \frac{n-1}{n} + \frac{1}{n|ActB|} \leq 1 \quad (13)$$

In the more general case when λ is variable:

$$\begin{aligned}\sum_b var(\hat{p}_n(b|a)) &= var\Big(M_{\lambda=1..n}(Resp_{i=1..n}(b|a))\Big) = \\ &= \frac{\sum_b \sum_{i=1}^{n} (\prod_{j=i+1}^{n} \lambda_j)^2 var(Resp_i(b|a))}{(\sum_{i=1}^{n} \prod_{j=i+1}^{n} \lambda_j)^2} \\ &\leq \frac{\sum_{i=1}^{n} (\prod_{j=i+1}^{n} \lambda_j)^2}{(\sum_{i=1}^{n} \prod_{j=i+1}^{n} \lambda_j)^2}(1 - \frac{1}{|ActB|})\end{aligned}$$

It is possible to compute

$$Lsqr_n = \sum_{i=1}^{n}(\prod_{j=i+1}^{n} \lambda_j)^2 = \lambda_n^2 Lsqr_{n-1} + 1$$

$$\text{and} \quad L_n = \sum_{i=1}^{n} \prod_{j=i+1}^{n} \lambda_j = \lambda_n L_{n-1} + 1$$

in an incremental way. Then the confidence can be defined as

$$Cbound(a) = 1 - \frac{Lsqr_n}{(L_n)^2}(1 - \frac{1}{|ActB|}) \quad (14)$$

which is never greater than $1 - \sum_b var(\hat{p}_n(b|a))$.

3.2 Adjusting the Confidence Value

The value of $dispb(a)$ proposed above can give some idea of the uncertainty the agent should have in $\hat{p}(\cdot|a)$. The most straightforward solution: $Cbound(a) = 1 - dispb(a)$ may not always work well for practical reasons, though. The agent can use a 'magnifying glass' parameter $\mathfrak{m}$ to sharpen her judgment:

$$Cbound(a) = (1 - dispb(a))^{\mathfrak{m}} \quad (15)$$

Since different learning methods show different dynamics of knowledge evolution, $\mathfrak{m}$ offers the agent an opportunity to 'tune' her confidence measure to the actual learning algorithm.

3.3 Forward-Oriented Distrust

In a perfect case we would be interested in the *real* variation of the sampling made so far – to have some clue about the expected (real) deviation from the estimation $\hat{p}_n$ obtained through the sampling. This value can be approached through its upper bound – as proposed in section 3.1. Alternatively we can try to approximate the variability we may expect from our estimator in the future (possibly with temporal discount).

It is worth noting that insufficient data can be seen as generating 'future oriented distrust': even if the agent's knowledge doesn't change much during the first few steps (e.g. the corresponding user's responses are identical) it may change fast in the very next moment. When the evidence is larger, the model of the reality being produced gets more stable and it can hardly be changed by a single observation. If we assume that the learning algorithm is correct – i.e. the model converges to the true user characteristics as the number of input data increases – then the agent can base her self-assessment on the possible future-oriented dispersion (possibly with a temporal discount Λ – to make the closer entries matter more than the farther ones):

$$\begin{aligned}Csize_\Lambda(a) &= (1 - fdisp_\Lambda(a))^{\mathfrak{m}} \quad (16)\\ fdisp_\Lambda(a) &= \lim_{k\to\infty} E\, fdisp_\Lambda^k(a) = \\ &= \lim_{k\to\infty} E\Big(\sum_b V_\Lambda(\hat{p}_{n+k}(b|a), ..., \hat{p}_n(b|a))\Big) \quad (17)\end{aligned}$$

where $\hat{p}$ is the agent's current model of the user, every $\hat{p}_{n+i}(\cdot|a), i = 1..k$ is obtained from $\hat{p}_{n+i-1}(\cdot|a)$ through response b_i^*, and the mean is taken over all the response sequences $(b_1^*, ..., b_k^*)$. The sample variance with discount/decay can be defined in a natural way as:

$$V_\Lambda(X) = M_\Lambda(X - M_\Lambda X)^2 \quad (18)$$

By properties (6), (7): $V_\Lambda(X) = M_\Lambda(X^2) - M_\Lambda^2(X)$. Assuming uniform a priori likelihood for all the possible sequences, the expected value can be approximated with simple averaging:

$$avg_{(b_{i=1..k}^*)}\, fdisp_\Lambda^k(a) = \frac{1}{|ActB|^k} \sum_{(b_{i=1..k}^*)} fdisp_\Lambda^k(a) \quad (19)$$

The limit in (17) can be then approximated iteratively for the generalized frequency counting presented in section 2. Let:

$$Mpsqr^k(a) \equiv avg_{(b_{i=1..k}^*)} \sum_b M_\Lambda(\hat{p}_{n+k}^2(b|a), ..., \hat{p}_n^2(b|a))$$

$$Msqr^k(a) \equiv avg_{(b_{i=1..k}^*)} \sum_b M_\Lambda^2(\hat{p}_{n+k}(b|a), ..., \hat{p}_n(b|a))$$

$$MP^k(a) \equiv avg_{(b_{i=1..k}^*)} \sum_b \hat{p}_{n+k}(b|a) M_\Lambda(\hat{p}_{n+k}(b|a), ..., \hat{p}_n(b|a))$$

$$Psqr^k(a) \equiv avg_{(b_{i=1..k}^*)} \sum_b \hat{p}_{n+k}^2(b|a)$$

Then for $0 < \Lambda < 1$

$$avg_{(b_{i=1..k}^*)}\, fdisp_\Lambda^k(a) = Mpsqr^k(a) - Msqr^k(a)$$

$$Mpsqr^k(a) = \frac{1-\Lambda^k}{1-\Lambda^{k+1}} Mpsqr^{k-1}(a) + \frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr^k(a)$$

$$\begin{aligned}Msqr^k(a) = &\frac{(1-\Lambda^k)^2}{(1-\Lambda^{k+1})^2} Msqr^{k-1}(a) + \frac{2(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} MP^k(a) \\ &- \frac{\Lambda^{2k}(1-\Lambda)^2}{(1-\Lambda^{k+1})^2} Psqr^k(a)\end{aligned}$$

fdisp ($\hat{p}$, a, Λ, λ, N, precision);

Iterative approximation of *fdisp*$_\Lambda(a)$. Returns the approximate value of the average future-oriented temporally-discounted dispersion (with temporal discount $0 < \Lambda < 1$). λ is the current temporal decay rate on the observation level, N represents the number of observations collected so far that takes into account the cumulative time decay; $\hat{p}$ is the current model of the user.

```
$Mpsqr, Msqr, MP, Psqr \leftarrow \sum_b \hat{p}^2(b|a)$;          /* initial values */
$k \leftarrow 0$;
$V \leftarrow 0$;
repeat
    $V_{old} \leftarrow V$;
    $k \leftarrow k+1$;
    $N \leftarrow N\lambda + 1$;
    $Psqr \leftarrow \left(\frac{N-1}{N}\right)^2 Psqr + \frac{2(N-1)}{|ActB|N^2} + \frac{1}{N^2}$;
    $MP \leftarrow \frac{(1-\Lambda^k)(N-1)}{(1-\Lambda^{k+1})N} MP + \frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr + \frac{1-\Lambda^k}{|ActB|(1-\Lambda^{k+1})N}$;
    $Msqr \leftarrow \frac{(1-\Lambda^k)^2}{(1-\Lambda^{k+1})^2} Msqr + \frac{2(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} MP - \frac{\Lambda^{2k}(1-\Lambda)^2}{(1-\Lambda^{k+1})^2} Psqr$;
    $Mpsqr \leftarrow \frac{1-\Lambda^k}{1-\Lambda^{k+1}} Mpsqr + \frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr$;
    $V \leftarrow Mpsqr - Msqr$;
until $|V - V_{old}| \leq$ precision;
return($V$);
```

Figure 1: The algorithm for iterative approximation of *fdisp*(a).

$$MP^k(a) = \frac{(1-\Lambda^k)(N_k-1)}{(1-\Lambda^{k+1})N_k} MP^{k-1}(a) + \frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr^k(a) + \frac{1-\Lambda^k}{|ActB|(1-\Lambda^{k+1})N_k}$$

$$Psqr^k(a) = \left(\frac{N_k-1}{N_k}\right)^2 Psqr^{k-1}(a) + \frac{2(N_k-1)}{|ActB|N_k^2} + \frac{1}{N_k^2}$$

where $N_k = N\lambda^k + \sum_{i=0}^{k-1}\lambda^i$, and $N = \sum_{i=1}^{n}\lambda_i$ is the actual (decayed) data size; $\lambda = \lambda_n$ is the current observation decay rate. The resulting algorithm is shown on figure 3.3. Moreover, the limit can be proved to exist, so the algorithm is convergent.

Proof: to prove the convergence of the sequence $V^k = avg_{(b^*_{i=1..k})}$ *fdisp*$^k_\Lambda(a)$, we will find an (a_k) such that $|V^k - V^{k-1}| \leq a_k$ for every k, and $\sum_{i=1}^k a_i$ forms a convergent series. Then the series $\sum_{i=1}^k (V^i - V^{i-1}) = V^k$ is also convergent.

Note that:

$$\begin{aligned}
&|V^k - V^{k-1}| = |Mpsqr^k - Mpsqr^{k-1} + Msqr^{k-1} - Msqr^k| = \\
&= |(\frac{1-\Lambda^k}{1-\Lambda^{k+1}} - 1)Mpsqr^{k-1} + (1 - \frac{(1-\Lambda^k)^2}{(1-\Lambda^{k+1})^2})Msqr^{k-1} \\
&+ \frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr^k - \frac{2(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} MP^k + \frac{\Lambda^{2k}(1-\Lambda)^2}{(1-\Lambda^{k+1})^2} Psqr^k| \\
&\leq |\frac{\Lambda^k(\Lambda-1)}{1-\Lambda^{k+1}} Mpsqr^{k-1}| + |\frac{\Lambda^k(2-2\Lambda-\Lambda^k+\Lambda^{k+2})}{(1-\Lambda^{k+1})^2} Msqr^{k-1}| \\
&+ |\frac{(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} Psqr^k| + |\frac{2(1-\Lambda)\Lambda^k}{1-\Lambda^{k+1}} MP^k| + |\frac{\Lambda^{2k}(1-\Lambda)^2}{(1-\Lambda^{k+1})^2} Psqr^k| \\
&\leq \frac{\Lambda^k(1-\Lambda)}{1-\Lambda^{k+1}} + \frac{\Lambda^k(1-\Lambda)(2-\Lambda^k(1+\Lambda))}{(1-\Lambda^{k+1})^2} + \\
&+ \frac{\Lambda^k(1-\Lambda)}{1-\Lambda^{k+1}} + \frac{2\Lambda^k(1-\Lambda)}{1-\Lambda^{k+1}} + \frac{\Lambda^{2k}(1-\Lambda)^2}{(1-\Lambda^{k+1})^2}
\end{aligned}$$

because $0 \leq Mpsqr, Msqr, MP, Psqr \leq 1$ by (8). Thus

$$\begin{aligned}
|V^k - V^{k-1}| &\leq \frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} + 2\frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} + \frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} \\
&\quad + 2\frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} + \frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} \leq \\
&\leq 7\frac{\Lambda^k(1-\Lambda)}{(1-\Lambda^{k+1})^2} \leq 7\frac{\Lambda^k(1-\Lambda)}{(1-\Lambda)^2} = \frac{7}{1-\Lambda}\Lambda^k,
\end{aligned}$$

QED. □

Note also that, for every k, $V^k \geq 0$ (because it's a sum of nonnegative elements); on the other hand $V^k \leq 1$ (because $V^k = Mpsqr^k - Msqr^k$, and $0 \leq Mpsqr^k, Msqr^k \leq 1$). In consequence:

$$0 \leq Csize_\Lambda(a) \leq 1 \tag{20}$$

The way both measures work has been studied through some experiments in section 4.

4 Simulations

The experiments were inspired by the following scenario: a software agent is designed to interact with users on behalf of an Internet banking service; she can make an offer to a user, and the user's response determines her output at this step of interaction. The banking agent is a 1-level agent, i.e. an agent that models other agents as stochastic (0-level) agents. The user is simulated as a random 0-level agent – in other words, his behavior can be described with a random probabilistic policy. The agent estimates the user's policy with a relative frequency distribution, counting the user's responses; at the same time she computes a confidence value for the profile acquired so far. 1000000 independent interactions (a sequence of 100 rounds each) with a random user process have been simulated; the average results are presented on the following charts.[1]

In the actual experiments the agent has had 3 possible offers at hand: the 'risky offer', the 'normal offer' and the 'safe offer', and the customer could respond with: 'accept honestly', 'cheat' or 'skip'. The complete table of payoffs for the game is given below. The 'risky offer', for example, can prove very profitable when accepted honestly by the user, but the agent will lose much if the customer decides to cheat; as the user skips an offer, the bank still gains some profit from the advertisements etc.

	accept	cheat	skip
risky offer	30	-100	1
normal offer	6	-20	1
safe offer	1.5	-1	1

Figures 2 and 3 show how the confidence values evolve for a static user (a user whose policy does

[1] only the output of the first 40 rounds is presented on most charts to emphasize the part where the main differences lie. The results for rounds $41 - 100$ were more or less the same.

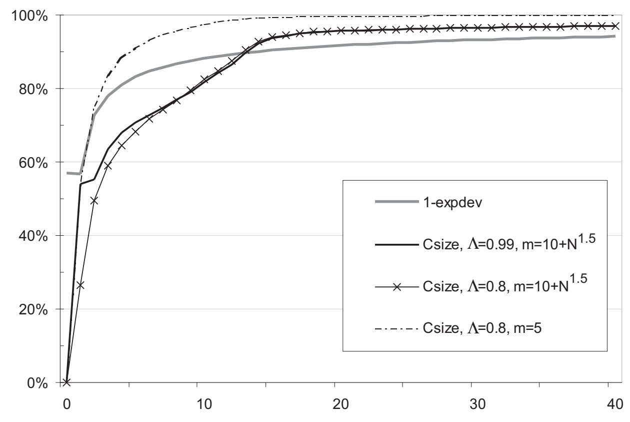


Figure 2: Confidence vs. accurateness: *Csize*

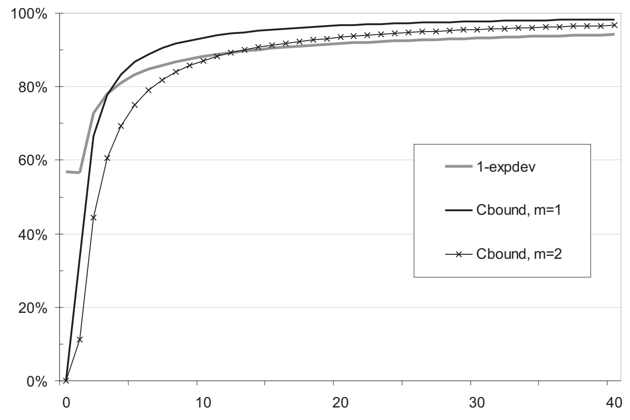


Figure 3: Confidence vs. accurateness: *Cbound*

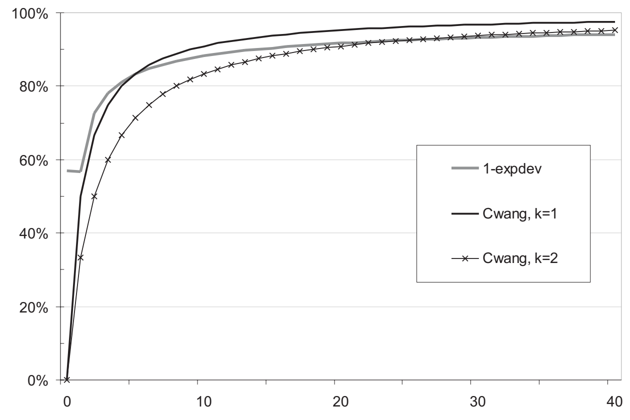


Figure 4: Wang's confidence for $k = 1$ and 2

not change throughout the experiment) and decay rate $\lambda = 1$, while figure 4 show the characteristics of the Wang's confidence *Cwang* $= N/(N + k)$. The confidence values are compared against the expected absolute deviation of the learned profile from the real policy of the user: $expdev = \sum_b |\hat{p}(b) - p(b)| \cdot p(b)$, or rather the 'accurateness' of the profile, i.e. $1 - expdev$.

The motivation behind the measures proposed here is that an agent can use several alternative models to make her decisions. If a numerical evaluation can be computed for every decision with respect to a particular model (the expected payoff, for instance), then the agent's decision may be based on a linear combination of the evaluations, with the confidence values providing weights. For example, the agent can use two models: the user's profile (frequency distribution computed from available data) and some default user model. If the agent trusts the user's profile in, say, 70% – the final evaluation may depend on the profile in 70%, and the remaining 30% can be derived from the default model. In consequence, the decision is based on both models at the same time, although in different proportions – weighting the partial evaluations with the confidence she has in them.

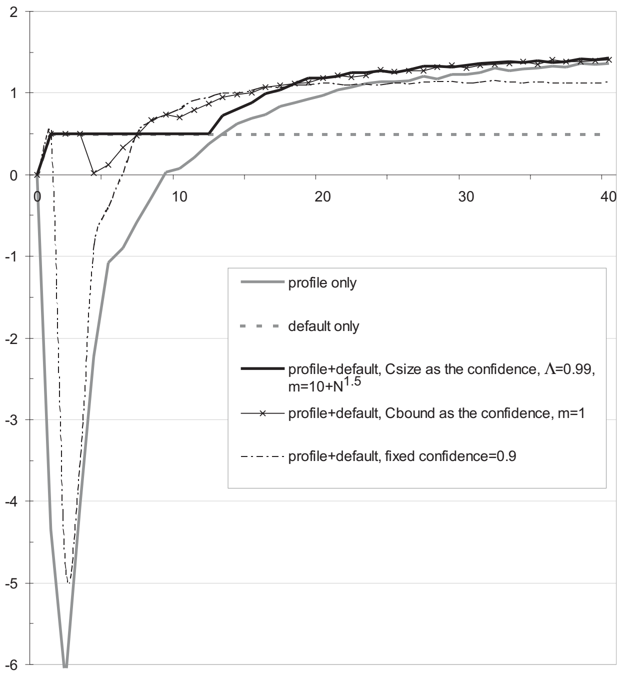


Figure 5: Hybrid agents vs. single-model agents: the average payoffs

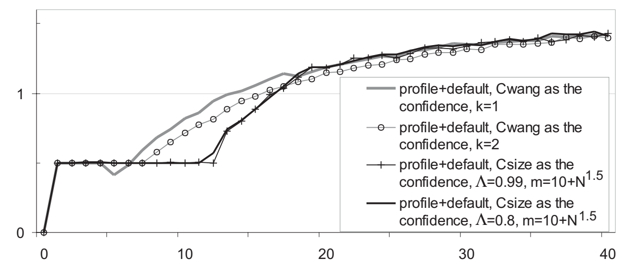


Figure 6: Hybrid agents vs. single-model agents: the average payoffs continued

The user's profile is computed as a plain frequency distribution. The default model, on the other hand, is defined in the Game Theory fashion: the user is assumed an enemy who always cheats. To get

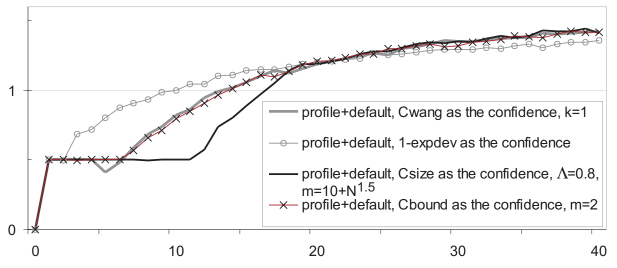


Figure 7: Hybrid agents vs. single-model agents: the average payoffs continued

rid of the exploration/exploitation tradeoff we assume also that the user is rather simple-minded and his response doesn't depend on the actual offer being made: $p(cheat)$, $p(accept)$ and $p(skip)$ are the same regardless of the offer (if he's dishonest, he cheats for a small reward as well as a big one, for instance). Now the agent can evaluate her actions with their expected payoffs. She computes the evaluation based on the user's profile: $e_p(a)$, and the evaluation based on the default model as well: $e_d(a)$; then she chooses the action with maximal value of $C \cdot e_p(a) + (1 - C) \cdot e_d(a)$. In consequence – as the charts show – the agent doesn't lose money at the beginning of an interaction (because C is low and therefore she's using mostly the default model). On the other hand, the confidence is almost 1 by the time the acquired knowledge becomes more accurate so the agent can start using the user profile successfully.

Figures 5, 6 and 7 show that an agent using such a hybrid model of the reality can be better off than an agent using either the profiles or the default user model alone.[2] *Cwang* (for $k = 1$) and *Cbound* (for $\mathtt{m} = 2$) give best results, while *Csize* fares slightly worse despite quite complicated parameters setting ($\Lambda = 0.8$ and variable $\mathtt{m} = 10 + N^{1.5}$). Experiments with other payoff tables gave similar results.

The measures presented here are primarily designed to tackle lack of data, not the user's dynamics. However, some experiments with dynamic users have also been run. Figure 8 presents the confidence evolution for a dynamic user and $\lambda = 0.95$. Figure 9 shows the results of the 'banking game' in the dynamic case. Here, the hybrid agent using *Cbound* fares best, with other hybrid agents close behind. Most notably, the *Cbound* agent is never worse than both single-model agents. The agent using only the default model is omitted on the chart to make it clearer: as before, her average payoff has been about 0.5 per round all the time.[3]

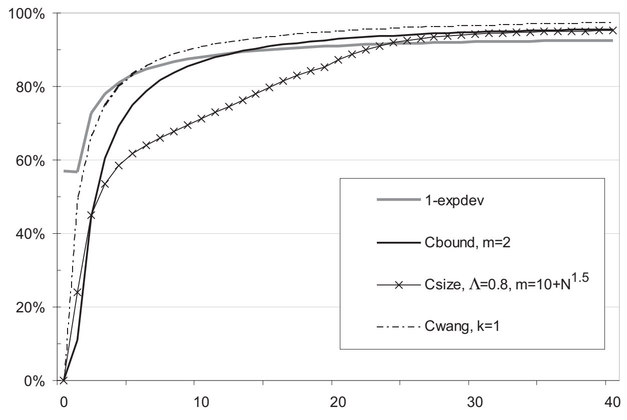


Figure 8: Confidence: interaction with a dynamic user, $\lambda = 0.95$

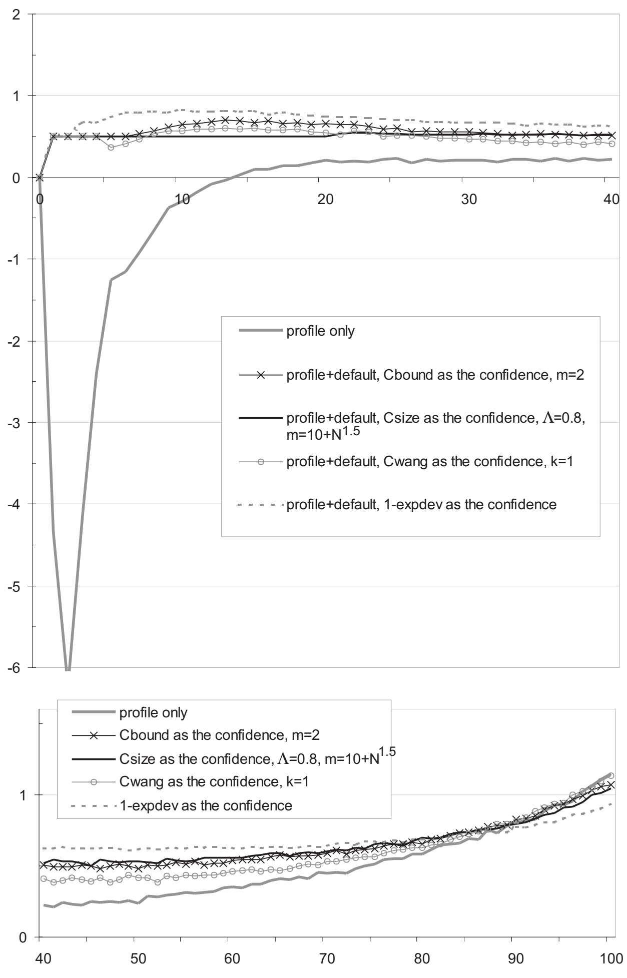


Figure 9: Hybrid agents vs. single-model agents: playing with a dynamic opponent, $\lambda = 0.95$

It should be obvious that the confidence needed to combine alternative models in the manner presented here is neither meta-probability nor meta-likelihood. The simulations suggest that the practical uncertainty concerned here is rather related to the distance/deviation of the model from the reality in a way.

[2] unfortunately, it isn't possible to present all the results on a single chart because the chart would be completely unreadable then.

[3] a similar chart for $\lambda = 1$ shows the same regularities, although the payoffs are generally worse because the learning method is less flexible.

Interestingly, an agent using $1 - expdev$ as her confidence measure gets positively best payoff in the initial phase of the interaction (and after that plays slightly worse) – see figures 7 and 9. Perhaps the expected absolute deviation isn't the best deviation measure for this purpose but it seems a close shot at least.

One issue should be made clear at the end, namely the way the dynamic users were simulated. It wasn't easy, because human users are hardly random with respect to their policies. True, humans' preferences drift – and the drift is never completely predictable – but neither is it completely chaotic. Real users are usually committed to their preferences somehow, so the preferences drift more or less inertly (the drift changes its direction in a long rather than short run). Here, random initial and final policies p_0, p_{100} were generated for every simulation, and the user was changing his preferences from p_0 to p_{100} in a linear way: $p_i(b) = p_0(b) + \frac{i}{100}(p_{100}(b) - p_0(b))$. It should be noted that the learning method used in the experiments (i.e. counting with decay) is *not* linear, so it's not true that this particular type of user simulation dynamics was chosen to suit the learning algorithm.

5 Conclusions

The experiments showed that the confidence measures can be useful – at least in some settings. Two measures have been proposed: *Cbound* and *Csize*, both based on the variance of the potential model evolution. In the simulations the agent using *Cbound* received slightly better results, especially for 'magnifying glass' parameter $\mathfrak{m} = 2$ (as long as the payoff was concerned). Moreover, the experiments with *Csize* revealed its important deficiency: in most cases a fixed $\mathfrak{m}$ proved too mild so the agent's confidence was reaching 1 almost at once (which is usually too fast). In consequence, the agent whose strategy was based upon *Csize* had to employ quite complicated 'tuning' scheme ($\mathfrak{m} = 10 + N^{1.5}$) plus an additional parameter Λ, while the agent using *Cbound* took (almost) the raw value of the bound. This suggests that the agent using *Csize* may expect serious problems with successful tuning of the confidence value in any real-life application, where the dynamics of the environment is changing, and the tuning process itself can hardly be cost-free.

The results of the simulations suggest that such practical uncertainty measure can be somehow based upon the estimated deviation of the model from the real state of affairs instead of the meta-probability of the model correctness or even the (potential) model variability over time. They show also that it may be captured approximately using very simple means: *Cwang* $= N/(N+1)$ for instance – in many domains of application.

The author would like to thank Mannes Poel for the discussions and all his suggestions.

References

D.A. Berry and B.W. Lindgren. 1996. *Statistics: Theory and Methods*. Wadsworth Publishing Company : Belmont, CA.

D. Carmel and S. Markovitch. 1996. Learning and using opponent models in adversary search. Technical Report CIS9609, Technion, Haifa.

D. Draper. 1995. Assessment and propagation of model uncertainty. *Journal of the Royal Statistical Society Series*, B(57):45–97.

S. Hochreiter and M.C. Mozer. 2001. Beyond maximum likelihood and density estimation: A sample-based criterion for unsupervised learning of complex models. *Advances in Neural Information Processing Systems*, 13. Proceedings of NIPS'2000.

G.J. Klir. 1999. Uncertainty and information measures for imprecise probabilities: An overview. In *Proceedings of the First International Symposium on Imprecise Probabilities and Their Applications*.

I. Koychev. 2000. Gradual forgetting for adaptation to concept drift. In *Proceedings of ECAI 2000 Workshop "Current Issues in Spatio-Temporal Reasoning"*, pages 101–106.

I. Koychev. 2001. Learning about user in the presence of hidden context. In *Proceedings of Machine Learning for User Modeling: UM-2001 Workshop*, pages 101–106. `http://www.dfki.de/∼rafer/um01-ml4um-ws/proceedings.html`.

S. Kumar. 1998. Confidence based dual reinforcement q-routing: an on-line adaptive network routing algorithm. Master's thesis, Department of Computer Sciences, The University of Texas at Austin. Tech. Report AI98-267.

H.E. Kyburg. 1988. Higher order probabilities and intervals. *International Journal of Approximate Reasoning*, 2:195–209.

E.C. Marshall and D.J. Spiegelhalter. 1999. Strategies for inference robustness in complex modelling: An application to longitudinal performance measures. Technical report, MRC Biostatistics Unit, Cambridge, UK.

J. Pearl. 1987. Do we need higher-order probabilities and, if so, what do they mean? In *Uncertainty in Artificial Intelligence Workshop*.

S. Sen and G. Weiss. 1999. Learning in multiagent systems. In Weiss G., editor, *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*, pages 259–298. MIT Press: Cambridge, Mass.

D.J. Spiegelhalter, N.G. Best, and B.P. Carlin. 1998. Bayesian deviance, the effective number of parameters, and the comparison of arbitrarily complex models. Technical report, MRC Biostatistics Unit, Cambridge, UK.

P. Wang. 2001. Confidence as higher-order uncertainty. In *Proceedings of the Second International Symposium on Imprecise Probabilities and Their Applications*, pages 352–361.

G. Widmer. 1997. Tracking context changes through meta-learning. *Machine Learning*, 27:256–286.

An approach to noncommunicative multiagent coordination in continuous domains

Jelle R. Kok Matthijs T. J. Spaan Nikos Vlassis
Intelligent Autonomous Systems Group, Informatics Institute
Faculty of Science, University of Amsterdam, The Netherlands*
{jellekok,mtjspaan,vlassis}@science.uva.nl

Abstract

Principled game-theoretic techniques exist for solving the problem of action coordination in a group of agents, however they typically suffer from an exponential blowup of the action space when many agents are involved. Coordination graphs (Guestrin et. al., 2002) offer tractable approximations via a context-specific decomposition into smaller coordination problems, and they are based on an iterative communication-based action selection procedure. We propose two extensions that apply when the agents are embedded in a continuous domain and/or communication is unavailable.

1 Introduction

Multiagent Systems (Weiss, 1999) is a relatively new field that has received considerable attention both in theory and applications. From an AI perspective, we can think of a multiagent system as a collection of agents that coexist in an environment, interact (explicitly or implicitly) with each other, and try to optimize a performance measure.

In this work we are interested in fully cooperative multiagent systems in which all agents share a common goal. A key aspect in such a system is the problem of *coordination*: how the individual agents can best choose their actions in order to successfully achieve a common goal (Boutilier, 1996).

Although in principle game theoretic techniques can be applied to solve the coordination problem (Osborne and Rubinstein, 1994), in practical situations involving many agents, even modeling an n-person game is intractable: the joint action space is exponentially large in the number of agents. However, one can often exploit the particular structure of a coordination problem in order to reduce its complexity.

A recent approach involves the use of a *coordination graph (CG)* (Guestrin et al., 2002a). This is a graph where each node represents an agent, and edges between nodes indicate that the corresponding agents have to coordinate their actions. In a context-specific CG (Guestrin et al., 2002b) the topology of the graph is dynamically updated based on the current context.

In this paper we extend a CG in two ways. First, we focus on agents that are embedded in a continuous domain (for example robotic agents in a soccer field) and are able to perceive their surroundings with sensors. For such a multiagent system, connectivity relationships between nodes in the coordination graph imply spatial relationships between agents, while the context is characterized by a continuous state variable. We propose a way to 'discretize' the context by appropriately assigning *roles* to the agents (Spaan et al., 2002) and then coordinating the different roles.

A second extension involves the way the agents compute their joint action. In the original formulation of CG, an agent needs to exchange information to and from its neighbors in order to compute its optimal action. This requires a communication channel which can be sometimes either unavailable or very costly to use. We propose a modification to the variable elimination algorithm of (Guestrin et al., 2002a) that allows each agent to efficiently predict the optimal action of its neighboring agents, making communication unnecessary.

The setup of the paper is as follows. In Section 2 we review the coordination problem, and in Section 3 we explain the concept of a CG. In

* This project is supported by PROGRESS—Dutch Technology Foundation STW project AES 5414.

	thriller	comedy
thriller	$1, 1$	$0, 0$
comedy	$0, 0$	$1, 1$

Figure 1: A coordination game.

Section 4 we describe our extensions, the role-dependent context and the noncommunicative case. In Section 5 we show some examples and in Section 6 we conclude and give hints for further research.

2 The coordination problem

We review here the agent coordination problem from a game theoretic point of view. A strategic game (Osborne and Rubinstein, 1994) is a tuple $(n, A_{1..n}, R_{1..n})$ where n is the number of agents, A_i is the set of actions of agent i and R_i is the payoff function for agent i. This payoff function maps the selected joint action $A = A_1 \times ... \times A_n$ to a real value: $R_i(A) \to \mathbb{R}$. Each agent independently selects an action from its action set, and then receives a payoff based on the actions selected by all agents. The goal of the agents is to select, via their individual decisions, the most profitable joint action.

A fully cooperative setting corresponds to a so-called coordination game in which all agents share the same payoff function $R_1 = \ldots = R_n = R$. Figure 1 shows an example of a coordination game between two agents. Each agent can choose between two types of movies, either a thriller or a comedy. They do not know in advance which movie the other agent will choose. Choosing the same movie results in an optimal joint action which offers them payoff 1, otherwise they receive payoff 0. It is clear that the agents have to coordinate their actions to maximize their payoff.

Formally, the coordination problem can be seen as the problem of selecting one out of many Nash equilibria in a coordination game. A Nash equilibrium defines a joint action $a^* \in A$ with the property that for every agent i holds $R_i(a_i^*, a_{-i}^*) \geq R_i(a_i, a_{-i}^*)$ for all $a_i \in A_i$, where a_{-i} is the joint action for all agents excluding agent i. Such an equilibrium joint action is a steady state from which no agent can profitably deviate given the actions of the other agents. For example, the strategic game in Figure 1 has two Nash equilibria corresponding to the situations where both agents select the same action.

There are several ways to solve a coordination game (Boutilier, 1996), for example by using communication or by imposing social conventions. The latter are constraints on the possible action choices of the agents. If we assume that the agents have the ability to identify one another, we can create a simple lexicographic convention using the following three assumptions:

- The set of agents is ordered.
- The set of actions of each agent is ordered.
- These orderings are common knowledge among agents (Geanakoplos, 1992).

The choice for an optimal joint action proceeds as follows. The first agent in the agent ordering chooses an optimal action (that corresponds to a Nash equilibrium) that appears first in its action ordering. The next agent then chooses its first optimal action in its action ordering given the first agent's choice. This procedure continues until all agents have chosen their actions. This general, domain-independent method will always result in an optimal joint action and moreover it can be implemented offline. During execution the agents do not have to explicitly coordinate their actions, e.g., via negotiation. If we would impose the ordering '$1 \succ 2$' (meaning that agent 1 has priority over agent 2) and 'thriller $\succ$ comedy' in our example, the second agent knows from the social conventions that the first will select the thriller and will therefore also choose the thriller.

In the above cases it is assumed that the Nash equilibria can be found and then coordination is the problem of selecting the same equilibrium. However, the number of joint actions grows exponentially with the number of agents, making it infeasible to determine all equilibria in the case of many agents. This calls for methods that first reduce the action space before solving the coordination problem. One such approach, explained next, is based on the use of a coordination graph that captures local coordination requirements between agents.

3 Coordination graphs

A coordination graph (CG) represents the coordination requirements of a system (Guestrin et

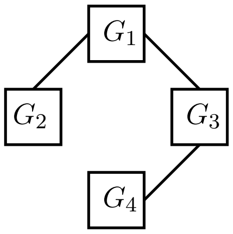


Figure 2: A CG for a 4-agent problem.

al., 2002a). A node in the graph represent an agent, while edges in the graph define dependencies between agents. Only agents that are interconnected have to coordinate their actions at any particular instant. Figure 2 shows a possible CG for a 4-agent problem. In this example, G_2 has to coordinate with G_1, G_4 has to coordinate with G_3, G_3 has to coordinate with both G_4 and G_1, and G_1 has to coordinate with both G_2 and G_3. Using such a graph, the global coordination problem can be replaced by a number of easier local coordination problems.

If the global payoff function can be decomposed as a sum of individual payoff functions, then solving for the joint optimal action can be done efficiently using a variable elimination algorithm (Guestrin et al., 2002a). The algorithm assumes an a priori elimination order that is common knowledge among the agents, and that each agent knows its neighbors in the graph (but not necessarily their payoff function which might depend on other agents). Each agent is 'eliminated' from the graph by solving a local optimization problem that involves only this agent and its neighbors: the agent collects from its neighbors all relevant payoff functions, then optimizes its decision conditionally on its neighbors' decisions, and communicates the resulting 'conditional' payoff function back to its neighbors. A next agent is selected from the list and the process continues. When all agents have been eliminated, each agent communicates its decision to its neighbors in the reverse elimination order.

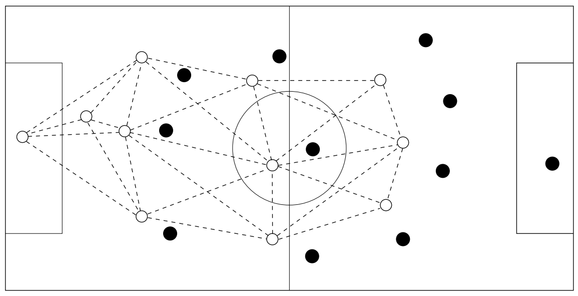
The local payoff functions can be matrix-based (Guestrin et al., 2002a) or rule-based (Guestrin et al., 2002b). In the latter case it is possible to use context-specific information to dynamically update the graph topology. A 'value rule' specifies how an agent's payoff depends on the current context, the latter being defined as a propositional rule over the state variables and the actions of the agent's neighbors. By conditioning on the current state the agents can discard all irrelevant rules, and this way the CG can be dynamically updated and simplified. Consider for example the situation where two plumbers have to fix the drainage system in a house. A value rule can specify that when the two plumbers are working in the same house they will get in each other's way, in which case the total payoff is decreased. In case the two plumbers are working in different houses, this value rule will not apply and the dependency in the graph is dynamically removed.

A limitation of this approach is that it is based on propositional rules and therefore only applies to discrete domains. Furthermore, in the variable elimination algorithm all coordinating agents must explicitly communicate their local payoff functions and their chosen actions using a message passing scheme. In the following we show how we can obtain context-specificity in a coordination graph when the agents reside in a continuous domain, and show how it is possible for each agent to predict the selected actions of its neighbors when communication is unavailable.

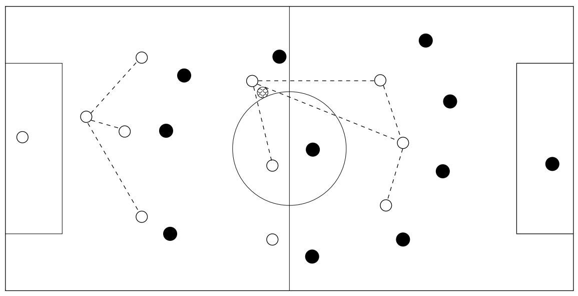
4 Coordination graphs in continuous domains

We are interested in problems where the agents are embedded in a continuous domain, have sensors with which they can observe their surroundings, and need to coordinate their actions. As a main example we will use the RoboCup simulation soccer domain (see (de Boer and Kok, 2002) and references therein) in which a team of eleven agents have to fulfill a common goal (scoring more goals than your opponent). Depending on the situation, certain agents on the field have to coordinate their actions, for example the agent that controls the ball must decide to which nearby agent to pass, etc. Such dependencies can be modeled by a CG that satisfies the following requirements: (i) its connectivity should be dynamically updated based on the current (continuous) state, (ii) it should be sparse in order to keep the dependencies and the associated local coordination problems as simple as possible.

We show an example of a continuous-domain



(a) Coordination graph.



(b) Reduced graph.

Figure 3: A coordination graph (a), and its context-specific reduction (b).

CG using the soccer domain. Figure 3(a) shows a picture of an a priori defined full coordination graph in which the dependencies between the teammates (represented by the open circles) are displayed. Based on the current context, e.g., the position of the ball in the field, the graph can be reduced as shown in Figure 3(b). The subgraph located in the left side of the field represents the relationship between the defenders trying to keep up a well-balanced defense. The subgraph on the right illustrates the local coordination game of the agent controlling the ball and the potential pass receivers. During the game, the coordination graph is continuously updated to reflect the current situation on the field.

Each node in such a CG has a natural 'location' within the domain while coordination dependencies automatically imply spatial relationships among agents. Moreover, contrary to the rule-based approach of (Guestrin et al., 2002b), the graph topology must depend on a context that is defined over a continuous state variable. In the above example, the context is based on the position of the ball which is a real variable having the soccer field as domain. We elaborate on this issue next.

4.1 Context-specificity based on roles

Conditioning on a context that is defined over a continuous domain is difficult in the original rule-based CG representation. A way to 'discretize' the context is by assigning *roles* to agents (Spaan et al., 2002). Roles are a natural way of introducing domain prior knowledge to a multiagent problem and provide a flexible solution to the problem of distributing the global task of a team among its members. In the soccer domain for instance one can easily identify several roles ranging from 'active' or 'passive' depending on whether an agent is in control of the ball or not, to more specialized ones like 'striker', 'defender', 'goalkeeper', etc.

Given a particular local situation, each agent is assigned a role that is computed based on a role assignment function that is common knowledge among agents. The set of roles is finite and ordered, so the most 'important' role is assigned to an agent first, followed by the second most important role, etc. By construction, the same role can be assigned to more than one agent, but each agent is assigned only a single role. Environment-dependent 'potential' functions can be used to determine how appropriate an agent is for a particular role given the current context. For details on the assignment of roles to agents see (Spaan et al., 2002).

Such an assignment of roles provides a natural way to parametrize a coordination structure over a continuous domain. The intuition is that, instead of directly coordinating the agents in a particular situation, we assign roles to the agents based on this situation and subsequently try to 'coordinate' the set of roles. For this, a priori rules exist that specify which roles should be coordinated and how.

As an example, consider again the left subgraph in Figure 3(b) involving four agents that organize the defense. The leftmost agent takes the role of sweeper while the other three all take the role of defender. It is common knowledge among the agents that the sweeper has to cover the space between the defenders and the goalkeeper to allow the defenders to advance to support the attack. As long as the four agents agree on their role assignment the problem of their coordination is simplified: the defenders only need to take into account the action of the sweeper in their strategy (apart from other factors such as the opponents) making sure it is the most retracted field player. In their local coordination game they do not need to consider other teammates such the goalkeeper or the attackers. Several other local coordination games could be going on at the same time (e.g., in the attack) without interfering with each other.

The roles can be regarded as an abstraction of a continuous state to a discrete context, allowing the application of existing techniques for discrete-state CGs. A particular assignment of k roles to a group of agents with the roles ordered according to their importance, can be regarded as instantiation of a discrete context variable that can take $O(k!)$ possible values, corresponding to all possible assignments of the roles to agents.

In practice, a simple hierarchical role assignment scheme can be used, for example the two roles 'active' and 'passive' can be first assigned based on who is in control of the ball, then among all 'passive' agents additional roles can be assigned like 'sweeper' or 'striker', etc. In other cases, a particular context may reduce the number of required roles to a manageable quantity. In soccer, for example, k is often equal to 2, which resembles the situation where a player needs to pass the ball to another player (see also Section 5).

Roles can reduce the action space of the agents by 'locking out' specific actions. For example, the role of the goalkeeper does not include the action 'score', and in a 'passive' role the action 'shoot' is deactivated. Such a reduction of the action space can offer computational savings, but more importantly it can facilitate the solution of a local coordination game by restricting the joint action space to a subspace that contains only one Nash equilibrium. For example, in Figure 1, if agent 2 is assigned a role that forbids him to select the action 'thriller' (e.g., because he is under 16), then agent 1, assuming he knows the role of agent 2, can safely choose 'comedy' resulting in coordination. Note there is only one Nash equilibrium in the subgame formed by removing the action 'thriller' from the action set of agent 2.

4.2 Non-communicating agents

Variable elimination in a CG requires that each agent first receives the payoff functions of its neighboring agents, and after computing its optimal conditional strategy it communicates a new payoff function back to its neighbors. Similarly, in the reverse process each agent needs to communicate its decision to its neighbors in order to reach a coordinated joint action. The elimination order is a priori defined and is common knowledge among the agents.

When communication is unavailable the variable elimination algorithm can still be used if we further impose the requirement that the payoff function of an agent i is common knowledge among all agents that are *reachable* from i in the CG. Since only agents that are reachable in the CG need to coordinate their actions, the second requirement in fact frees agents from having to communicate their local payoff functions during optimization.

Moreover, in the noncommunicative case the elimination order neither has to be fixed in advance nor has to be common knowledge among all agents as in (Guestrin et al., 2002a), but each agent is free to choose *any* elimination order, e.g., one that allows the agent to quickly compute its own optimal action. This is possible because a particular elimination order affects only the speed of the algorithm and not the computed joint action.

In summary, each agent i maintains a pool of payoff functions, corresponding to all payoff functions of the agents in its subgraph. Starting from itself, agent i keeps eliminating agents using an appropriate elimination order, until it computes its own optimal action unconditionally on the actions of the others. For each eliminated agent j, the newly generated payoff functions are introduced into the pool of payoff functions of agent i and the process continues. In the worst case, agent i needs to eliminate all agents $j \neq i$, for j reachable from i. Note that, although each agent computes its own action in a different way (during optimization the pool will look different for different agents), the resulting joint action will always be the optimal one.

In terms of complexity, the computational costs for each individual agent are clearly increased to compensate for the unavailable communication. Instead of only optimizing for its own action, in the worst case each agent has to calculate the action of every other agent in the subgraph. The computational cost per agent increases thus linearly with the number of new payoff functions generated during the elimination procedure. Communication, however, is not used anymore which allows for a speedup of the complete algorithm since these extra individual computations may now run in parallel. This is in contrast to the original CG approach where computations need to be performed sequentially.

Finally, we note that the common knowledge assumption is strong and even in cases where communication is available it cannot always be guaranteed (Fagin et al., 1995). In multiagent systems without communication common knowledge can be guaranteed if all agents consistently observe the same world state, but this is also violated in practice due to partial observability of the environment (a soccer player has a limited field of view). In our case, when the agents have to agree on a particular role distribution given a particular context, the only requirement we impose is that the role assignment in a particular local context is based on those parts of the state that are, to a good approximation, fully observable by all agents involved in the role assignment. For example, in the left subgraph of Figure 3(b) the particular role assignment may require that all four agents observe the position of each other in the field, as well as the positions of their nearby opponents, and have a rough estimate of the position of the ball (e.g., ensuring that the ball is far away). As long as such a context is encountered, a local graph is formed which is disconnected from the rest of the CG and can be solved separately, as explained above.

5 Experiments

We have applied the above ideas in our simulation robot soccer team (de Boer and Kok, 2002) with promising results. In the current phase we have not developed the CG framework to its full extent, but have tested it on simple situations where useful intuition can be gained.

We have implemented a simple role assignment function that assigns the role 'active' or 'passive' to a teammate based on whether it has

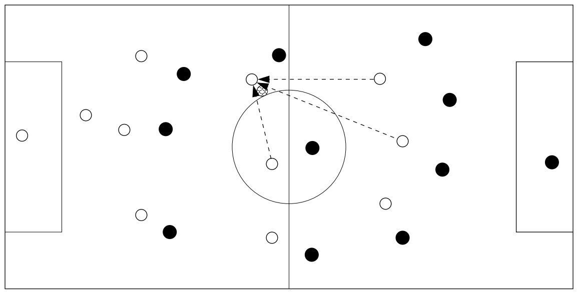


Figure 4: A simple situation involving one active and three passive agents.

the ball or not (for simplicity we focus here on the case where our team has the ball). At any instant only one agent is active and all the other 10 teammates are passive. Such a situation is shown in Figure 4 where one active and three passive teammates have to pairwise coordinate their actions.

Moreover, by construction an agent in a passive role always performs the same action, namely, moving towards its strategic position. The latter is computed based on the agent's home position (which is fixed throughout the game and known to all agents) and the position of the ball in the field which serves as an attraction point. As mentioned in section 4.1, such a drastic reduction of an action set greatly simplifies the local coordination game, because now the action choices of the three passive agents do not depend on the action choice of the active agent. In Figure 4 this is depicted by the directed edges between the agents.

Assuming the assignment of roles to the agents is common knowledge among reachable agents, the coordination problem resides now fully by the active agent. The latter has to choose one of the three teammates to pass the ball to, while we have assumed that the teammates follow their strategy independently of what the active or other passive agents do. Moreover, assuming that the active agent can also observe the position of the ball, it can predict the strategic position and thus the optimal action of each passive agent. The active player can now select to pass to the teammate that results in the highest future reward for the local coordination game; it will pass to the predicted position of the teammate with the maximum clearance from the opponents. Since the simula-

	With CG	Without
Wins	7	1
Draws	2	2
Losses	1	7
Avg. score	2.1	0.9
St. dev.	1.05	0.9

Table 1: Results of 10 games against ourselves, with and without CG.

tion server dynamics are known, predicting the one-step look-ahead reward is trivial (de Boer and Kok, 2002).

To test this approach we played games against ourselves, with one team using a CG and one team using no coordination at all during passing. In the latter case an active player would simply pass the ball to the last observed position of its teammate. Table 1 shows the results over the course of 10 full-length games. The results show that even the use of such a limited-scope CG has a positive effect on the performance on the team as a whole. Moreover, it turned out that the only statistically significant difference between the two teams was in passing. The successful passing percentage over these 10 matches was 80.12% for the team with the CG and 72.56% for the team without. These percentages indicate that due to the better coordination of the teammates, fewer mistakes were made when the ball was passed from one teammate to the other.

6 Conclusions and future work

We proposed two extensions to the framework of coordination graphs (Guestrin et al., 2002a) for the cases where the agents are embedded in a continuous domain and/or communication is unavailable. We argued that context-specificity is possible by appropriately assigning roles to the agents given a local situation. We also showed that we can dispense with communication if additional assumptions about common knowledge are introduced. We have not fully exploited the proposed framework in practice, but preliminary experiments in simulated soccer give promising results.

As future work, we first want to investigate the connotations of the common knowledge assumptions and how such knowledge can be obtained in practical situations. Second, we are interested in applying reinforcement learning techniques to a continuous-domain CG in order to learn the payoff functions in an automatic way, and we are looking for ways to efficiently plan ahead in a CG when an environment model is available. Finally, from an application point of view we want to apply the CG model to its full extent to the simulation RoboCup, where the agents need to continuously coordinate their actions, the context is time- and space-varying, and communication is restricted.

Acknowledgements

We thank the reviewers for their motivating comments. This research is supported by PROGRESS, the embedded systems research program of the Dutch organization for Scientific Research NWO, the Dutch Ministry of Economic Affairs and the Technology Foundation STW, project AES 5414.

References

C. Boutilier. 1996. Planning, learning and coordination in multiagent decision processes. In *Proc. Conf. on Theoretical Aspects of Rationality and Knowledge*.

R. de Boer and J. R. Kok. 2002. The incremental development of a synthetic multi-agent system: The UvA Trilearn 2001 robotic soccer simulation team. Master's thesis, University of Amsterdam, The Netherlands, February.

R. Fagin, J.Y. Halpern, Y. Moses, and M.Y. Vardi. 1995. *Reasoning about Knowledge*. The MIT Press, Cambridge, MA.

J. Geanakoplos. 1992. Common knowledge. *J. of Economic Perspectives*, 6(4):53–82.

C. Guestrin, D. Koller, and R. Parr. 2002a. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 14*. The MIT Press.

C. Guestrin, S. Venkataraman, and D. Koller. 2002b. Context-specific multiagent coordination and planning with factored MDPs. In *AAAI 8th Nation. Conf. on Artificial Intelligence*, Edmonton, Canada, July.

M. J. Osborne and A. Rubinstein. 1994. *A course in game theory*. MIT Press.

M. T. J. Spaan, N. Vlassis, and F. C. A. Groen. 2002. High level coordination of agents based on multiagent Markov decision processes with roles. In A. Saffiotti, editor, *IROS'02 Workshop on Cooperative Robotics*, Lausanne, Switzerland, October.

G. Weiss, editor. 1999. *Multiagent Systems: a Modern Approach to Distributed Artificial Intelligence*. MIT Press.

Model Growth

Jeroen van Maanen
Jeroen.van.Maanen@xs4all.nl

Abstract

The algorithmic method of inductive inference that Ray Solomonoff proposes in (Solomonoff, 1964) is not interactive. Marcus Hutter defines how to add interactivity to the inductive method based on the assumption that the environment supplies a utility function (Hutter, 2000). This paper discusses the possibility of a framework based on a utility function that is internal to the learning subject and independent of the environment. The internal utility function should measure the amount of information extracted from the interaction with the environment. The Minimum Description Length principle (MDL) proposed by Jorma Rissanen (Rissanen, 1989) supplies a framework that clearly separates a statistical model that represents the extracted information from the exact representation of the data. Algorithmic Statistics (Gács et al., 2001) should be able to bridge the gap between the algorithmic approach of Solomonoff and the statistical approach of Rissanen.

1 Introduction

Learning is not only about predicting the right answer to questions. The hard part of learning is often to ask the right questions. See also (van Maanen, 2002). Marcus Hutter defines how to add interactivity to the inductive method based on the assumption that the environment supplies a utility function. His reason for this set-up is that (Hutter, 2000)

> eventually we (humans) will be the environment with which the system will communicate and *we* want to dictate what is good and what is wrong, not the other way round.

If the context would be control theory this would be a reasonable assumption, but the context is artificial intelligence. Intelligent systems determine their own priorities. Even when the environment is highly optimised for teaching the teachers have to apply techniques that try to align the learning goals of the pupils with the teaching goals of the school. Even then, many students aim for average grades that let them pass major milestones with a more or less safe margin rather than maximal grades. They optimise a function that differs significantly from the externally supplied utility function.

What would be a more natural utility function for a learning system than its own learning rate? A learning system can be seen as a system that communicates with its environment and builds an internal model of that environment. The growth of that model, or the complexity of it, can be used as an exact measure for the learning rate of the learning system. When the results on Algorithmic Statistics (Gács et al., 2001) are applied to a slightly modified version of Hutters model called AIξ[1] we can expect to find that it is possible to define a universal autonomous learning system.

2 Example

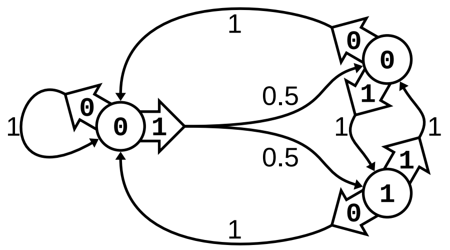
We will introduce an example of an environment that is stripped of all real-world complexities. This example is used in later sections to illustrate relevant aspects of formal models of learning. We use some abstract terminology to remind us where we would like to apply our results. With the *subject* we mean the person, animal or hypothetical device that learns. With the *environment* we mean the real or simulated

[1] AIξ is pronounced 'aixi' and can be written as 'AIXI' when Greek letters are not available.

environment that can be accessed by the subject. With the *interface* we mean the sensors and actors with which the subject interacts with the environment. The interaction between the subject and the environment results can be seen as a sequence of *events*. Each event is composed of an *action* that is performed by the subject and a *response* from the environment. We assume that actions as well as responses can be finitely encoded.

We will closely follow the notation used by Hutter. See (Hutter, 2000) for details and discussion. Without loss of generality we can assume that there are prefix-free sets of binary strings X and Y that encode responses and actions respectively. For our example we restrict ourselves to single bits: $X = Y = \mathbb{B}^1 = \mathbb{B} = \{0,1\}$. Strings over X are denoted $s = x_1 x_2 \cdots x_n$ with $x_k \in X$. $l(s) = l(x_1) + l(x_2) + \cdots + l(x_n)$ is the length of s. Analogous definitions hold for $y_k \in Y$. The elements of X and Y are called *words* rather than letters. The string $s = y_1 x_1 \cdots y_n x_n$ represents action/response pairs in chronological order. Due to the prefix property of X and Y, s can be uniquely separated into words. We further use the following abbreviations: ε is the empty string, $x_{n:m} = x_n x_{n+1} \cdots x_{m-1} x_m$ for $n \leq m$ and ε otherwise. $x_{<n} = x_{1:n-1}$. Analogous abbreviations are used for y. Furthermore we use $yx_n = y_n x_n$, $yx_{n:m} = yx_n \cdots yx_m$ and so on. We will disregard the encoding process and refer to yx_k as the k-th event. The string $yx_{<k}$ represents the entire *history* of events up to, but not including, the k-th iteration.

The example environment that is used in this paper acts like a Markov Decision Process (MDP). The behavior of the process is informally presented in the following diagram.



break Each circle in the diagram represents a possible state of the environment. The word that the environment produces when it enters a state is printed inside the circle. A wide arrow is originating from each state for every action that the subject can perform. The word corresponding to the action is inscribed in the arrow. The thin arrows originating from each action arrow represent possible state transitions of the environment. Each state transition is labelled with a probability. The sum of the probabilities of the state transitions that originate from an action arrow is unity.

The essence of this example is that it has two modes: stationary and alternating, and that the subject can switch between these modes at will. The subject will see only half of the environment if it chooses the same action for every cycle. If it acts randomly it will be hard to spot the regularities. Therefore we expect that a learning subject will exhibit some interesting behavior in this environment.

For a more formal treatment of this example we introduce a notation for probabilities of responses that is similar to Hutter's. Let $t \in (Y \times X)^*$, $y \in Y$, $x \in X$, and $p \in \mathbb{R}$, then $q(t\,y\,\underline{x}) = p$ means that for all sequences of events s the probability that the next response is x, given that the history is $s\,t\,y$, equals p. An underlined variable $\underline{x}$ represents a variable and other non-underlined variables represent conditions. The value is only defined when the conditions are sufficient to determine the probability. The response has to be independent of events that are more than $k - n$ cycles in the past. As the state of the environment depends only on the last few words of the history, it can be formally defined as follows:

$$\begin{array}{rcl} q(0\,\underline{1}) & = & 0 \\ q(001\,\underline{1}) & = & 0.5 \\ q(101\,\underline{1}) & = & 1 \\ q(111\,\underline{1}) & = & 0 \end{array}$$

The fact that the environment happens to act like a Markov Decision Process with a finite number of states is an artefact of the example. It does not imply any general constraints on the environment or the models that the subject constructs of the environment. For example, the example could be refined with a rule that depends on more data that gives more information about $q(001\,\underline{1})$. The new rule could depend on the last response on a 1-action and the number of preceding 00 events, *e.g.*,

$$\begin{aligned} q(10\,0^{2n}\,001\,\underline{1}) &= 1/2 + 2^{-n-2} \\ q(11\,0^{2n}\,001\,\underline{1}) &= 1/2 - 2^{-n-2} \end{aligned}$$

for $n \in \mathbb{N}$, where 0^{2n} represents a string of n times the 00 event. That would rule out a finite number of states for the process that generates the environment. The refinement could be done in such a way that, for simple strategies to generate the actions, the interaction with the original MDP would have the same stationary distribution as the interaction with the more complex model.

Analogous to probabilities of responses we introduce probabilities of actions: $p(yx_{<k}\underline{y_k})$ is the probability that the k-th action is x_k given that the history ends with $yx_{<k}$. If we look at the unmodified example we can analyse its behavior for three simple ways to choose the actions: $p(\varepsilon\underline{1}) = 0$ (always zero), $p(\varepsilon\underline{1}) = 1$ (always one), and $p(\varepsilon\underline{1}) = 1/2$. In the first case the interaction will consist of a constant sequence of pairs of zeros. In the second case the interaction will look like:

$$\cdots 10\;11\;10\;11\;10\;11 \cdots$$

In the third case the interaction can be described as a Markov Chain over events with the following stationary distribution:

event	00	01	10	11
probability	1/2	0	1/4	1/4

3 The algorithmic approach

Ray Solomonoff proposes to use the probability that a universal Turing machine reproduces the history as basis for establishing the likelihood of new data given the history (Solomonoff, 1964). This corresponds to Bayesian prediction using all effectively enumerable probability distributions over infinite strings as a hypotheses class using a universal prior. The prior is universal in the sense that it attributes a positive probability to all hypotheses in the class and that it dominates all possible effectively enumerable priors save for a multiplicative constant that depends on the pair of priors, but not on the hypotheses.

Marcus Hutter adds actions to this framework. In his AIξ model a learning system, a subject in our terminology, has to act upon its model of the environment. The model of the environment and the strategy for choosing actions are almost symmetrically described by conditional probability distributions over histories. The AIξ model is defined in terms of chronological Turing machines instead of the monotonous Turing machines used by Solomonoff. Chronological Turing machines alternately read one word from a one-way input-tape and write one word to a one-way output-tape. The words are chosen from prefix-free sets as described in Section 2. The symmetry between the model of the environment and the strategy for generating actions is broken in the AIξ model because only the output of the environment gets special treatment. It is split in an credit part and a residue. The credit defines the utility function that the subject has to optimise. Like Solomonoff the AIξ model fixes one universal chronological Turing machine U as a reference[2]. Every chronological Turing machine M that we would like to consider as a candidate for a model of the environment can be simulated on U. In other words: there exists a program q_M such that $U(q_M y_{<k}) = M(y_{<k})$. The function $\xi(q_M) = 2^{-l(q_M)}$ can be interpreted as the probability that U will use M to simulate the environment. Using ξ as a universal prior distribution over possible models of the environment and a utility function $C_{km_k}(p,q)$ the AIξ model defines the optimal action $\dot{y}_k$ as

$$\dot{y}_k = \underset{y_k}{\text{maxarg}} \max_{p \in P_k(y_k)} \sum_{q \in Q_k} \xi(q) \cdot C_{km_k}(p,q)$$

where $P_k(y_k) = \{p \in \mathbb{B}^* \mid U(p\;\dot{x}_{<k}) = \dot{y}_{<k} y_k\}$ and $Q_k = \{q \in \mathbb{B}^* \mid U(q\;\dot{y}_{<k}) = \dot{x}_{<k}\}$. The dots above $x_{<y}$ and $y_{<k}$ indicate that these are actual values that have been exchanged between the subject and the environment. The utility function $C_{km_k}(p,q)$ computes the credits that the subject would receive on cycles k to m_k when it uses strategy p and when q is a perfectly accurate model of the environment. The AIξ model converges to models where $\xi(q)$ is replaced by another recursively enumerable prior on (programs for) chronological Turing machines. We need knowledge about the environment for a

[2]The prefix-free set that defines the words on the input tape has to be extended to allow for a program to be read before or together with the first word of the simulated input

better prior as well as for an optimal universal Turing machine. Given that we are studying learning, *i.e.*, processes that increase knowledge, all prior knowledge is dangerous. We could forget to measure it beforehand and count it as obtained by the subject afterwards. Prior knowledge can also reduce a potential learning situation to a mere search for some global maximum. Therefore we cannot expect subjects that have to *learn* about the environment while interacting with it to do much better than the AIξ model

Application to the example

To apply Hutter's approach to the example we need to specify a credit function for the output words that the environment can produce. As $X = \mathbb{B}$ every possible credit function is equivalent with either $C(x) = x$ or $\bar{C}(x) = 1 - x$.

If the credit function is $C(x) = x$ a subject that uses the AIξ model will find out quickly that the best strategy is to always choose the 1-action. Likewise, if the credit function is $\bar{C}(x) = 1 - x$ a subject that uses the AIξ model will find out quickly that the best strategy is to always choose the 0-action. In either case no further information is gathered about the transitions between the left and the right half of the state diagram. The difference between the simple example and the extended example will never become apparent in the history. If we want to 'teach' the complete environment to the subject, we need to extend it in such a way that the subject can 'tell' the environment what its model is, *i.e.*, include q_M in y_k. Then we would have to let the environment evaluate the accuracy of M and return a credit that is based on both the accuracy of M and the length of q_M. That is quite a burden to place on the environment.

4 The Minimum Description Length approach

Rissanen's Minimum Description Length principle (MDL) is explicitly formulated in terms of hypotheses (Rissanen, 1989). The process of revising evaluations of hypotheses is clearly a form of inference. MDL is based on reproducing a sequence of symbols exactly, just like Solomonoff's inductive inference procedure. MDL assumes that we are given a hypotheses class $\mathcal{C}$. Every hypothesis in the given class is a distribution over all infinite sequences of symbols from our alphabet. MDL also requires that every hypothesis H in the hypotheses class can be finitely described by a code c_H.

For learning subjects the hypotheses are functions that compute the conditional probability of responses of the environment given actions by the subject. Using the Kraft inequality (Li and Vitányi, 1993) we can construct an encoding $e_{H,y_{<k}}$ for finite sequences of symbols such that for each finite history $y\!x_{<k}$ we have

$$l(e_{H,y_{<k}}(x_{<k})) \approx -\log_2 H(y_1\underline{x}_1 \cdots y_{k-1}\underline{x}_{k-1})$$

Now for each hypothesis H a transcription of the symbols that were exchanged on the interface $y\!x_{<k}$ can be described by concatenating c_H, $y_{<k}$ and $e_H(y\!x_{<k})$. According to MDL the best hypothesis is the hypothesis that minimizes the length of this description. As y_k is given and thus fixed we can therefore minimize

$$l(c_H) + l(e_{H,y_{<k}}(x_{<k}))$$

Application to the example

We could take all Markov Decision Processes with finitely many states and probabilities that are rational numbers as our model class $\mathcal{C}$. Each model can be described as follows:

$$\begin{array}{c} n \\ x_{s_1} p_{0,s_1,s_2} \cdots p_{0,s_1,s_n}\ p_{1,s_1,s_2} \cdots p_{1,s_1,s_n} \\ \vdots \\ x_{s_n} p_{0,s_n,s_1} \cdots p_{0,s_n,s_{n-1}}\ p_{1,s_n,s_1} \cdots p_{1,s_n,s_{n-1}} \end{array}$$

where n is the number of states, x_{s_i} is the response of the process when entering state s_i, and p_{y_t,s_i,s_j} is the probability of a state change to s_j given state s_i and action y_t. The probability that the state will not change given an action can be found by subtracting the probabilities of the non-trivial state changes from unity.

A simple way to specify a number $n \in \mathbb{N}$ in a binary alphabet in such a way that the set of codes of numbers is prefix-free is to precede the binary notation of $n+1$ with $\lfloor \log_2 n + 1 \rfloor$ times a 0 bit. As the binary representation of a nonzero natural number always starts with a one bit we can recover a number from a string by the following procedure. First we count the leading zeros and read that many digits after the leading one bit. Then we decode the binary sting that consists of all the bits we read, the initial zeros don't harm, and subtract one. Fractions are encoded by specifying numerator and denominator.

If we follow these rules for the generating model of the example we have three states and twelve fractions to encode. The number three is encoded in five bits. Six of the fractions are zero and are encoded in two bits each. The other fractions are encoded in six bits each. The length of the model totals $5 + (1 + 2 \times 2 + 2 \times 6) + 2 \times (1 + 2 \times (2 + 6)) = 56$ bits. The trivial model requires $3 + 2(1 + 2 \times 1 \times 6) = 29$ bits. If the actions are determined by coin tosses, the generation model can be expected to be better than the trivial model after n action/response pairs, where:

$$56 + \frac{3}{4}n < 29 + n$$

So $n > 108$. It should be clear much earlier that choosing 1 actions leads to an earlier expected model growth. This can be seen by evaluating models that compress the data (temporarily disregarding the length of the encoding of the model) and evaluating the probability that such a model will become the MDL model given that it fits and given a potential strategy. If, for example, it is established that x_k is always zero when y_k is zero when $k = 10$, then the generating model becomes a candidate for the MDL hypothesis for $k = 66$ if all actions are chosen to be 1.

After the discovery the subject will continue to search for possible extensions of the model. The subject will alternate between sequences of zeros and ones of varying length to search to explore $q(001\underline{x})$. If the example is extended as described in Section 2 it will find longer and longer finite truncations of the infinite Markov Decision Process and the model that the subject has of the environment will converge to the true model.

The most important aspect of the algorithmic approach that is lacking in the MDL approach is universality. So far we have discussed a hypotheses class that contains models that are arbitrarily close to the 'true' behavior of the environment. That means that the hypotheses class contains hidden information about the environment. We would rather have the subject learn the fact that the environment acts like a Markov Decision Process by itself. Analogous to the algorithmic approach the hypotheses class should consist of all recursively enumerable semi-measures on the set of infinite binary strings. With a hypotheses class that contains all regularities that can be effectively described we could plug MDL into Hutter's framework and define a universal autonomous learning system.

5 Algorithmic statistics

In the algorithmic approach the subject evaluates descriptions of the total known history of the interaction with its environment. The lengths of these descriptions are used to determine the relative predictive value of the algorithmic processes behind those descriptions. In the MDL approach descriptions of the history consist of two parts: a description of a hypothesis and a description of the history given this hypothesis. This separation of the description in two parts is relevant, because we want to use the length of the first part to guide the actions of the subject. Murray Gell-Mann and Seth LLoyd discuss a similar separation of a description of a finite binary string into two parts (Gell-Mann and LLoyd, 1996). The first part of their description defines a set of strings that describes the regularities in the given string. This set is chosen in such a way that the given string is a 'typical' member of the set. The second part of the description pinpoints given the string in this set. They call the length of a smallest description of a string its total information and the length of a smallest description of a set, such that the string is a typical member of that set, its effective complexity. In their conclusion Gell-Mann and LLoyd even make the connection to learning systems. Given our observation about the MDL approach we come to a different analysis of the importance of these information measures to learning. A small total information just means that there are regularities to be found in the history, but if the effective complexity rises as words are added to the history the subject learns in the sense that new regularities are added to its description of the environment. Therefore the growth rate of the effective complexity of the history would be a good measure for the learning rate of the subject.

In their article about algorithmic statistics Péter Gács, John Tromp, and Paul Vitányi analyse the relation between data and models in depth (Gács et al., 2001). Although their conclusions about the practical applicability of the universal information measures they found are sobering, it would still be worthwhile to investigate whether it is possible to avoid uncomputability limits. This could be done, for example, by introducing a cut-off on the computation time necessary to produce the history from its description. The universal information measures can also be used to prove bounds on the learning speed of subjects that use randomness to sample the hypotheses space that might be hard to prove otherwise.

6 Conclusion

To remove the external utility function from Hutter's AIξ model while preserving its universality we can take the following steps.

1. Use algorithmic statistics to define a universal form of MDL.
2. Use universal MDL models instead of the raw descriptions of the history.
3. Evaluate actions based on expected model growth instead of the externally supplied utility function.

Further research

The steps above should be formalized. The resulting formal model should be studied to find bounds on learning speed and subjectiveness. To determine the subjectiveness of the model we must ask the question: how far can universal subjects based on different universal machines be apart when interacting with the same environment? This question is especially interesting because we need to define what it means for two subjects to interact with the same environment. As the subjects act differently they might learn different things about the environment and face entirely different questions rather quickly. Another important direction would be to construct practicable algorithms that approach the universal model as closely as possible.

References

Péter Gács, John T. Tromp, and Paul M.B. Vitányi. 2001. Algorithmic statistics. *IEEETIT: IEEE Transactions on Information Theory*, 47. `http://citeseer.nj.nec.com/gacs01algorithmic.html`.

Murray Gell-Mann and Seth LLoyd. 1996. Information measures, effective complexity and total information. *Complexity*, 2:44–52.

Marcus Hutter. 2000. A theory of universal artificial intelligence based on algorithmic complexity. Technical report. 62 pages, `http://xxx.lanl.gov/abs/cs.AI/0004001`.

Ming Li and Paul M.B. Vitányi. 1993. *An Introduction to Kolmogorov Complexity and its Applications*. Springer Verlag.

Jorma J. Rissanen. 1989. *Stochastic Complexity in Statistical Enquiry*. World Scientific, Singapore.

Ray J. Solomonoff. 1964. A formal theory of inductive inference, part I. *Information and Control*, 7:1–22.

Jeroen van Maanen. 2002. Towards a formal theory of learning systems. Unpublished. `http://www.sollunae.net/zope/wiki/EnglishSummary`.

Context-based policy search: transfer of experience across problems

Leonid Peshkin *pesha@ai.mit.edu*
Harvard Center for Artificial Intelligence
Dworkin Bld. 134, Cambridge, MA 02138

Edwin D. de Jong *edj@cs.vu.nl*
Artificial Intelligence Department
Vrije Universiteit Amsterdam

Abstract

An important question in reinforcement learning is how generalization may be performed. This problem is especially important if the learning agent receives only partial information about the state of the environment. Typically, the bias required for generalization is chosen by the experimenter. Here, we investigate a way for the *learning method* to extract bias from learning one problem and apply it in subsequent problems. We use a gradient-based policy search method, and look for controllers that consist of a context component and an action component. Empirical results on a two-agent coordination problem are reported. It was found that learning a bias made it possible to address problems that were not solved otherwise.

1. Introduction

Reinforcement learning problems with large state spaces typically require the use of generalization. Any form of generalization implies that choices must be made regarding the similarity of different situations, and any such choices constitute a bias. The success of a particular method for generalization depends on whether this bias is appropriate for the problem at hand [12].

For standard *Markov Decision Processes* (MDPs) the observation of the learner captures all relevant information about the state of the environment. In *Partially Observable* MDPs (POMDPs), the choice of the appropriate action may in principle depend on all previous observations. Thus, the learner has to extract relevant information from the history of its interactions with the environment. This renders the need for appropriate generalization even more pressing.

While in general it may not always be possible to recover all required information about the state of the environment from observations, a certain class of POMDPs can be transformed into MDPs by distinguishing among different *contexts* in which the agent may find itself, and providing the current context as an extra input. A context here refers to a subset of the possible histories of interaction between the agent and its environment, which consist of observations, actions, and rewards. If contexts with the above property can be extracted from the interaction history, then the remaining problem can be addressed using the standard *reinforcement learning* methods. For information about reinforcement learning in general the reader may consult [15, 16, 10].

For problems having the above structure, the optimal policy can be described as a set of behaviors, each of which corresponds to a particular context. In this paper, we suggest that this structure has a potential for meta-learning. If related environments require similar behaviors, while differing in their correspondence between contexts and input observations, then behaviors can be transferred from previously solved problems to more difficult variants of those problems. Thus, we have identified a particular mechanism for extracting useful bias from related problems (see e.g. [5, 4, 2, 3, 6]), and a corresponding class of problems which could benefit from this mechanism. We investigate this method for a simple case, and report empirical results.

[0]Originally presented at: ICML 2002 Workshop on the Development of Representations, see `http://demo.cs.brandeis.edu/icml02ws/`. Current version and related work available from `http://www.eecs.harvard.edu/~pesha/papers.html`

2. Architecture and Learning Mechanism

Partial observability of the environmental state requires the policy representation to include some form of memory in order to specify the optimal policy. In our case, this is achieved by using a finite state controller (see e.g. [11]). Furthermore, in order to *find* optimal policies for partially observable problems, the learning method must be able to search for policies that use memory. To this end, we employ a gradient-based method for policy search, see [13, 1] for the general method and [14] for the discussion of cooperation without explicit coordination in agents controlled by FSCs.

The main idea of this paper is to search for policies that can be described as a set of contexts and corresponding behaviors, so that the behaviors can be transferred between problems. To make this possible, we construct the agent from two components, see figure 1. The first component, called the *context component*, determines the current context from the interaction history. In general this history may include past observations, actions and rewards. In this particular problem, the context component uses the current observation and the context at the previous time step to determine the current context. The *action component* receives the current context and, optionally, the observations as input, and specifies a behavior or partial policy for each context.

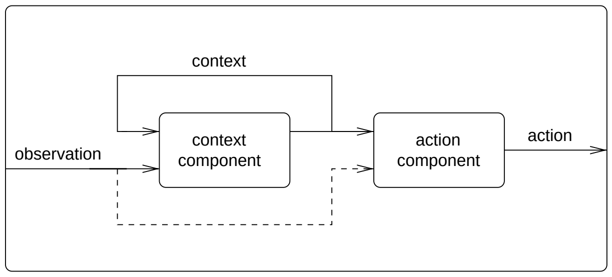


Figure 1. Proposed architecture. The *context component* determines the current context from the previous observation and context. The *action component* uses this information, possibly in combination with the current observation, to produce a corresponding behavior.

Separating the context and action components in the representation of a policy allows these components to be learned separately. Other examples of taking advantage of such controller structure are presented in [9] and [7, 8]. The former uses a controller consisting of two disjoint neural networks, while the latter demonstrates how to use the natural continuity of the Euclidean coordinate system to generalize over the observation space.

In order to transfer experience, we find an optimal controller on a small instance of the problem at hand, and retain the action component of the controller when scaling up to a larger instances of the problem. This provides a natural way to incorporate bias into the learning process.

The finite state controller will now be described in more detail. The controller consists of an internal state transition function that determines the context from observations, and an action function that associates each context with a corresponding behavior.

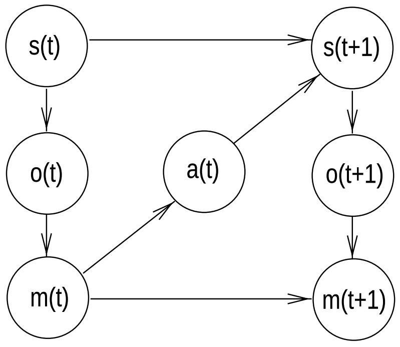


Figure 2. An influence diagram for agent with FSCs in POMDP, showing the relations between environmental states (s), observations (o), internal states (m), and actions (a) at subsequent time steps.

The complete controller for an agent with action space A and observation space O is a tuple $\langle M, \mu_a, \mu_m \rangle$, where M is a finite set of internal controller states (contexts); $\mu_m : M \times O \to \mathcal{P}(M)$ is the internal state transition function that maps an internal state and observation into a probability distribution over internal states; and $\mu_a : M \to \mathcal{P}(A)$ is the action component that maps an internal state into a probability distribution over actions. We assume that both the internal state transition function and the action function are stochastic, that their derivatives exist, and that these are bounded away from zero. Figure 2 depicts an influence diagram for an agent using this controller.

We consider series of problems that require similar behaviors. Thus, by retaining the part of the policy μ_a that specifies the behaviors learned on a small problem, a useful starting point for addressing larger but similar problems is obtained. The context component μ_m on the other hand must be learned anew, since the particular observations that identify each context may vary across different environments. In the following sections, we first describe the task in detail, then experiments are reported that illustrate the advantage of aforementioned learning with bias.

3. Task Description: Block Moving

The metaphor for the task used in the experiments is *"block moving"* which is a multi-agent problem derived from the "load-unload" problem [11, 13]. It requires the cooperation of two agents and constitutes a *coordination problem* in the sense that the agents need to adjust their actions to one another. This produces a non-trivial form of partial observability. In the real-world version of this problem, the activity of the two people lifting a heavy block should be synchronized for satisfactory results, and thus requires coordinating the moment at which to start moving. The information about the state of each agent is exchanged through (possibly non-verbal) communication.

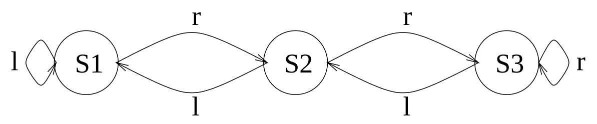


Figure 3. State transition function. The diagram shows three states: *load* (S3), one intermediate state (S2), and *unload* (S1)

In the simple, abstract task we study here, the process is modeled as follows. Each agent can move independently between several locations (see figure 3 for an example involving three locations). When two agents are in state S3, they automatically load a block. From then on, they must move in synchrony in order not to drop the load. Since the agents perceive only their own location and not that of the other agent, communication is necessary.

The reward function for the task is depicted in figure 4. The states in this diagram are collective states, specifying the state of both the first and second agent (A1 and A2). Connections without arrows mark transitions that can have either direction. The projection of the diagram onto either horizontal axis yields the state transition diagram for a single agent shown above in figure 3.

The reward function not only depends on the current state of both agents, but also on a history of both of their states. The only aspect of this history that is relevant to the reward function is whether the load has been lifted and not dropped since it was lifted. The two collective states with this property are in the upper level of the diagram, marked *loaded*. All remaining collective states are at the lower, unloaded level.

A potential for communication is provided as follows. In addition to the action an agent can select to move

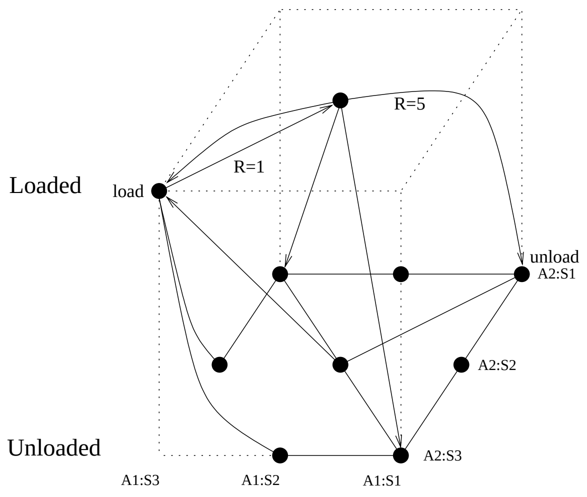


Figure 4. The reward structure for "block moving" domain.

between states, each agent has a single bit that it can set or reset. Each agent reads the bits of all other agents, in addition to the sensor information specifying its own location. If both agents would always start in state $S1$, they could learn to move to state $S3$ as quickly as possible, and return to $S1$ from there. This ballistic policy would cause them to move in synchrony, without any need for communication. In order to disallow this strategy, the initial position of each agent is selected randomly from all locations except "load".

For the three-location instance depicted in figure 3 the optimal strategy has the following form. Each agent moves to state $S3$, informs the other agent of its presence there and, if necessary, waits for the other agent to arrive. Then each agent moves to $S2$, and from there back to $S1$. If both agents execute this policy, they move together after picking up their load. Thus, by the use of communication, the agents can synchronize their behavior and collect the maximum amount of reward.

Part of the difficulty of this task is that it requires establishing a convention. While it is clear that the only possibility for communication is for each agent to set or reset its bit, the sending and receiving agent must converge on the same choice of which setting to use for which case. Since both agents are equivalent and may function both as sender and as receiver, this problem needs to be solved twice (not necessarily in the same way). A greater difficulty than the selection of such a convention however is that of settling on sending and receiving behavior simultaneously; while the first agent to arrive can only determine whether the second agent has arrived through communication, the second

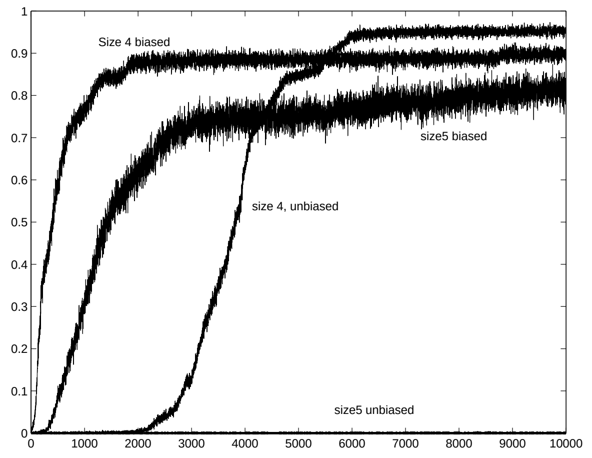


Figure 5. Empirical comparison of biased and unbiased learning by policy search in FSCs for the "block lifting" environment of various size.

agent can only discover that signaling this information is useful if the first agent acts on it, using the same convention. The necessity to simultaneously arrive at compatible communication and action policies makes the *block moving* problem a challenging one.

The difficulty of the problem strongly depends on the size of the problem instance. A version of the problem with three states has been described, involving a "load" state (S3), an intermediate state (S2), and an "unload" state (S1). To obtain larger versions of the problem, we introduce one or more additional intermediate states. Since the part of the policy space that must be considered grows exponentially with the length of the smallest optimal policy, even adding one or two states makes the problem considerably more difficult.

4. Empirical Results

In our experiments we begin by learning the optimal policy with no initial bias in the instance of the problem with three locations all together, counting **load** and **unload** locations. For this trivial instance of the domain even an exhaustive evaluation of all policies would be feasible, so it comes as no surprise that agents rapidly converge to a good policy. The resulting action function is used as a bias - a starting point for action functions in learning policies for larger instances of the domain.

Figure 5 illustrates the advantage of learning with bias in our domain. All plots are averaged over 10 runs. The learning rate is kept constant. A policy is learned in the space of three-state finite space controllers. For the domain with four locations, both biased and unbiased learning converge, but unbiased does so after a significant delay. For the domain with five locations unbiased learning never picks up for the trial length used. Using the bias learned on a smaller version of the problem makes it possible to learn even on this difficult instance, and turns out to be even more efficient than unbiased learning for a smaller domain with four locations.

In the experiments, the action function from a smaller problem instance was used as a starting point in the learning process, and could in principle be modified. In practice, we found that the action component did not change, and only the context component was learned anew.

5. Discussion

The signal from the other agent in our environment formed a part of the observation. A potential change in the representation of the controller would be to separate the location from the signal and feed the signal part of the observation directly into the action function. This would be justified by the fact that as we increase the size of the domain, the number and the semantics of the messages does not change. The location numbering on the other hand does change, and needs to be learned for every instance. Another alternative approach to the approach that has been investigated would be to fix the action function and only learn internal state transition function. We plan to experiment with some of these variants of the setup in future experiments.

In a sense, an agent that develops a categorization into different situations and learns what behavior to use in each situation, establishes a convention with itself in the form of a mapping between contexts and behaviors. Given a mapping from possible world states to internal states, there is a corresponding mapping from internal states to partial policies that in combination leads to the optimal complete policy.

The issue of arising at a convention becomes more pronounced in problems such as that used here, where multiple agents have the ability to exchange signals. This variant of establishing a mapping between internal states and signals to produce can be seen as a rudimentary form of communication development; while any consistent signaling convention is sufficient to allow agents to produce optimal behavior, and the specific signaling convention that will be used is therefore arbitrary, arriving at such a convention is a difficult coordination problem.

6. Conclusion

We investigated an approach to partially observable reinforcement learning problems where the representation of the policy is separated into a context component and an action component.

Useful bias was extracted from a simple version of the problem by retaining the action component. When put to use on more difficult problems, this method solved problems which an unbiased approach was unable to address (within the given amount of computation). In future work, we hope to explore other mechanisms for extracting bias from learning and utilizing it to aid subsequent learning. We believe this may in principle make it possible to address problems that can not practically be addressed by conventional learning methods.

Acknowledgement

EdJ gratefully acknowledges an NWO Talent-fellowship.

References

[1] Leemon C. Baird. *Reinforcement Learning Through Gradient Descent.* PhD thesis, CMU, Pittsburgh, PA, 1999.

[2] Eric Baum. *Neural Networks and Machine Learning*, chapter Manifesto for an Evolutionary Economics of Intelligence, pages 285–344. Springer-Verlag, 1998.

[3] Eric Baum. Toward a model of intelligence as an economy of agents. *Machine Learning*, 35:155–185, 1999.

[4] Eric B. Baum. Evolution of cooperative problem solving in an artificial economy. *Neural Computation*, 12:2743–2775, 2000.

[5] Jonathan Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12:149–198, 2000.

[6] Edwin D. De Jong and Jordan B. Pollack. Utilizing bias to evolve recurrent neural networks. In *Proceedings of the International Joint Conference on Neural Networks*, volume 4, pages 2667–2672, 2001.

[7] G. Z. Grudic and L. H. Ungar. Localizing policy gradient estimates to action transitions. In *Proceedings of the Seventeenth International Conf. on Machine Learning.* Morgan Kaufmann, 2000.

[8] G. Z. Grudic and L. H. Ungar. Localizing search in reinforcement learning. In *Proceedings of the Seventeenth National Conf. on Artificial Intelligence*, 2000.

[9] Dean F. Hougen, Maria Gini, and James Slagle. An integrated connectionist approach to reinforcement learning for robotic control. In *Proceedings of the Seventeenth International Conf. on Machine Learning.* Morgan Kaufmann, 2000.

[10] Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of AI Research*, 4:237–277, 1996.

[11] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie P. Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the Fifteenth Conf. on Uncertainty in Artificial Intelligence*, pages 427–436. Morgan Kaufmann, 1999.

[12] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Department of Computer Science, Rutgers University, New Brunswick, New Jersey, May 1980.

[13] Leonid Peshkin. *Reinforcement Learning by Policy Search.* PhD thesis, Brown University, Providence, RI, 2001.

[14] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie P. Kaelbling. Learning to cooperate via policy search. In *Proceedings of the Sixteenth Conf. on Uncertainty in Artificial Intelligence*, pages 307–314, San Francisco, CA, 2000. Morgan Kaufmann.

[15] Christian R. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives.* PhD thesis, MIT, Cambridge, MA, 2001.

[16] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, Cambridge, MA, 1998.

Selecting Relevant Features for Splice Site Prediction by Estimation of Distribution Algorithms

Yvan Saeys[1], Sven Degroeve[1], Dirk Aeyels[2], Yves Van de Peer[1] and Pierre Rouzé[1,3]

[1] Department of Plant Systems Biology, Ghent University, Flanders Interuniversity Institute of Biotechnology (VIB), K.L. Ledeganckstraat 35, Ghent, 9000, Belgium
[2] SYSTeMS Research Group, Ghent University, Technologiepark - Zwijnaarde 9, Zwijnaarde, 9052, Belgium
[3] Laboratoire associé de l'INRA (France)

1 Introduction

For many biological processes, it is still not clear which elements contribute to the observed behaviour. One example is the occurence of splice sites in gene sequences, an important characteristic for gene finding in genome sequencing projects, as well as for better understanding the molecular mechanism of gene expression. The DNA sequence of most genes are coding for messenger RNA (mRNA) themselves encoding proteins. While in lower organisms (prokaryotes) the mRNA is a mere copy of a fragment of the DNA, in higher organisms (eukaryotes) the DNA contains non-coding segments in genes (introns) which should be precisely spliced out to produce the mRNA. The splice sites we refer to here are the border sides of such introns. The splice site on the left (upstream part) of the intron is called the donor site, the other site is termed the acceptor site. Due to the completion of the sequencing of the genome of *Arabidopsis thaliana*, a model system for plants, much data became available, allowing the use of supervised learning methods to automate the process of splice site prediction. As it is not clear which features are relevant for an accurate splice site prediction these learning methods are usually provided with many features, assuming that this will increase the probability of including relevant information. Since not all features are important, and some of the features might be correlated, there is a need to search for an optimal subset of features that maximizes the predictive accuracy of the classification system. Traditional Feature Subset Selection (FSS) methods are sequential and are based on a greedy heuristic. Sequential Forward Elimination (SFE) starts with the empty feature set and iteratively adds features, while Sequential Backward Elimination (SBE) starts with the full feature set and iteratively discards features. More advanced methods use heurstics to search the space of feature subsets, like e.g. genetic algorithms. Recently, Estimation of Distribution Algorithms (EDAs; Mühlenbein and Paaβ, 1996) emerged as a more general framework of genetic algorithms. Instead of using the traditional crossover and mutation operators to create the new population, a more statistical approach is used to estimate the distribution of the parameters from a selected group of individuals. Creation of the new population is then performed by sampling individuals from the estimated distribution. EDAs have proven to outperform the standard genetic algorithms in many problems where multiple dependencies among parameters exist, and they usually need fewer fitness evaluations to obtain good solutions. We combined the use of Estimation of Distribution Algorithms with a wrapper approach for feature subset selection. The results of our experiments show that it clearly outperforms the traditional sequential methods for FSS.

2 Methods

2.1 Splice site data sets

The *Arabidopsis thaliana* data set was generated by aligning mRNAs obtained from the public EMBL database with the BAC-sequences that were used for the *Arabidopsis* chromosome assembly. Redundant genes were excluded resulting in a data set containing 1495 genes. From each gene only these introns confirming the GT-AG consensus were used to construct the set of positive instances. All GT dinucleotides at the start of these introns are positive donor instances and all AG dinucleotides at the end of these introns are positive acceptor instances. The negative donor instances are defined as, for all genes, all GT dinucleotides that are located between 100 nucleotide positions upstream of the first donor and 100 nucleotide positions downstream of the last acceptor in that gene and that are not donor sites. The negative acceptor instances are defined as all AG dinucleotides within the same range and that are not acceptor sites. More information on the procedure for creating these datasets can be found in (Degroeve et al., 2002).

Splice site prediction can be divided into two subtasks : prediction of donor sites and prediction of acceptor sites. Each of these subtasks can be formally stated as a two-class classification task : {donor site, non-donor site} and {acceptor site, non-acceptor site}. The features describing the positive and negative instances can be divided into two subsets : position-dependent and position-independent features. In our experiments we used a fixed window of p nucleotide positions to the left (upstream the splice site) and q positions to the right (downstream the splice site) where $p = q = 50$. From this local context, 100 position-dependent and 128 position-independent features were extracted. The position-dependent features were then converted into binary format using sparse vector encoding. This results in 400 position-dependent and 128 position-independent features. A training data set with balanced class distribution was compiled by random selection of 1000 positive instances and 1000 negative instances (GT_{2000} and AG_{2000}). For the test data set we extracted all candidate splice sites within the interval as defined above from 50 genes. This results in a test data set GT_{50} with 281 positive and 7505 negative instances and a test data set AG_{50} with 281 positive and 7643 negative instances.

2.2 Estimation of Distribution Algorithms

During the last years, Estimation of Distribution Algorithms (EDA's) emerged as a more general framework for genetic algorithms. The main critics for standard genetic algorithms include the large number of parameters that have to be tuned, the difficult prediction of the movements of the populations in the search space and the fact that there is no mechanism for capturing the relations among the variables of the problem. EDA's try to overcome these difficulties by providing a more statistical analysis of the selected individuals, thereby explicitly modelling the relationships among the variables.

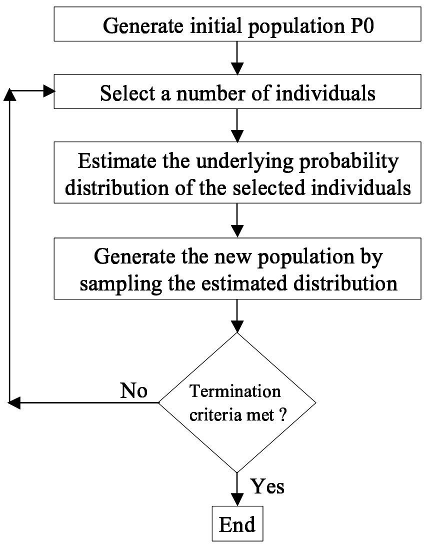

Generate initial population P0
Select a number of individuals
Estimate the underlying probability distribution of the selected individuals
Generate the new population by sampling the estimated distribution
No
Termination criteria met ?
Yes
End

Figure 1: Schematic overview of the EDA algorithm.

Figure 1 illustrates the main scheme of the EDA approach. The actual estimation of the underlying probability distribution of the selected individuals represents the core of the new EDA paradigm, and can be considered an optimization problem on its own. Depending on the domain (discrete or continuous), different estimation algorithms with varying complexity (modelling univariate, bivariate or multivariate dependencies) were designed. For an overview see Larrañaga and Lozano (2001). In the most complex case of multivariate dependencies, Bayesian Networks are frequently used. A greedy search algorithm is then used to find a suitable (and often constrained) network that is likely to generate the selected individuals.

2.3 Classification models

As described above, our data sets contain positive and negative instances that are described by q nucleotide positions downstream and p nucleotide positions upstream the consensus. Formally, a data set T contains l instances $\mathbf{x_i}$ ($i = 1, \ldots, l$) with each $\mathbf{x_i}$ labelled as y^+ or y^- (known as *classes*), indicating a positive or negative instance, respectively. Each index x_{ij} ($j = 1, \ldots, n$) in vector $\mathbf{x_i}$ is a feature F_j.

Two methods for discriminating between positive and negative instances are described below. They are supervised classification methods that induce a decision function from the instances in T which can then be used to classify a new instance $\mathbf{z}$ not seen in T.

2.3.1 Support Vector Machines

The Support Vector Machine (SVM) (Boser et al., 1992; Vapnik, 1995) is a data-driven method for solving two-class classification tasks. The Linear SVM (LSVM) separates the two classes in T with a hyperplane in the feature space such that:

(a) the "largest" possible fraction of instances of the same class is on the same side of the hyperplane, and

(b) the distance of either class from the hyperplane is maximal.

The prediction of a LSVM for an unseen instance $\mathbf{z}$ is 1 (classified as a positive instance) or -1 (classified as a negative instance), given by the decision function

$$pred(\mathbf{z}) = \mathrm{sgn}(\mathbf{w} * \mathbf{z} + b). \qquad (1)$$

The hyperplane is computed by maximizing a vector of Lagrange multipliers α in

$$W(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i\, \alpha_j\, y_i\, y_j\, K(\mathbf{x_i}, \mathbf{x_j}),$$

$$\text{constrained to: } 0 \leq \alpha_i \leq C \text{ and } \sum_{i=1}^{l} \alpha_i\, y_i = 0, \quad (2)$$

where C is a parameter set by the user to regulate the effect of outliers and noise, i.e. it defines the meaning of the word "largest" in (a).
Function K is a kernel function and maps the features in T, called the input space, into a feature space defined by K in which then a linear class separation is performed. For the LSVM this mapping is a linear mapping:

$$K(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} * \mathbf{x_j}. \qquad (3)$$

The non-linear mappings used in this paper is the Polynomial-SVM (PSVM):

$$K(\mathbf{x_i}, \mathbf{x_j}) = (\mathbf{x_i} * \mathbf{x_j} + 1)^d, \qquad (4)$$

After calculating the α_i's in (2), the decision function (1) becomes:

$$pred(\mathbf{z}) = \mathrm{sgn}(\sum_{i=1}^{l} \alpha_i\, y_i\, K(\mathbf{x_i}, \mathbf{z}) + b). \qquad (5)$$

For the LSVM this function reduces to (1) with

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i\, \mathbf{x_i}\, y_i\,. \qquad (6)$$

In (5) each α_i is associated with $\mathbf{x_i}$. After optimizing (2) many α_i's will become zero and the corresponding $\mathbf{x_i}$ will not be used in the decision function (5). All $\mathbf{x_i}$ for which the α_i is not zero are called the support vectors. Typically the size of the set of support vectors is much smaller than l.
The run-time complexity for training a Support Vector Machine is low order polynomial, usually approximately quadratic in the number of training samples (Hush and Scovel , 2000; Joachims, 1998).

2.3.2 Naive Bayes Classifier

The NBC (Duda and Hart, 1973) follows the Bayes optimal decision rule, that tells us to assign a class y^c (c in {+,-}) to an unseen instance $\mathbf{z}$ with features $(F_1^z, F_2^z, \ldots, F_n^z)$ that maximizes $P(y^c|F_1^z, \ldots, F_n^z)$, or the probability of the class y^c given the features $(F_1^z, F_2^z, \ldots, F_n^z)$. By using Bayes' rule we can write $pred(\mathbf{z}) = y^c$ as:

$$y^c = \mathrm{argmax}_c \frac{P(F_1^z, \ldots, F_n^z|y^c) \times P(y^c)}{P(F_1^z, \ldots, F_n^z)} \qquad (7)$$

The naive Bayes method then simplifies the problem of estimating $P(F_1^z...F_n^z|y^c)$ by making the arguable naive independence assumption that the probability of the features given the class is the product of the probabilities of the individual features given the class:

$$P(F_1^z, \ldots, F_n^z|y^c) = \prod_{1 \leq j \leq n} P(F_j^z|y^c). \qquad (8)$$

The time complexity of the naive Bayes method is essentially linear in the number of training samples (McCallum and Nigam, 1998).

2.4 Feature subset selection methods

For selecting an optimal subset of features from $\{F_1, F_2, \ldots, F_n\}$, given the instances in T, one needs to define what is meant by "optimal subset of feature" (referred to as the *selection criterion*), and define a *search algorithm* to search for this optimal subset of features in the space of feature subset candidates. A review of different search algorithms can be found in (Kohavi and John, 1997; Boz, 2002) and techniques to combine feature subset selection and naive Bayes have been discussed in the literature (Hall, 1999; Langley and Sage, 1994)
As the number of feature subsets increases exponentially with increasing n (number of features) and n is relatively large, two techniques are justified : greedy search and heuristics. In this paper we will compare three methods : greedy search combined with SVM, greedy search combined with NBC, and heuristic search by using the EDA-approach combined with the NBC.

2.4.1 Greedy search

Both the SVM and NBC are known to perform well in high-dimensional input spaces because they implicitly avoid overfitting. This allows us to start the search algorithm with the full feature set. This set is known to be a good point to start our search in the space of feature subset candidates. The candidate space is explored with just one operator which eliminates a feature from the current subset. This bottom-up search procedure is called a sequential backward elimination (SBE) procedure and is greedy enough for our data sets. A simple criterion consists in selecting that feature that decreases the predictive performance of the model the least. We tested the SBE procedure both on the SVM and the NBC. For the NBC this can be done as follows. At iteration l the feature set consists of n_l features and n_l models can be trained, leaving out each feature once in each model. At iteration $l + 1$ the feature set is then chosen belonging to the model with the best predictive performance. For SVM a slightly varying procedure is used : at iteration i for each feature F still in the feature subset we evaluate the generalization performance of the SVM model when setting F to its mean value in T (training set). This means that when considering a feature F for elimination, no model is retrained, but the alphas (2) of the previous model are reused, while setting x_{ij} to its mean value for all training instances. More details can be found in (Degroeve et al., 2002; Guyon et al., 2000).

2.4.2 Heuristic search

We combined the use of Estimation of Distribution Algorithms with a wrapper approach (Kohavi and John, 1997) for feature subset selection. The individuals in the population are represented as binary feature vectors, a 0 indicating an irrelevant feature, a 1 indicating a relevant feature. The goal of the EDA is then to look for the best subset with respect to some optimization criterion. In our case, the optimization criterion is a combination of the accuracy of the classification system coupled to the EDA and the number of features used to reach this accuracy. If two feature subsets achieve the same accuracy with respect to the same classification system, then the subset with the least number of features will result in a better fitness.
As the number of features in our real-world problem is quite large, we need to use quite large populations to allow a good estimation. Furthermore, a considerable amount of time is spent in analysing the fitness of each individual. For each individual a new model has to be trained, and this model has to be evaluated on a test set. Therefore, we need a fast classification algorithm and a fast estimation algorithm. In our experiments we used the NBC as the classification system, and the Univariate Marginal Distribution Algorithm (UMDA; Mühlenbein, 1998) as the estimation algorithm. Just like NBC, UMDA simplifies the estimation by assuming all features are independent : $p_l(x) = \prod_{i=1}^{n} p_l(x_i)$. Although using the naive assumption that parameters are independent both NBC and UMDA have shown to perform well in several fields such as text and image classification. As an adaptation to the standard UMDA we slightly modified the algorithm by replacing zero/one probabilities by very small/large probabilities.

3 Results

3.1 Ignoring feature dependencies

To investigate the effect of ignoring feature dependencies, we first compare the simple EDA-UMDA approach to a standard genetic algorithm (GA) with two-point crossover. Both

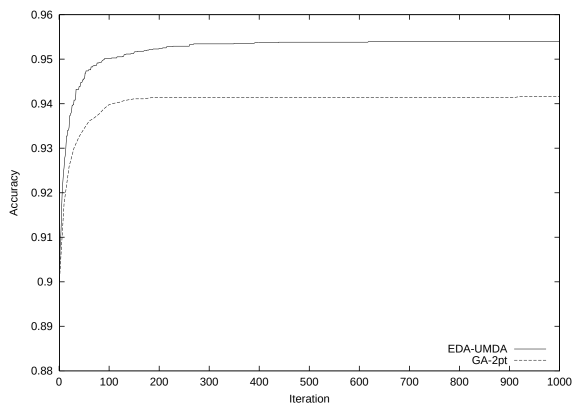


Figure 2: EDA versus GA fitness comparison.

EDA and GA used a NBC as a classification system. In both algorithms the fitness is defined as the accuracy obtained by the NBC on the test set for the selected subset of features. Both algorithms used a population of 500 individuals and an elitist approach where in each iteration the 50 best individuals survive. In both cases truncation selection was used, for the GA the crossover and mutation rate were set to 0.8 and 0.01. Figure 2 shows the comparison of both algorithms for a feature set of 528 features. Similar results were obtained on the feature set of 400 features. Clearly the EDA converges faster than the traditional GA (the EDA needs less iterations to achieve the same fitness as the GA) and also converges to better solutions than the GA. This suggests that ignoring feature dependencies can lead to good solutions. A similar conclusion was stated in (Cantú-Paz, 2002), where the author concluded that the complicated dependency learning EDA's are not needed, and the simple compact GA (Harik et al., 1998) will suffice. It has to be pointed out that the EDA-UMDA approach is very similar to the compact GA, or to a GA with uniform crossover.

3.2 Feature subset selection

To compare greedy and heuristic feature selection for splice site prediction we evaluated both techniques for the Naive Bayes Classifier. The greedy algorithm starts with the full feature set and iteratively removes features, the heuristic algorithm finds an optimal subset of features with regard to the classification accuracy of the NBC. Afterwards the greedy algorithm is applied to the solution found by the EDA, iteratively discarding features. Furthermore we evaluated the results for a Support Vector Machine with a second degree polynomial kernel (PSVM) also using the greedy approach. As a selection criterion, different measures can be used : accuracy, correlation coefficient, harmonic mean of sensitivity and specificity, or a combination of measures such as a weighted sum of accuracy and the number of eliminated features. The figures show the results when such a combination of the accuracy and the number of feature is used as selection criterion, but similar results are obtained with the other possible criteria.

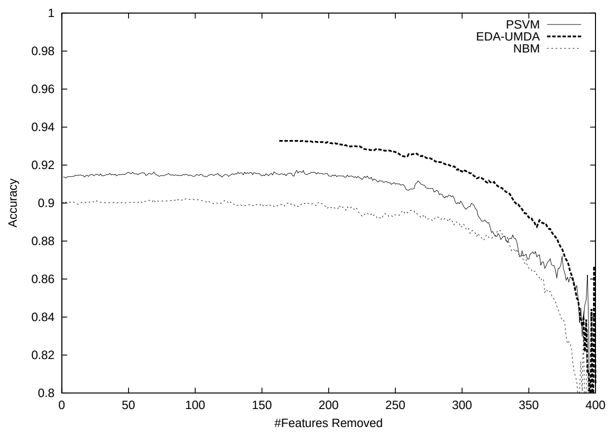


Figure 3: FSS on a set of position-dependent features.

Figure 3 and 4 show the accuracy of the three FSS methods, once for a feature set containing only position dependent features (400 features, figure 3) and once for a feature set containing position-dependent and position-independent features (528 features, figure 4). In the case of position-dependent features the SVM-model clearly performs better than the NBC. However, when applying the EDA heuristic method to the NBC, it clearly outperforms the SVM with a greedy approach. In the case of a mixture between position-dependent and position-independent features, the SVM-model starts with higher accuracy, but the NBM ap-

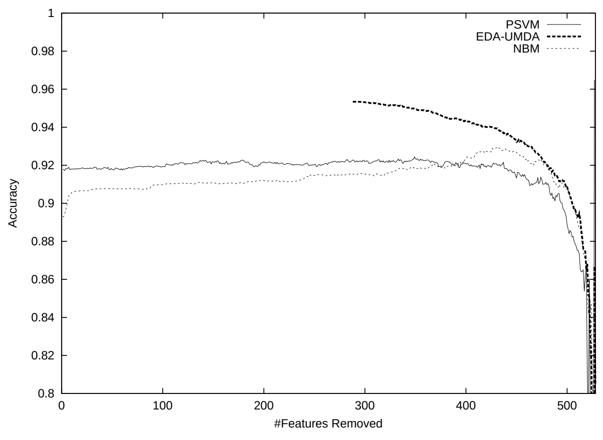


Figure 4: FSS on a set of position-dependent and position-independent features.

proach achieves higher accuracies with less features. Also here the EDA heuristic method achieves a better accuracy than both SVM and NBC with a greedy approach.

3.3 Biological relevance

The use of more position-independant features strongly influences the number of features needed to obtain good classification results. This can be demonstrated by comparing a set containing only position-dependent features with a set consisting of both position-dependent and position-independent features. For both feature sets we performed feature selection with EDA-UMDA, combined with the SBE greedy search algorithm. We then selected for each set the features that were sufficient and necessary to obtain a sensitivity of 0.93 and a specificity of 0.25. In the case of a set with only position-dependent features, 47 features were needed to achieve these values for accuracy and specificity. In the case of an extended set including also all triplets, only 25 features were needed to achieve these values. Clearly these features thus capture significant position-independent characteristics allowing a better discrimination of splice sites.

4 Related work

Genetic algorithms have been frequently used for feature subset selection in small scale (less than 100 features) domains (Kudo and Sklansky , 2000; Siedelecky and Sklansky, 1988; Vafaie and De Jong, 1993). The use of EDA's for feature subset selection was pioneered by (Inza et al., 1999) and the use of EDA's for FSS in large scale domains was reported to yield good results (Larrañaga and Lozano, 2001). Cantú-Paz (2002) compared several EDA's with the simple GA for small scale domains (at most 35 features) using a Naive Bayes classifier, and concluded that the complicated dependency learning EDA's are not significantly better than the simple compact GA.
Recently the technique of feature distributional clustering was combined with Support Vector Machines for text categorization (Bekkerman et al., 2001). This method performs feature selection by distributional clustering of words via the information bottleneck method (Tishby et al. , 1999) and can be considered a sophisticated filter method.

5 Conclusions and future work

Feature subset selection by estimation of distribution algorithms is able to select highly relevant features for splice site prediction. Future research will focus on including more position-independent information, possibly also structural information to achieve better results. More advanced feature selection frameworks such as a combined filter/wrapper model will then be needed, as the feature sets will get very large. Other interesting future directions are the combination of EDA with more advanced classification systems, and the development of faster estimation algorithms for multiple dependencies.

References

Bekkerman, R., El-Yaniv, R., Tishby, N. and Winter, Y. 2001. *On Feature Distributional Clustering for Text Categorizatio.* In *Proceedings of SIGIR-01, 24th ACM International Conference on Research and Development in Information Retrieval, pp. 146-153.*

Boser,B., Guyon,I. and Vapnik,V.N. 1992. *A training algorithm for optimal margin classifiers.* In *Proc. COLT (Haussler,D. ,ed.), ACN Press, 144-152.*

Boz, O. 2002. *Feature subset selection by using sorted feature relevance.* In *Proc. Int. Conf. on Machine Learning and Applications (ICMLA 2002).*

Cantú-Paz, E. 2002. *Feature subset selection by estimation of distribution algorithms.* In *W. B.*

Langdon, E. Cantu-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, N. Jonoska (Eds.), GECCO-2002: Proceedings of the Genetic and Evolutionary Computation Conference. (pp. 754). San Francisco, CA: Morgan Kaufmann.

Degroeve,S., De Baets,B., Van de Peer,Y. and Rouzé,P. 2002. *Feature Subset Selection for Splice Site Prediction.* In *Bioinformatics, 18-2:75-83.*

Duda,R.O. and Hart,P.E. 1973. *Pattern classification and scene analysis. New York, NY, Wiley.*

Guyon, I., Weston, J., Barnhill, S. and Vapnik, V. 2000. *Gene selection for cancer classification using support vector machines.* In *Machine Learning, 46:389-422.*

Hall, M.A. 1999. *Correlation based feature selection for machine learning. Doctoral dissertation, Department of Computer Science, The University of Waikato, Hamilton, New Zealand.*

Harik,G.R., Lobo, G.G. and Goldberg, D.E. 1998. *The compact genetic algorithm.* In *Proceedings of the International Conference on Evolutionary Computation 1998 (ECEC '98), pp. 523-528. Piscataway, NJ: IEEE Service Center.*

Hush, D. and Scovel, C. 2000. *Polynomial-time decomposition for support vector machines. Technical report, Los Alamos National Laboratory, Los Alamos, NM 87545.*

Kudo, M. and Sklansky, J. 2000. *Comparison of algorithms that select features for pattern classifiers.* In *Pattern Recogn. 33:25-41.*

Inza, I., Larrañaga, P., Etxebarria, R. and Sierra, B. 1999. *Feature subset selection by Bayesian networks based on optimization.* In *Artifical Intelligence, 27(2):143-164.*

Joachims, T. 1998. *Making large-scale support vector machine learning practical.* In *B. Scholkopf, C. Burges, A. Smola. Advances in Kernel Methods: Support Vector Machines, MIT Press, Cambridge, MA, December 1998.*

Kohavi,R. and John,G. 1997. *Wrappers for feature subset selection.* In *Artificial Intelligence Journal, 97:273-324.*

Langley, P. and Sage, S. 1994. *Induction of selective Bayesian classifiers..* In *Proceedings of the Tenth Conference on Uncertainty in Artifical Intelligence, pp. 399-406. Morgan Kaufmann, Seattle, WA.*

McCallum, A. and Nigam, K. 1998. *A comparison of event models for naive Bayes text classification..* In *AAAI/ICML-98 Workshop on Learning for Text Categorization, pp. 41-48. AAAI Press, 1998..*

Mühlenbein,H. and Paaβ G. 1996. *From recombination of genes to the estimation of distributions. Binary parameters..* In *Lecture Notes in Computer Science 1411 : Parallel Problem Solving from Nature, PPSN IV, (pp. 178-187).*

Mühlenbein,H. 1998. *The equation for response to selection and its use for prediction.* In *Evolutionary Computation 5, pp. 303-346.*

Larrañaga,P. and Lozano,J.A. 2001. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers.*

Siedelecky, W. and Sklansky, J. 1988. *On automatic feature selection. Int. J. Pattern Recogn. 2: 197-220.*

Tishby, N., Pereira, F.C. and Bialek, W. 1999. *The information bottleneck method.* In *Proc. of the 37-th Annual Allerton Conference on Communication, Control and Computing, pp. 368-377.*

Vapnik,V.N. 1995. *The nature of statistical learning theory. Springer-Verlag.*

Vafaie, H. and De Jong, K. 1993. *Robust feature selection algorithms. Proceedings Fifht International Conference on Tools with Artificial Intelligence, pp. 356-363.*

Identifying Mislabeled Training Examples in ILP Classification Problems

Sofie Verbaeten
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
Sofie.Verbaeten@cs.kuleuven.ac.be

Abstract

We consider the problem of noisy training examples, more precisely mislabeled training examples, in the context of ILP classification problems. We address this problem by preprocessing the training set, i.e. by identifying and removing outliers from the training set. We study a number of filtering techniques, some of which were proposed in the literature for attribute-value problems. We evaluate these techniques on a Bongard data set, which we artificially corrupt with different levels of classification noise.

1 Introduction

In many applications of machine learning the data to learn from is imperfect. Different kinds of imperfect information exist, and several classifications are given in the literature. In (Lavrač and Džeroski, 1994) the following types of imperfect data for Inductive Logic Programming (ILP) tasks are discussed: 1) noise, that is, random errors in training examples and background knowledge, 2) too sparse training examples, from which it is difficult to reliably detect correlations, 3) inappropriate or insufficient background knowledge, and 4) missing argument values in the training examples. In this paper, we consider the problem of noise or random errors in training examples. We address this problem in the context of ILP classification problems.

One of the problems created by learning from noisy data is overfitting, that is, the induction of an overly specific hypothesis which fits the (noisy) training data well but performs poor on the entire distribution of examples. Many techniques exist which allow ILP algorithms to handle noisy data and to prevent overfitting. Classical noise-handling mechanisms are based on appropriate search heuristics and stopping criteria used in the hypothesis construction, or on some form of post-processing of hypotheses. These techniques modify the learning algorithm itself to make it more noise-tolerant. Another approach is to pre-process the input data before learning. This approach consists of filtering the training examples (hopefully removing the noisy examples), and applying a learning algorithm on the reduced training set. As pointed out in (Gamberger et al., 2000), this separation of noise detection and hypothesis formation has the advantage that noisy examples do not influence the hypothesis construction. We will follow this approach here.

In the context of attribute-value learning there are a number of proposals for identifying and removing noisy examples from the training data. Our filtering techniques for ILP are based on some of these proposals, more precisely on the ideas presented in (Brodley and Friedl, 1999) and (John, 1995).

Many of the methods for filtering training data are motivated by the technique of removing outliers in regression analysis (Weisberg, 1980). An outlier is a case that does not follow the same model as the rest of the data and appears as though it comes from a different probability distribution. As such, an outlier does not only include erroneous data but also surprising correct data. One of the difficulties in removing noisy data is that also correct exceptions might be removed. One work addressing this problem is (Srinivasan et al., 1992). We will not cover this topic here.

In classification problems noise in the training examples can be caused by erroneous argument values and/or erroneous classifications. Quinlan (Quinlan, 1986) demonstrated that, for high levels of noise, removing noise from attribute information decreases the predictive accuracy of the resulting classifier if the same attribute noise is present when the classifier is subsequently used. This is not the case for noise in the classification of examples. In this paper, we will consider misclassified examples in the training data.

Summarising, we consider the problem of noisy training examples, more precisely erroneous classifications, in the context of ILP classification problems. We address this problem by pre-processing the training set, more precisely by identifying and removing outliers from the training set. We study a number of filtering techniques, some of which were proposed in the literature for attribute-value problems. We evaluate these techniques on a Bongard data set, which we artificially corrupt with different levels of classification noise.

In the next section, we briefly recall the first order decision tree induction system Tilde. Our filters are based on Tilde, and Tilde is also used to evaluate the filtering techniques. In section 3, we introduce the different filtering techniques. These techniques are evaluated in section 4 on a Bongard data set. We conclude and discuss topics of future research in section 5.

2 Tilde

Tilde (Top-down Induction of Logical Decision Trees) (Blockeel and De Raedt, 1998) is an ILP extension of the C4.5 decision tree algorithm (Quinlan, 1993). Instead of using attribute-value tests in the nodes of the tree, logical queries are used. As in C4.5, Tilde builds the decision tree in a top-down way, starting with the empty tree and all training examples. In each node, Tilde generates all possible tests and computes an heuristic value, in our experiments information gain ratio, for each of these tests. The test which scores best is placed in the node. The examples in the current node are then sorted down the tree: the examples passing the test are propagated to the left, the other examples to the right. The procedure is repeated for the left and right subtree. A node is turned into a leaf when the examples it covers are of a single class. After a tree is constructed, a post-pruning algorithm is used. The post-pruning algorithm that we will use here is the C4.5 post-pruning method which is based on an estimate of the error on unseen cases.

Note that the logical queries in the nodes of a first order decision tree may contain logical variables. These variables may be shared among different nodes under the restriction that a variable that is introduced in a node (that is, it does not occur in higher nodes) must not occur in the right branch (the "no"-branch) of that node. The reason for this restriction follows from the semantics of a first order decision tree. A variable that is introduced in a node is existentially quantified within the conjunction of that node. When this conjunction fails (and we thus enter the right subtree) no further reference to that variable is needed.

3 Filtering Algorithms

In this section we introduce the different filtering techniques: Robust Tilde, Voting filters and Iterated Voting filters.

3.1 Robust Tilde

The first filtering technique is based on the proposal of John (John, 1995). There, a robust version of the decision tree system C4.5 is presented, but in fact it can be applied to any decision tree learner. Here we apply it to Tilde.

Robust decision trees take the idea of pruning one step further: training examples which are locally uninformative and harmful, that is, examples which are misclassified by the pruned tree, are also globally uninformative. Therefore, after pruning a decision tree, the training examples which are misclassified should be removed from the training set and the tree needs to be rebuilt using this reduced set. This process is repeated until no more training examples are removed. The robust decision tree algorithm can be described as follows:

1. learn a decision tree from the training set, and prune this tree,
2. for each example in the training set, if the pruned tree misclassifies the example, then remove the example from the training set,
3. if no examples are removed from the training set in the previous step, then stop, else go to step 1.

Note that in the case of robust decision trees, the final learning algorithm is the same as the learning algorithm used during filtering. We apply this technique to Tilde and call the resulting system Robust Tilde (abbreviated as "R" in the next section).

3.2 Voting Filters

In (Brodley and Friedl, 1999) a general method for filtering training sets with classification noise is proposed. The idea is to use a set of learning algorithms to create classifiers that act as a filter for the training data. More precisely, the general method is as follows:

1. m learning algorithms (called filter algorithms) are chosen,
2. the training data is divided in n non-overlapping parts,
3. for each of the n parts, the m algorithms are learned on the other $n-1$ parts,
4. the m resulting classifiers are used to label each instance in the excluded part (in this way each example gets m labels),
5. the filter compares the original class of each example with the m labels it got, and decides whether or not to remove the example,
6. the filtered set of training examples is given as input to the final learning algorithm, the resulting classifier is the end product.

A variation of instances of this general scheme exists depending on the choice (and the number m) of filter algorithms, the value of n and the decision procedure used in step 5. In (Brodley and Friedl, 1999), different instances of this scheme are studied and empirically evaluated: the Single Algorithm filter ($m = 1$), and the Consensus and Majority Vote filter ($m > 1$). Step 5 in the Single Algorithm is simple: if the class of a training example differs from the (one) label it got, the example is removed. In the Consensus filter, a training example is removed only if all the labels it got (m) differ from its class. And in the Majority Vote filter, a training example is removed if the majority of the labels it got ($m/2$ – or $(m+1)/2$ in case m is odd – or more) differ from its class.

We next discuss in more detail which Voting filters we evaluate here.

3.2.1 Single Algorithm Filter

We consider a Single Algorithm filter with Tilde as filter algorithm. We use three different values for n, namely $2, 5$ and 10. In the next section, our Single Algorithm filter is abbreviated as "S(2)" (for $n = 2$), "S(5)" (for $n = 5$), and "S(10)" (for $n = 10$).

Note that the Single Algorithm filter is related to the Robust decision tree method. The difference between the two approaches is that the Single Algorithm filter uses a cross-validation over the training data with one iteration whereas the Robust decision tree method deals with all the training examples and performs multiple iterations.

3.2.2 Consensus Filter and Majority Vote Filter

Our Consensus and Majority Vote filters differ from the ones described in (Brodley and Friedl, 1999) in the sense that we do not use m different filter algorithms, we only use Tilde. More precisely, we divide the training set in n parts, and for each of the combinations of $n-1$ parts, Tilde learns on this combination. The resulting classifiers will be used to give labels to the examples in their own training sets.[1] In this way, each training example gets $n-1$ labels (since it belongs $n-1$ times to a training set). Note that the resulting filter method might be more conservative[2], since votes are given to the examples in the training set. We consider such a Consensus and Majority Vote filter for $n = 6$ (each example gets 5 labels) and $n = 10$ (each example gets 9 labels). In the next section, our Consensus filter is abbreviated as "C(6)" (for $n = 6$), and "C(10)" (for $n = 10$), and our Majority Vote filter as "M(6)" (for $n = 6$), and "M(10)" (for $n = 10$).

Our Voting filters are in a way dual to our Single Algorithm filter: instead of using the learned decision tree to classify examples in the subset excluded from the training set, the tree is used to classify the examples in its own training set. Our Voting algorithms have some similarities with the Robust approach. More precisely, in the Robust decision tree method, the learning algorithm learns on the training data and

[1]As described in Section 2, Tilde with pruning is used; this is of course essential for this approach.

[2]meaning that it will remove less examples

removes those training examples that are misclassified by the learned (and pruned) tree. Our Voting algorithms also learn trees which classify the examples in their own training set. The difference is that, for the Voting algorithms, these training sets consist of $n-1$ of the n parts, and hence each example gets $n-1$ votes. Also, the Voting algorithms are run only once, whereas Robust Tilde consists of a number of iterations (it keeps on running until no more training examples are removed). In the next subsection, we present a hybrid approach combining Robust Tilde and our Voting algorithms: the Voting filters will be repeated until no more examples are removed.

3.3 Iterated Voting Filters

The Iterated Voting filters combine the Robust method with the Voting filters. The method works as follows:

1. filter the training data using a Voting filter (Single Algorithm filter, Consensus filter or Majority Vote filter),
2. if no training examples are removed in the previous step, then the reduced set of training examples is given as input to the final learning algorithm, else repeat the previous step with the reduced training set.

When the basic filtering method (of step 1) is a Single Algorithm filter S(n), we call the resulting filter the Iterated Single Algorithm filter IS(n); when the basic filtering method is a Consensus filter C(n), we call the resulting filter the Iterated Consensus filter IC(n); and when the basic filtering method is a Majority Vote filter M(n), we call the resulting filter the Iterated Majority Vote filter IM(n). In the experiments, we consider the following Iterated Voting filters: IS(2), IS(5), IS(10), IC(6), IC(10), IM(6), and IM(10).

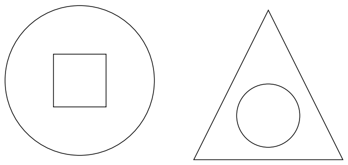
4 Empirical Evaluation

We first describe the data set that we used in the experiments. Then we explain how the experiments were carried out, and finally we discuss the results.

4.1 Data Set and Noise Introduction

We want to evaluate how well the different filtering techniques perform on data sets with different amounts of classification noise. Since real world data sets often contain already an unknown number of noisy examples, we chose to do the experiments on a Bongard problem.

Bongard data sets consist of configurations of geometrical objects, more precisely triangles, squares and/or circles. The direction in which triangles point can be up or down. An object can be inside another object. A typical Bongard example is shown below.



```
circle(o1).
square(o2).
triangle(o3).
circle(o4).
in(o2,o1).
in(o4,o3).
config(o3,up).
```

The advantage of using such a Bongard data set is that we can generate a noise-free Bongard set and introduce a controlled amount of noise artificially. The Bongard data set with which the experiments were performed contains 392 examples, 128 of them are classified as positive, the other 264 as negative.

Different levels of noise were introduced. A noise level of $x\%$ means that for a randomly chosen subset of $x\%$ of the training examples, the class-value of these examples was flipped.[3] We introduced noise levels of 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35% and 40%.[4]

4.2 Experimental Method

All the filters are based on Tilde. Tilde was also used to evaluate the performance of the different filtering techniques.

In order to obtain a more reliable estimate of the performance of the filters, all experiments were carried out in 10-fold cross-validation and the results were averaged over all 10 folds. For each of the 10 runs, we made a training set (9 parts) and a test set (remaining 1 part). The training set was corrupted by introducing classification errors using noise levels of 0%, 5%, 10%, 15%, 20%, 25%, 30%, 35% and 40%. Each of the above described filtering techniques was then run on the (noisy[5]) training set. After filtering the training set, Tilde was used to learn a

[3]Positive examples are made negative and vice versa.

[4]More precisely, taking the total number of examples into account, these (rounded) percentages were 0%, 5.10%, 9.92%, 15.02%, 20.07%, 24.94%, 30.05%, 35.09% and 39.97%.

[5]For each noise level and each of the 10 training sets,

decision tree on the reduced training set. This decision tree was validated on the (noise-free) test set. Results were obtained by taking the mean of the results of the 10 runs. For each of the 9 noise levels and each of the 10 runs, we also run Tilde directly on the (unfiltered, noisy) training set.

In the next subsections, we report results concerning filter precision, tree size and accuracy. Because of lack of space, we can not include all the results, we refer to (Verbaeten and Cardoen, 2002) instead.

4.3 Filter Precision

For each of the 9 noise levels (N), we report the following: the percentage of examples which are removed (F), the percentage of examples which are removed and are actually noisy (N & F), the percentage of noisy examples in the filtered data sets (RN), and the probability of making an error of type 1 and of type 2.

A type 1 error (E_1) is made when a correct example is filtered. The probability of making a type 1 error is estimated as $P(E_1) = \frac{\sharp(\mathcal{F}\cap\mathcal{C})}{\sharp\mathcal{C}}$, where $\mathcal{F}$ is the set of filtered examples and $\mathcal{C}$ is the set of correct (non-noisy) examples. A type 2 error (E_2) is made when a noisy example is not removed from the training set. The probability of making a type 2 error is estimated as $P(E_2) = \frac{\sharp(\overline{\mathcal{F}}\cap\mathcal{N})}{\sharp\mathcal{N}}$, where $\overline{\mathcal{F}}$ is the complement of the set $\mathcal{F}$ (i.e. the examples which are not removed) and $\mathcal{N}$ is the set of noisy examples.

We only report the results for the filters R, S(5), IS(5), M(6) and IM(6).

Robust Tilde

N	F	N & F	RN	$P(E_1)$	$P(E_2)$
0	0.03	0	0	0.0003	NA
5.10	5.58	4.90	0.21	0.007	0.039
9.92	10.29	9.81	0.13	0.005	0.011
15.02	15.99	13.95	1.29	0.024	0.072
20.07	21.68	18.40	2.14	0.041	0.083
24.94	25.28	22.79	2.87	0.033	0.086
30.05	30.27	25.74	6.13	0.065	0.143
35.09	33.82	27.35	11.67	0.100	0.221
39.97	38.32	25.71	23.08	0.210	0.357

the classification errors were introduced only once (in a random way), and the different filtering techniques were run on the same noisy training sets.

Single Algorithm Filter, $n = 5$

N	F	F & F	RN	$P(E_1)$	$P(E_2)$
0	0.17	0	0	0.002	NA
5.10	6.60	4.93	0.18	0.018	0.033
9.92	12.16	9.78	0.16	0.026	0.014
15.02	17.60	13.97	1.28	0.043	0.070
20.07	24.83	18.79	1.69	0.076	0.064
24.94	29.65	22.53	3.42	0.095	0.097
30.05	36.48	26.96	4.85	0.136	0.103
35.09	42.01	27.83	12.50	0.218	0.207
39.97	46.43	29.68	19.11	0.279	0.257

Iterated Single Algorithm Filter, $n = 5$

N	F	N & F	RN	$P(E_1)$	$P(E_2)$
0	0.37	0	0	0.004	NA
5.10	8.02	5.05	0.06	0.031	0.011
9.92	13.78	9.89	0.03	0.043	0.003
15.02	19.45	14.17	1.06	0.062	0.057
20.07	27.13	19.30	1.04	0.098	0.038
24.94	33.68	23.84	1.64	0.131	0.044
30.05	42.66	28.91	1.94	0.196	0.038
35.09	48.30	30.38	8.92	0.276	0.134
39.97	55.39	33.70	13.5	0.361	0.157

Majority Vote Filter, $n = 6$

N	F	N & F	RN	$P(E_1)$	$P(E_2)$
0	0.03	0	0	0.0003	NA
5.10	5.47	4.90	0.21	0.006	0.039
9.92	10.40	9.72	0.22	0.008	0.020
15.02	15.48	13.83	1.41	0.019	0.079
20.07	20.35	18.62	1.81	0.022	0.072
24.94	25.00	22.53	3.20	0.033	0.097
30.05	28.23	25.77	5.93	0.035	0.142
35.09	31.80	25.96	13.28	0.090	0.260
39.97	34.21	24.71	23.16	0.158	0.382

Iterated Majority Vote Filter, $n = 6$

N	F	N & F	RN	$P(E_1)$	$P(E_2)$
0	0.03	0	0	0.0003	NA
5.10	5.56	4.96	0.15	0.006	0.028
9.92	11.14	9.72	0.23	0.016	0.020
15.02	15.73	13.92	1.32	0.021	0.074
20.07	21.32	18.79	1.62	0.032	0.064
24.94	25.93	23.10	2.49	0.038	0.074
30.05	31.09	27.32	3.94	0.054	0.091
35.09	35.40	27.78	11.32	0.117	0.208
39.97	39.31	26.79	21.66	0.209	0.330

From all our experiments (see (Verbaeten and Cardoen, 2002)), we can conclude that, except for the (Iterated) Consensus filters, all filters perform well up to a noise level of 25% (or higher). That is, in the reduced training sets only a small percentage of the examples are noisy.

If we look at the different Single Algorithm filters, we see that (if we are concerned with filter precision), S(10) should be preferred over S(5), which in turn should be preferred over S(2). Indeed, our experiments show that the higher the parameter n in the Single Algorithm filter, the less aggressive the filter is (i.e. throwing out less examples) and the smaller the probabilities $P(E_1)$ and $P(E_2)$ are. We might explain this by observing that the classifiers, which act as filter, are trained on $n-1$ parts of the training set. So, the higher n, the more training data is available. However, it should be observed that this training data is noisy as well. It can be easily seen that the higher the parameter n, the more time the filtering takes. We were not concerned with efficiency in this paper.

Comparing S(10) (the "best" filter of the Single Algorithm filters) with the Robust filter, we notice that S(10) is more aggressive (removes more examples) than the Robust filter. More precisely, S(10) removes more correct examples (higher $P(E_1)$) than the Robust filter but removes also more noisy examples (smaller $P(E_2)$) than the Robust filter.

For the iterated versions of the Single Algorithm filters, we see that IS(10) is less aggressive than IS(5), which in turn is less aggressive than IS(2). Also, IS(5) is more precise than IS(2) (smaller $P(E_1)$ and $P(E_2)$), and IS(10) is more precise than IS(5) (smaller $P(E_1)$, but $P(E_2)$ is more or less the same).

From all filters presented here, IS(2) is the most aggressive one (removes the most examples).

Comparing the Single Algorithm filters S(n) with their iterated versions IS(n) ($n = 2, 5, 10$), we observe that IS(n) is more aggressive than S(n), has higher $P(E_1)$ and smaller $P(E_2)$. This can be easily explained since IS(n) is just S(n) run several times (until no examples are removed anymore).

Comparing IS(10) (and IS(5)) and the Robust filter: the Robust filter is less aggressive than IS(10) (and hence IS(5)), IS(5) and IS(10) have smaller $P(E_2)$ than the Robust filter, and the Robust filter has smaller $P(E_1)$ than IS(10) (and hence IS(5)).

The Consensus filters C(6) and C(10) are the most conservative (the less aggressive) filters. From all filters, they have the smallest $P(E_1)$ and highest $P(E_2)$ values. One can choose to use a Consensus filter if the training set is small and the cost of adding new training examples is high. The iterated versions of the Consensus filters are (as can be easily understood) less conservative (but in comparison with the other filters, still more conservative). This results in a slightly higher, but still acceptable, $P(E_1)$, but a smaller $P(E_2)$.

The Majority Vote filters M(6) and M(10) have a precision comparable with the precision of the Robust filter. The precision of the iterated versions of the Majority Vote filters are comparable with (but slightly higher $P(E_1)$ and slightly lower $P(E_2)$ than) the precision of the Majority Vote filters.

4.4 Tree Size

We tabulate the number of nodes in the trees induced from the filtered sets. We also report the number of nodes in the trees induced from the unfiltered sets (column under "T"). Note that, since we take the mean over 10 runs, the values in the tables below are not always integers. When the number of nodes in a tree induced from a filtered set is smaller than or equal to the number of nodes in the tree induced from the unfiltered data set, the number is put in bold.

N	T	R	S(5)	IS(5)	M(6)	IM(6)
0	7.2	**7.2**	**7**	**7**	**7.2**	**7.2**
5.10	8.8	**8**	**8**	**7.4**	**7.7**	**7.7**
9.92	9.2	**8.3**	**7.9**	**6.9**	**8.2**	**8**
15.02	8.3	**7.3**	**7**	**5.9**	**7.8**	**7.8**
20.07	8.5	**7.8**	**6.9**	**5.3**	8.7	**8.5**
24.94	9.3	**8.3**	**8.4**	**5.4**	**8.2**	**7.7**
30.05	12.3	**10.9**	**8.2**	**4.5**	**7.9**	**7.8**
35.09	11.6	12.2	**8.8**	**4.5**	**11.4**	**9.2**
39.97	9.4	10.5	**7.5**	**4.2**	11.7	9.5

The trees induced by Tilde from a filtered training set are almost always smaller than the trees induced by Tilde from the noisy training set (except for the (Iterated) Consensus filters where the trees are larger for noise levels from 15-20% to 40%). Using the IS(2) filter results in the smallest trees (see the following subsection).

4.4.1 Comparison with Random Data Reduction

In (Oates and Jensen, 1997) it is empirically shown that for many data sets there is a nearly linear relationship between training set size and tree size, even after accuracy has ceased to increase. This suggests that all data reduction techniques will see some decrease in tree size simply because they are reducing the size of the training set. Therefore, each data reduction method should be compared with random data reduction. Only then one can have an idea of how much of the reduction in tree size is directly attributable to how a data reduction method selects instances to remove.

In (Oates and Jensen, 1997) this analysis was done for Robust C4.5 (John, 1995) and it was shown that 41.67% of the decrease in tree size is attributable to reduction in training set size. The remainder is due to the removal of uninformative examples.

We investigated this issue for some of our filters, and report our results for the IS(2) filter[6] here. To determine how much of IS(2)'s effect on tree size for a given data set is due to reduction of training set size alone, we need to know the following (we follow the approach of (Oates and Jensen, 1997)):

- the size of the tree that Tilde builds on the entire data set (T),
- the size of the tree that Tilde builds on the filtered data set (IS(2)),
- the size of the tree that Tilde builds on the data set from which we randomly removed the same percentage of examples as IS(2) did (RDR).

The percentage of IS(2)'s effect on tree size which is due to reduction in training set size can then be computed as:

$$100 * \frac{\text{T - RDR}}{\text{T - IS(2)}}$$

This percentage is shown in the following table in the column under "Effect". In the column under "F" the percentage of training examples that were removed by the IS(2) filter is reported (note that this is also the percentage of randomly removed training examples).

[6]Recall that, using the IS(2) filter results in the smallest trees.

N	T	IS(2)	F	RDR	Effect
0	7.2	**6.3**	6.26	8.1	-100
5.10	8.8	**5.4**	12.70	9	-5.88
9.92	9.2	**5.1**	17.97	**8.5**	17.07
15.02	8.3	**4.4**	22.31	**7.9**	10.26
20.07	8.5	**3.9**	32.54	**8.1**	8.70
24.94	9.3	**4.2**	37.24	**8.5**	15.69
30.05	12.3	**3.8**	48.04	**7.4**	57.65
35.09	11.6	**2.2**	57.06	**9.7**	20.21
39.97	9.4	**2.3**	58.84	**6.6**	39.44

We see that, for all noise levels, a great percentage of the reduction in tree size is attributable to how IS(2) selects examples to remove. For the noise level of 10% (17.97% of the examples are removed) and higher, we also see a reduction in tree size using random data reduction, but this reduction is never as big as the reduction we observe when we use IS(2).

4.5 Accuracy

We tabulate the accuracies of the trees induced from the filtered sets (on the non-noisy test sets). We compare them with the accuracies (also on the non-noisy test sets) of the trees induced from the unfiltered, noisy sets (reported in the column "T"). We only report our results for the filters R, M(6), and IS(2).

In the column under "RDR" we report the accuracies of the trees induced on the training set from which we randomly removed the same percentage of training examples as the IS(2) filter did (see previous subsection).

When the accuracy obtained by using Tilde on a filtered training set is equal to or higher than the accuracy obtained by using Tilde on the unfiltered training set, it is put in bold.

N	T	R	M(6)	IS(2)	RDR
0	1	**1**	**1**	0.93	**1**
5.10	0.99	**0.99**	**0.99**	0.91	0.97
9.92	0.99	**0.99**	0.99	0.92	0.98
15.02	0.94	**0.94**	**0.95**	0.91	0.94
20.07	0.95	0.95	**0.96**	0.86	0.94
24.94	0.93	**0.94**	0.92	0.87	0.90
30.05	0.90	0.90	**0.93**	0.84	0.83
35.09	0.85	**0.86**	0.85	0.74	0.72
39.97	0.75	**0.75**	0.74	0.70	0.69

From the experiments (see (Verbaeten and Cardoen, 2002)) we see that using a (Iterated) Single Algorithm filter before learning with Tilde does not increase accuracy (except for a few cases). More interestingly in this respect is Robust Tilde and the (Iterated) Majority Vote filters, which perform best w.r.t. accuracy. We also observe that the (Iterated) Consensus and (Iterated) Majority Vote filters are especially useful for noise levels of 15% and 20%. Because Majority Vote filters perform better than Consensus filters, we might conclude that retaining noisy examples hinders performance (in terms of accuracy) more than throwing out correct examples. This trend is particularly important when one has an abundance of data. This was also observed in (Brodley and Friedl, 1999) for other (non-artificial) data sets.

Strangely enough, we observe that up to a noise level of 25%, the accuracy of Tilde using RDR is always higher than[7] the accuracy of Tilde using the IS(2) filter. This is not what we expected, and we plan to investigate this issue further.

Finally, we should note that the accuracy of Tilde on an unfiltered noisy training set is still very good: Tilde[8] is very noise-tolerant.

5 Conclusion and Future Work

We addressed the problem of training sets with mislabeled examples in ILP classification problems. We proposed a number of filtering techniques for identifying and removing noisy examples. We experimentally evaluated these techniques on a Bongard data set which we artificially corrupted with different levels of classification noise. We reported results concerning filter precision, tree size and accuracy.

There are several issues which remain to be studied. First of all, we only validated the filtering techniques on one (artificial) data set. It remains to be seen if our conclusions also hold for other (non-artificial) data sets.

A hypothesis of interest is whether a majority vote ensemble classifier can be used instead of filtering. This hypothesis was tested in (Brodley and Friedl, 1999). There, two majority vote ensemble classifiers were formed: one from the filtered and one from the unfiltered data. The resulting classifiers were then used to classify the uncorrupted test data. The experiments in (Brodley and Friedl, 1999) show that the majority vote classifier performed better than the individual classifiers, but that it cannot replace filtering when data are noisy. It is concluded that the best approach is to combine filtering and voting. We are currently testing this hypothesis in our setting.

We were not concerned with efficiency in this paper. But it can be easily understood that some of our filters take quite some time. The Robust filter and the Iterated Voting filters run Tilde several times on slightly modified (reduced) training sets. An efficient algorithm for updating first-order trees would be beneficial in this respect.

[7]but still not higher than the accuracy of Tilde trained on the unfiltered noisy data set

[8]Recall that we use Tilde with pruning.

References

H. Blockeel and L. De Raedt. 1998. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June.

C.E. Brodley and M.A. Friedl. 1999. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167.

D. Gamberger, N. Lavrač, and S. Džeroski. 2000. Noise detection and elimination in data preprocessing: experiments in medical domains. *Applied Artificial Intelligence*, 14:205–223.

G.H. John. 1995. Robust decision trees: Removing outliers from databases. In U.M. Fayyad and R. Uthurusamy, editors, *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 174–179. AAAI Press.

N. Lavrač and S. Džeroski. 1994. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

T. Oates and D. Jensen. 1997. The effects of training set size on decision tree complexity. In *Proceedings of The Fourteenth International Conference on Machine Learning*, pages 254–262. Morgan Kaufmann, Nashville, TN.

J.R. Quinlan. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

J.R. Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in machine learning. Morgan Kaufmann.

A. Srinivasan, S. Muggleton, and M. Bain. 1992. Distinguishing exceptions from noise in nonmonotonic learning. In *Proceedings of the 2nd International Workshop on Inductive Logic Programming*.

S. Verbaeten and T. Cardoen. 2002. Techniques for identifying mislabeled training examples in ilp classification problems. Technical Report CW-347, Department of Computer Science, Katholieke Universiteit Leuven.

S. Weisberg. 1980. *Applied linear regression*. John Wiley & Sons.

Locally Linear Generative Topographic Mapping

J.J. Verbeek and **N. Vlassis** and **B. Kröse**
Intelligent Autonomous Systems Group
Informatics Institute, University of Amsterdam
{*verbeek,vlassis,krose*}*@science.uva.nl*

Abstract

We propose a method for non-linear data projection that combines Generative Topographic Mapping and Coordinated PCA. We extend the Generative Topographic Mapping by using more complex nodes in the network: each node provides a linear map between the data space and the latent space. The location of a node in the data space is given by a smooth non-linear function of its location in the latent space. Our model provides a piece-wise linear mapping between data and latent space, as opposed to the point-wise coupling of the Generative Topographic Mapping. We provide experimental results comparing this model with GTM.

1 Introduction

The Generative Topographic Mapping (GTM) (Bishop et al., 1998b) is a tool for non-linear data projection that has found many applications since its introduction. The idea is to fix a set of points in a latent space (which is used for visualization), these points are mapped through a Radial Basis Function Network (Ripley, 1996) into the data-space, where they are the means of the components of a Gaussian mixture. For projection, the data posterior probabilities on the mixture components are computed and the latent-space coordinates are averaged accordingly.

In this paper we extend the GTM by using more expressive covariance structure for the Gaussians in the mixture and, more importantly, by attaching to each mixture component a linear map between data and latent space.

The model may also be regarded as a restricted form of the Coordinated Principal Component Analysis (CPCA) model (Verbeek et al., 2002) (on its turn a restricted form of the coordinated factor analyzers model (Roweis et al., 2002)). CPCA extends probabilistic Principal Component Analysis (PCA) (Tipping and Bishop, 1999; Roweis, 1997) by integrating the mixture of probabilistic PCA's into one non-linear mapping between a data and a latent space. This is done by assigning a linear map from each PCA to a single, global, latent space. The objective function of CPCA sums data log-likelihood and a measure of how consistent the predictions of the latent coordinates are. With consistent we mean that PCA's that give high likelihood to a data point in the data space should give similar predictions on the (global) latent coordinate.

Here, we restrict the locations of the mixture components in the D-dimensional data space to be given by a smooth (non-linear) function of their location in the d-dimensional latent space ($d \ll D$). The relation between the proposed model and the CPCA is analogue to the relation between the GTM and Kohonen's Self-Organizing Map (Kohonen, 2001).

CPCA can be used for data visualization, however the model only aims at data log-likelihood maximization and at uni-modal distributions on the latent coordinate given a data point (consistency). What we would also like is uni-modal distributions in the other direction: the density on the data-space given a latent coordinate. By constraining the data-space centers of the Gaussian mixture components to be given by a smooth non-linear mapping of their latent space centers we safeguard CPCA against mixture components that are close in latent space but widely separated in the data space (which can cause multi-modal distributions from latent to data space).

The rest of the paper is organized as follows: in the following section we will discuss GTM and PCA and CPCA in some more detail.

Then, in Section 3 we present the new generative model and an EM-style learning algorithm is given in Section 4. Experimental results are presented and discussed in Section 5. We end the paper with a discussion and some conclusions in Section 6.

2 Generative Topographic Mapping, PCA and Coordinated PCA

Generative Topographic Mapping The Generative Topographic Mapping (GTM) (Bishop et al., 1998b) model is a tool for data visualization and data dimensionality reduction. Typically GTM is used for high dimensional numerical data. The data density is modeled by a mixture distribution that is parameterized in a way that enables low dimensional visualization of the data.

The generative model of GTM is a mixture of k spherical Gaussians, all having the same variance σ^2 in all directions. The components of the Gaussian mixture are also called nodes, and play a similar role as the nodes in Kohonen's Self-Organizing Map (Kohonen, 2001). With each mixture component s, a fixed latent coordinate $\boldsymbol{\kappa}_s$ is associated (typically the latent space is a two dimensional Euclidean space, and the $\boldsymbol{\kappa}_s$ are chosen on a rectangular grid). In the latent space a set of m fixed (non-linear) smooth basis-functions $\phi_i, \ldots, \phi_m$ are defined, the vector $\boldsymbol{\phi}(\boldsymbol{\kappa}_s)$ denotes the response of the m basis functions on input $\boldsymbol{\kappa}_s$. The mean of mixture component s is given by $\mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_\mathbf{s})$, where $\mathbf{W}$ is a $D \times m$ matrix. Hence, due to the smoothness of the basis-functions and because the multiplication with $\mathbf{W}$ is just a linear map, components with close latent coordinates will have close means in the data space.

Suitable parameters σ and $\mathbf{W}$ of GTM are typically found by employing a simple Expectation-Maximization (EM) algorithm. Like most EM algorithms, the algorithm can get stuck at locally optimal solutions, there is no guarantee to find globally optimal parameters

In order to visualize the high dimensional data in the latent space, for each data point the posterior distribution on the mixture components is computed. The latent coordinates of the nodes are then weighted according to the posterior distribution to give the latent coordinate for the data point.

Principal Component Analysis Principal Component Analysis (PCA) (Jolliffe, 1986) is a widely known and used method for data visualization and dimensionality reduction. Assuming the data has zero mean, the data vectors $\{\mathbf{x}_n\}$ are modeled as $\mathbf{x}_n = \boldsymbol{\Lambda}\mathbf{g}_n + \epsilon_n$. Here, $\mathbf{x}_n$ is a D dimensional data vector, $\mathbf{g}_n$ the corresponding latent coordinate, $\boldsymbol{\Lambda}$ a $D \times d$ dimensional matrix and ϵ_n a residual vector. PCA uses the linear mapping $\boldsymbol{\Lambda}$ that minimizes sum of the squared norms of the residual vectors to map the data into a d dimensional representation.

The PCA linear map (matrix) $\boldsymbol{\Lambda}$ can also be found as the limiting case of $\sigma \to 0$ of fitting a Gaussian density to the data where the covariance matrix is constrained to be of the form $\sigma^2\mathbf{I} + \boldsymbol{\Lambda}\boldsymbol{\Lambda}^\top$ The latter model is known as Probabilistic PCA (Roweis, 1997).

Coordinated PCA Several Probabilistic PCA's can be combined in a mixture, by taking a weighted sum of the component densities (Tipping and Bishop, 1999). In the mixture case a set local PCA's is fitted to the data. Data can now be mapped in either one of the PCA models, however coordinates of the different PCA spaces cannot be related to each other. The coordinated PCA model (Roweis et al., 2002; Verbeek et al., 2002) resolves this problem. Each PCA space is mapped linearly into a global low dimensional space. The global low dimensional space provides a single low dimensional coordinate system for the data. The question is which linear maps between the PCA's and the global latent space we would like.

It turns out that the data log-likelihood is invariant for how the different PCA spaces are coordinated, i.e. mapped into the global space. Therefore in (Roweis et al., 2002; Verbeek et al., 2002) a penalty term is added to data log-likelihood to find a coordinated mixture of PCA's. For every data point $\mathbf{x}_n$ we can compute the posterior probability on every PCA s. Furthermore, a prediction $p(\mathbf{g}_n|s, \mathbf{x}_n)$ on the global latent coordinate can be made using either PCA. The penalty term measures for every data point the ambiguity of the prediction on its global latent coordinate as given by $p(\mathbf{g}_n|\mathbf{x}_n) = \sum_s p(s|\mathbf{x}_n)p(\mathbf{g}_n|s, \mathbf{x}_n)$. The measure of ambiguity is the smallest Kullback-Leibler divergence between $p(\mathbf{g}_n|\mathbf{x}_n)$ and any Gaussian density.

It turns out that adding this penalty term is exactly the same as using a variational EM algorithm (Neal and Hinton, 1998). Hence, it is relatively straightforward to derive EM algorithms to optimize the parameters of the model for this objective function.

In the next section we describe a generative model that combines the ideas of GTM and Coordinated PCA.

3 The Generative Model of Locally Linear GTM

The generative model is best understood by starting in the latent space. First one of the k 'units' is chosen uniformly random, let's call it s. Then, a latent coordinate is drawn from $p(\mathbf{g} \mid s)$. Finally, an observation $\mathbf{x}$ is drawn from $p(\mathbf{x} \mid \mathbf{g}, s)$. Using $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to denote a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, the distributions, collectively parameterized by $\boldsymbol{\theta} = \{\sigma, \rho, \alpha, \mathbf{W}, \boldsymbol{\Lambda}_1, \ldots, \boldsymbol{\Lambda}_k\}$, are:

$$\begin{aligned} p(s) &= \frac{1}{k} \\ p(\mathbf{g} \mid s) &= \mathcal{N}(\boldsymbol{\kappa}_s, \alpha^2 \rho \sigma^2) \\ p(\mathbf{x} \mid \mathbf{g}, s) &= \mathcal{N}(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_s) + \boldsymbol{\Lambda}_s(\mathbf{g} - \boldsymbol{\kappa}_s)/\alpha, \sigma^2 \mathbf{I}) \end{aligned}$$

The marginal distribution on $\mathbf{x}$ given s reads:

$$p(\mathbf{x} \mid s) = \mathcal{N}(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_s), \sigma^2(\mathbf{I} + \rho \boldsymbol{\Lambda}_s \boldsymbol{\Lambda}_s^\top))$$

Each mixture component $p(\mathbf{x} \mid s)$ is a Gaussian with variance σ^2 in all directions except those directions given by the columns of $\boldsymbol{\Lambda}_s$ where the variance is ρ times larger. The form of the covariance matrix is that of probabilistic PCA, however here the vectors spanning the PCA subspace (given by the columns of $\boldsymbol{\Lambda}$) all have the same norm.

The distribution on latent coordinates is a mixture of isotropic Gaussians (spherical covariance matrix) with equal mixing weights. The columns of the matrix $\boldsymbol{\Lambda}_s$ span the PCA subspace of mixture component s, and provide the mapping for component s between the latent space and the data space. The parameter α is a magnification factor between latent and data space.

The generative model is almost the same as in (Verbeek et al., 2002), the only difference is that we constrain the data-space mean of the mixture component s to $\mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_s)$ and keep the $\{\boldsymbol{\kappa}_s\}$ fixed. The mapping is a linear sum, parameterized by the $D \times m$ matrix $\mathbf{W}$, of m non-linear basis functions $\phi_1, \ldots, \phi_m$ of d inputs. The output of the function $\phi_i(\boldsymbol{\kappa})$ is the i-th element of the vector $\boldsymbol{\phi}(\boldsymbol{\kappa})$. In fact, we also add one constant basis function and d linear functions which just output one of the components of the input vector. These basis-functions can be used to capture linear trends in the data. In our model we consider the basis functions $\{\phi_i\}$ to be fixed as well as the $\{\boldsymbol{\kappa}_s\}$, all other parameters are fitted to a given data set.

To compare, the generative model of GTM, which is a special case with $\rho = 0$ of the model used here, looks like:

$$\begin{aligned} p(s) &= \frac{1}{k} \\ p(\mathbf{g}|s) &= \delta(\boldsymbol{\kappa}_s) \\ p(\mathbf{x}|s, \mathbf{g}) &= \mathcal{N}(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_s), \sigma^2 \mathbf{I}), \end{aligned}$$

where $\delta(\boldsymbol{\kappa}_s)$ denotes the distribution that puts all mass at $\boldsymbol{\kappa}_s$. This is similar to the model used in this paper, but (1) GTM uses a mixture of spherical Gaussians for the data space model and (2) GTM uses a mixture of delta peaks to model the density in the latent space.

Note that in our model, given a specific mixture component, the expected value of $\mathbf{x}$ depends linearly on $\mathbf{g}$ and vice versa, see Section 4.3. Whereas using GTM, given a mixture component s, there is no modeling of uncertainty in the latent coordinate $\mathbf{g}$ corresponding to a data point $\mathbf{x}$, it is simply assumed that $\mathbf{g} = \boldsymbol{\kappa}_s$.

4 Learning

The learning algorithm tries to achieve two goals. On the one hand we want a good fit on the observed data, i.e. high data log-likelihood. On the other hand, we want the mappings from data to latent space to give 'consistent' predictions, i.e. if a data point is well modeled by two mixture components in the data space, then these should give similar predictions on the corresponding latent coordinate of the data point. In the following subsections we first discuss the objective function, then we describe how we initialize all the model parameters, and finally we discuss the learning algorithm.

4.1 The Objective Function

As shown by Roweis et. al. (Roweis et al., 2002) the double objective can be pursued with a variational Expectation-Maximization (EM) algorithm. The free-energy interpretation of EM (Neal and Hinton, 1998) clarifies how we can use EM-like algorithms to optimize penalized log-likelihood objectives. The penalty term is the Kullback-Leibler (KL) divergence between the posterior on the hidden variables and a specific family of distributions, in our case uni-modal Gaussian distributions. The objective function sums data log-likelihood and the KL divergence which is a measure of how uni-modal the predictions on the latent coordinate are.

The objective $\mathbf{\Phi}$ can be decomposed in two ways, given in equations (1) and (2) below, which can be associated respectively with the E and M step of EM. The first is the log-likelihood minus the KL-divergence, which is convenient for the E-step. The second decomposition isolates an entropy term independent of $\boldsymbol{\theta}$, which is convenient in the M-step.

$$\begin{aligned}\mathbf{\Phi} &= -\frac{\lambda}{2} \parallel \mathbf{W} \parallel^2 + \sum_n \log p(\mathbf{x}_n;\boldsymbol{\theta}) \\ &\quad - \sum_n \mathcal{D}_{KL}(Q_n(\mathbf{g},s) \parallel p(\mathbf{g},s \mid \mathbf{x}_n;\boldsymbol{\theta})) \quad (1)\\ &= -\frac{\lambda}{2} \parallel \mathbf{W} \parallel^2 + \sum_n \mathrm{E}_{Q_n} \log p(\mathbf{x}_n,\mathbf{g},s;\boldsymbol{\theta}) \\ &\quad + \sum_n \mathcal{H}(Q_n(\mathbf{g},s)), \quad (2)\end{aligned}$$

where $\mathcal{D}_{KL}$ and $\mathcal{H}$ are used respectively to denote Kullback-Leibler divergence and entropy (Cover and Thomas, 1991). The first term in both decompositions implements a Gaussian prior distribution over the elements of the matrix $\mathbf{W}$, with inverse variance λ on each element. It makes sense to put a uniform prior on on the weights for the linear and constant basis-functions. However, to keep notation simple, we assume equal prior on all weights here.

The distributions $Q_n(\mathbf{g},s)$ over mixture components s and latent coordinates $\mathbf{g}$ are restricted to be independent over s and $\mathbf{g}$, and of the form:

$$Q_n(\mathbf{g},s) = q_{ns}Q_n(\mathbf{g}) \quad \text{and} \quad Q_n(\mathbf{g}) = \mathcal{N}(\mathbf{g}_n,\mathbf{\Sigma}_n)$$

It turns out that $\mathbf{\Phi}$ is maximized w.r.t. $\mathbf{\Sigma}_n$ if $\mathbf{\Sigma}_n = v^{-1}\mathbf{I}$, where $v = (\rho+1)/(\alpha^2\rho\sigma^2)$ is the inverse variance of $p(\mathbf{g} \mid \mathbf{x}, s)$. Hence, in the following we assume this equality and do not use $\mathbf{\Sigma}_n$ as a free parameter anymore. The $\{q_{ns}\}$ play the role of the 'responsibilities' in normal EM. Finally, $\mathbf{g}_n$ is the expected latent coordinate for data point $\mathbf{x}_n$, if we approximate the true distribution on the latent coordinate with a uni-modal Gaussian. Hence, we might use the $\mathbf{g}_n$ for visualization purposes.

4.2 Initialization

We initialize the model using Principal Component Analysis (PCA) (Jolliffe, 1986). Suppose, without loss of generality, that the data has zero mean. We use a projection of the data to the two dimensional PCA subspace to initialize the locations $\mathbf{g}_n$ of the distributions $Q_n(\mathbf{g})$. The columns of the loading matrices $\{\mathbf{\Lambda}_s\}$ are initialized as the principal eigenvectors of the data covariance matrix. The variance σ^2 is initialized as the mean variance in directions outside the PCA subspace. The $\{\boldsymbol{\kappa}_s\}$ are initialized on a square grid such that they have zero mean and the trace of their covariance matrix matches that of the covariance of the $\{\mathbf{g}_n\}$. The $\mathbf{W}$ is then taken as to map the $\{\boldsymbol{\kappa}_s\}$ onto $\{\mathbf{\Lambda}_s\boldsymbol{\kappa}_s\}$. Then, we assign the data point n to the component with $\boldsymbol{\kappa}_s$ closest to $\mathbf{g}_n$. Using the non-empty clusters, we compute the ρ and α. To initialize ρ, we use the mean value (over all non-empty clusters) of the total variance in the cluster in the data space minus $D\sigma^2$ and divide the sum over $d\sigma^2$. For α^2 we use the average of the mean variance in the cluster in the latent space and divide it over $\rho\sigma^2$. Finally, we can compute optimal q_{ns} and start the learning algorithm with a M-step of the EM-algorithm described in the next section.

The inverse variance λ of the prior on the elements of $\mathbf{W}$ can be made larger to get smoother functions. In our experiments we used $\lambda = 1$.

4.3 Updating the Parameters

Using the notation:

$$\begin{aligned}\mathcal{E}_{ns} = \frac{1}{2\sigma^2}\mathbf{x}_{ns}^\top\mathbf{x}_{ns} + \frac{v}{2}\mathbf{g}_{ns}^\top\mathbf{g}_{ns} - \sigma^{-2}\alpha^{-1}\mathbf{x}_{ns}^\top\mathbf{\Lambda}_s\mathbf{g}_{ns} \\ + D\log\sigma + \frac{d}{2}\log(\rho+1),\end{aligned}$$

$$\mathbf{x}_{ns} = \mathbf{x}_n - \mathbf{W}\boldsymbol{\phi}(\boldsymbol{\kappa}_s), \quad \text{and} \quad \mathbf{g}_{ns} = \mathbf{g}_n - \boldsymbol{\kappa}_s.$$

The objective (2) can be written as:

$$\mathbf{\Phi} = -\frac{\lambda}{2} \parallel \mathbf{W} \parallel^2 - \sum_{ns} q_{ns} \left[\log q_{ns} + \mathcal{E}_{ns} \right] + c,$$

where c is some constant in the parameters. All the update equations below can be found by setting the derivative of the objective to zero for the different parameters. First, we give the E-step update equations:

$$q_{ns} = \frac{e^{-\mathcal{E}_{ns}}}{\sum_{s'} e^{-\mathcal{E}_{ns'}}}, \qquad \mathbf{g}_n = \sum_s q_{ns} \langle \mathbf{g}_n \rangle_s,$$

where for all N data points the expected latent coordinate given component s is:

$$\langle \mathbf{g}_n \rangle_s = \mathrm{E}_{p(\mathbf{g}|\mathbf{x},s)}[\mathbf{g}] = \boldsymbol{\kappa}_s + \frac{\alpha\rho}{\rho+1} \mathbf{\Lambda}_s^\top \mathbf{x}_{ns}.$$

Next, we give the M-step update equations. For $\mathbf{\Lambda}_s$ we have to minimize:

$$\sum_n q_{ns} \mathbf{x}_{ns}^\top \mathbf{\Lambda}_s \mathbf{g}_{ns}.$$

This problem is known as the 'weighted Procrustes rotation' (Cox and Cox, 1994). Let

$$\mathbf{C} = [\sqrt{q_{1s}}\mathbf{x}_{1s} \cdots \sqrt{q_{Ns}}\mathbf{x}_{Ns}][\sqrt{q_{1s}}\mathbf{g}_{1s} \cdots \sqrt{q_{Ns}}\mathbf{g}_{Ns}]^\top$$

with SVD: $\mathbf{C} = \mathbf{UL\Gamma}^\top$, (the $\mathbf{g}_{ns}$ have been padded with zeros to form D-dimensional vectors). The optimal $\mathbf{\Lambda}_s$ is given by the first d columns of $\mathbf{U\Gamma}^\top$. For $\mathbf{W}$ we find:

$$\mathbf{W} = \Big[\sum_{ns} q_{ns} (\mathbf{x}_n - \alpha_s^{-1} \mathbf{\Lambda}_s \mathbf{g}_{ns}) \boldsymbol{\phi}(\boldsymbol{\kappa}_s)^\top \Big] \times \Big[\sum_{ns} q_{ns} \boldsymbol{\phi}(\boldsymbol{\kappa}_s) \boldsymbol{\phi}(\boldsymbol{\kappa}_s)^\top + \lambda\sigma^2 \mathbf{I} \Big]^{-1}.$$

For the updates of the three variance parameters we use the following notation:

$$A = \sum_{ns} q_{ns} \mathbf{x}_{ns}^\top \mathbf{\Lambda}_s \mathbf{g}_{ns}, \quad B = \sum_{ns} q_{ns} \mathbf{x}_{ns}^\top \mathbf{x}_{ns}, \quad C = \sum_{ns} q_{ns} \mathbf{g}_{ns}^\top \mathbf{g}_{ns}.$$

Simultaneously setting the derivatives of $\mathbf{\Phi}$ w.r.t. α, ρ and σ to zero gives:

$$\rho + 1 = \frac{(D-d)A^2}{d(BC - A^2)}, \quad \alpha = \frac{C(\rho+1)}{A\rho}, \quad \sigma^2 = \frac{B - \alpha^{-1}A}{ND}$$

5 Experimental Illustration

In this section we provide experimental results comparing the presented model with GTM. The first two experiments use artificially generated data. In the last experiment we use chemical chromatogram data from the Unilever research department.

Artificial data The first data set consisted of points in $\mathbb{R}^2$ drawn along the sine function. We used GTM (20 units, 5 basis-functions) and our new model (5 units, 3 basis-functions) to map the data to a one dimensional latent space. The fitted models and the latent space representations are given in Figure 1. We see that for GTM the distribution of the latent coordinates clusters around the mixture component locations. For our model, while using 4 times fewer mixture components, the latent coordinates do not show this clustering due to the piece-wise linear maps.

Next, we generated a data set in $\mathbb{R}^3$ by first drawing uniformly random from the unit square in $\mathbb{R}^2$, then to every data point $\mathbf{x}_n$ we then added a third coordinate $\mathbf{x}_n^{(3)}$ with:

$$\mathbf{x}_n^{(3)} = (\mathbf{x}_n^{(1)} - 1/2)^2 + 2(\mathbf{x}_n^{(2)} - 1/2)^3 + \epsilon,$$

where ϵ is a random variable with distribution $\mathcal{N}(\epsilon; 0, 100^{-1})$. In Figure 2 we show the projections found by our method and GTM. Again we see that GTM concentrates the latent coordinates around the locations of the units, the projection 'clumps' around the units. Whereas our method gives a projection that matches the original distribution on the unit square better.

Chromatogram data We demonstrate the performance of the proposed method and GTM on a problem from high-throughput screening of Maillard reactions, which involves the heating of sugars and amino acids, giving rise to complex mixtures of volatile flavor compounds. The reaction product mixtures are characterized through fast gas chromatography. At Unilever R&D labs, it was found that a single GTM plot of the chromatographic data, based on two latent variables, is equally informative as the screening of many score plots from various PCA combinations. The data was preprocessed by mapping it with PCA onto a 20 dimensional linear subspace of the original 2800 dimensional space.

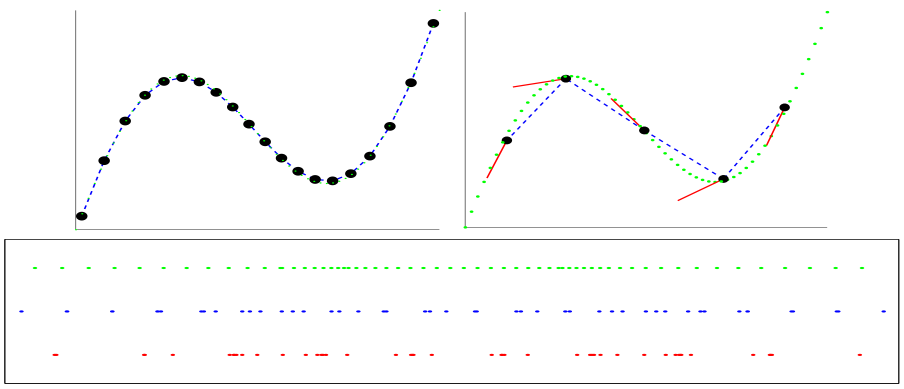


Figure 1: Top left panel: GTM fit (disks give means of mixture components, dashed line the ordering of the components in latent space) on data (dots). Top right panel: our new model (the sticks depict vectors $\sqrt{\rho}\sigma\mathbf{\Lambda}_s$ used for the linear maps). Bottom panel: (top) latent coordinates for our method, and GTM (middle 20 nodes, bottom 10 nodes). Both sets of latent points have the same variance.

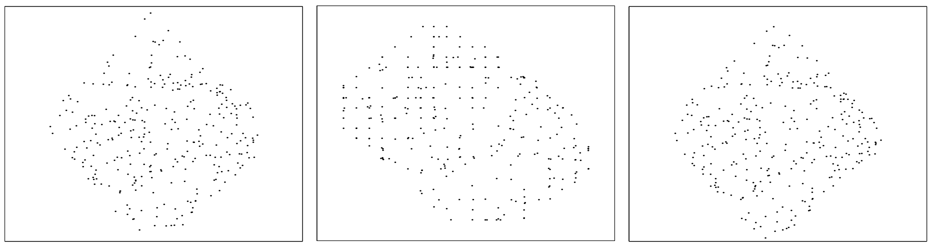


Figure 2: Projections of (from left to right) the proposed model, GTM and original 2D coordinates (which are rotated).

We call an ingredient separable if all the measurements on compounds which contain, or do not contain, the ingredient form a cluster in the projected data. In our experiment we found that most ingredients which were separable using GTM were also separable with our new projection method. In the experiments GTM used 400 nodes and 16 basis-functions, our model used only 36 units and 16 basis functions.

6 Discussion and Conclusions

Discussion Constraining the means of the mixture components to the manifold spanned by a smooth mapping ensures that components that are close in latent space will also be close in the data space. This makes the constrained version of Coordinated PCA more suitable for visualization purposes.

Comparing the presented model with GTM, we observe that we extended the point-wise coupling between data and latent space to a locally linear coupling. This removes the effect of 'clumpy' projections, as discussed in the previous section.

Several variations on the presented model are possible. For example, we might model the variances per mixture component and make the $\{\boldsymbol{\kappa}_s\}$ free parameters. However, here we wanted to keep the model close to GTM. As for the last option, the optimization w.r.t. the $\{\boldsymbol{\kappa}_s\}$ is a somewhat more involved due to the non-linear mappings $\boldsymbol{\phi}$. However, using local linear approximations of the non-linear functions we could do the optimization.

Note that the component covariance structure is not aligned with the non-linear function. However, in practice we *do* expect the local subspaces to be more or less aligned with the manifold due to the coordination (see Figure 1). To speed-up the algorithm, we might set the $\{\mathbf{\Lambda}_s\}$

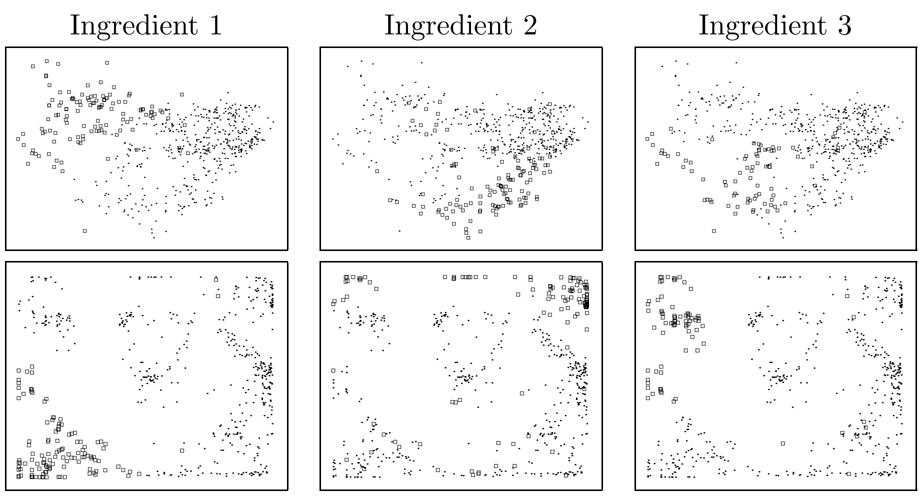


Figure 3: Projections of all the data for the proposed model (top) and GTM (bottom). From left to right we highlight compounds with one of three ingredient with boxes.

as manifold aligned and check whether this increases the objective function. We only need re-compute the $\{\mathbf{\Lambda}_s\}$ with SVD if taking them manifold aligned does not increase the objective function.

In (Bishop et al., 1998a) an extension of GTM is discussed where the the covariance matrix of $p(\mathbf{x}|s)$ is taken to be of the form $\sigma^2\mathbf{I} + \eta\mathbf{D}\mathbf{D}^\top$, where the d columns of the $D \times d$ matrix $\mathbf{D}$ contain the partial derivatives of the RBF network.[1] This generative model has a similar form of the covariance matrix in the data space as we use, however in the latent space it uses the same discrete distribution as GTM. Therefore, the extension described in (Bishop et al., 1998a), just like normal GTM, doesn't have the linear dependence of the expected value of $\mathbf{g}$ on $\mathbf{x}$ given s.

Conclusions We proposed a generative model and corresponding learning algorithm for visualization and projection of high-dimensional data. The model integrates the GTM and Coordinated PCA.

Experimental results show that the model does not suffer from the 'clumpy' projections of GTM. Furthermore, in the latent-space we have a true density function where GTM only provides a discrete 'spike' distribution on the latent-space. Also, the proposed model provides a density $p(\mathbf{x} \mid \mathbf{g})$, whereas for GTM this distribution is not clearly defined.

Finally, the proposed model provides a measure of uncertainty on the found latent coordinates summarized in the distributions $Q_n(\mathbf{g})$.

[1]The M step of the proposed EM algorithm is not exact, due to the dependence of $\mathbf{D}$ on $\mathbf{W}$.

Acknowledgments

We would like to thank Dick de Ridder and Sam Roweis for fruitful discussions. We would also like to thank Claire Boucon and Sijmen de Jong (Unilever R&D Vlaardingen, The Netherlands) for providing the Maillard reaction data and experimental results. This research is supported by the Technology Foundation STW (project nr. AIF4997) applied science division of NWO and the technology program of the Dutch Ministry of Economic Affairs.

References

C. M. Bishop, M. Svensén, and C. K. I. Williams. 1998a. Developments of the generative topographic mapping. *Neurocomputing*, 21:203–224.

C. M. Bishop, M. Svensén, and C. K. I Williams. 1998b. GTM: The generative topographic mapping. *Neural Computation*, 10:215–234.

T. Cover and J. Thomas. 1991. *Elements of Information Theory*. Wiley.

T.F. Cox and M.A.A. Cox. 1994. *Multidimensional Scaling*. Number 59 in Monographs on statistics and applied probability. Chapman & Hall.

I.T. Jolliffe. 1986. *Principal Component Analysis*. Springer-Verlag.

T. Kohonen. 2001. *Self-Organizing Maps*. Springer Series in Information Sciences. Springer-Verlag, Heidelberg, Germany.

R. M. Neal and G. E. Hinton. 1998. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M.I. Jordan, editor, *Learning in Graphical Models*, pages 355–368. Kluwer Academic Publishers, Dordrecht, The Netherlands.

B.D. Ripley. 1996. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, U.K.

S.T. Roweis, L.K. Saul, and G.E. Hinton. 2002. Global coordination of local linear models. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, USA. MIT Press.

S.T. Roweis. 1997. EM Algorithms for PCA and SPCA. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10, pages 626–632. MIT Press.

M.E. Tipping and C.M. Bishop. 1999. Mixtures of probabilistic principal component analysers. *Neural Computation*, 11(2):443–482.

J.J. Verbeek, N. Vlassis, and B. Kröse. 2002. Coordinating Principal Component Analyzers. In J.R. Dorronsoro, editor, *Proceedings of International Conference on Artificial Neural Networks*, Lecture Notes in Computer Science, pages 914–919, Madrid, Spain, August. Springer.

Detecting orthogonal class boundaries in entropy behaviour

Floor Verdenius
Agrotechnological Research Institute (ATO BV)
Wageningen University & Research Centre
PO box 17
6700 AA Wageningen
The Netherlands
F.Verdenius@ato.wag-ur.nl

Maarten van Someren
Department of Social Science Informatics
University of Amsterdam
Roetersstraat 15
1018 WB Amsterdam
The Netherlands
Maarten@swi.psy.uva.nl

Abstract

One of the key problems in machine learning is technique selection. Machine learning techniques construct models of a particular class. Finding an appropriate model class for a domain is an important step in technique selection. Here we focus on the model class of nested hyper-rectangles, orthogonal to numerical variables. We show that patterns in this class produce sharp and deep *cusps* in the distribution of information gain over variables. Cusps correspond to points on the scale of a variable where the distribution of information gain is non-differentiable. Next we describe a method based on wavelets that can recognise such cusps in noisy distributions. Using artificially generated data and UCI data sets, we show that the *cuspiness* of information gain distributions can be used to choose between methods for nested hyper-rectangles and methods for linear hyperplanes.

Introduction

Machine learning techniques construct models of data within a particular model class. That model class is usually characterised by the technique's hypothesis language. Heuristic machine learning techniques typically maximise a multi-attribute criterion that includes simplicity and fit with the data. Such maximisation approximates the best hypothesis within its model class. Given a set of data with a learning goal and a set of available techniques the choice of an appropriate technique is often difficult to make. One of the key factors in determining a technique is the choice of an appropriate model class. An inappropriate choice of model class and associated induction method leads to sub-optimal models in terms of generalisation, comprehensibility and model efficiency. For example, if class separation requires a split of the instance space that corresponds to a hyperplane, then learning a univariate decision tree will at best produce a complex tree as an approximation of the actual pattern. Vice versa, a hyperplane will be a poor approximation of a pattern that has the shape of nested hyper-rectangles, orthogonal to the variables. Both accuracy and comprehensibility will be sub-optimal.

This paper investigates whether it is possible to detect strict class boundaries for single attributes. Strict boundaries suggest on a single attribute suggest orthogonal nested hyper-rectangles as class boundaries. A gradual increasing probability of a class along an attribute suggests a hyperplane that is not orthogonal to attributes. We shall call "strict" boundaries *orthogonal boundaries* because they suggest orthogonal models and gradual distributions *linear* because they suggest a linear hyperplane model. Section 1 introduces entropy behaviour of the different concept types. Section 2 presents the Continuous Wavelet Transform (CWT) technique and how it is used to analyse entropy behaviour. Section 3 presents experimental evaluation of the approach. The results from our preliminary work lead to various discussion topics in section 4.

1 Entropy behaviour

1.1 Definition of Entropy behaviour

The entropy H(X) of a discrete random variable X is defined as

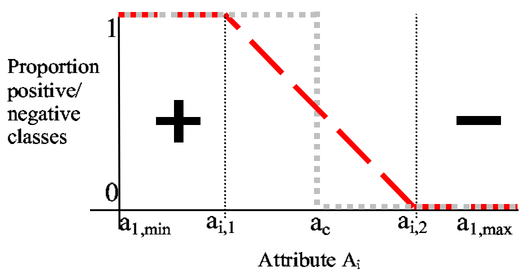


Figure 1. Class distributions for an orthogonal class boundary (dotted) A_i=a_c and a linear class boundary over A_i = [$a_{i,1}$, $a_{i,2}$].(dashed)

$$H(X) = -\sum_{x_i \in X} p(x_i)\log p(x_i)\text{,} \tag{1}$$

with the log to the base 2, and p(x) being the probability of X=x_i. The class distribution of a classified data set D with k_D elements serves as a random variable. The entropy over the class distribution of a data set is often used in machine learning techniques, such as decision tree learning (Mitchell, 1997, ch. 3, e.g. C4.5: Quinlan, 1993) and rule learning (Clark & Niblett, 1989) to evaluate the predictive value of a "split" of a variable into intervals. The variable and split that are most predictive (gain most information) is selected for building a decision tree or a rule. Here, however, we propose to analyse the distribution of information gain over the entire range of a variable to assess how well a complete "ONHS" model will explain the data. Let H_{tot} be the entropy over the class distribution of the total data set. When partitioned at A = a_i, the data set falls apart in two subsets $D_{A\leq ai}$ and $D_{Ai>ai}$. For these subsets, the partial entropies are $H(D_{A\leq ai})$ and $H(D_{A>ai})$. If the number of instances in the respective data sets is $k_{A\leq ai}$ and $k_{A>ai}$ respectively, the joint entropy of the partition A = a_i is defined as:

$$H(D_{A=a_i}) = \frac{k_{A\leq a_i}}{k_D} H(D_{A\leq a_i}) + \frac{k_{A>a_i}}{k_D} H(D_{A>a_i}) \tag{2}$$

Entropy behaviour $B_A(a_i)$ over numerical variable A is the function that couples every possible value a_i in A to the information gain: H_{tot}-$H(D_{A=ai})$. Given a new, previously unseen data set with an unknown underlying class distribution, the analytical form of B_A is unknown. However, it can be approximated in case a variable contains sufficient potential partition points a_i, that is, if the variable counts a substantial amount of different values.

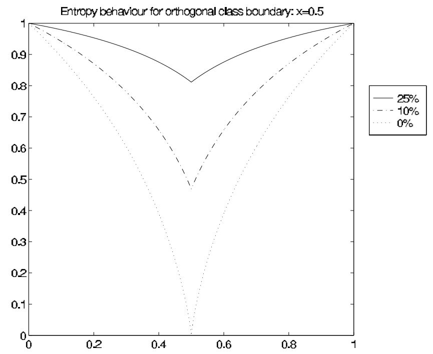


Figure 2. Theoretical entropy behaviour for an orthogonal class boundary at noise levels 0%, 10% and 25%.

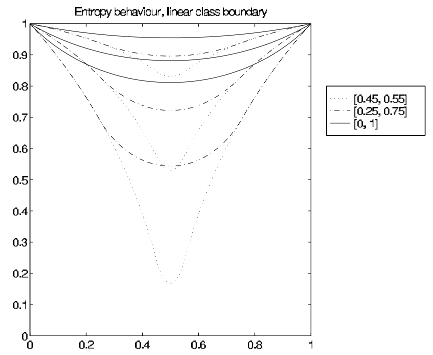


Figure 3. Theoretical entropy behaviour for linear class boundaries over three intervals A_i = [$a_{i,1}$, $a_{i,2}$] (see legend), at noise levels 0%, 10% and 25%.

1.2 Class boundaries and entropy behaviour

Figure 1 shows the class distributions for an orthogonal class boundary at A_i = a_c and for a linear class boundary over the interval A_i = [$a_{i,1}$, $a_{i,2}$]. Figure 2 shows the resulting entropy behaviour for a_c=0.5 for three noise levels. If we assume a noise free orthogonal class distribution with the class boundary at A_I = a_c, we have complete class information, and consequently $H(A_I$=$a_c)$ = 0. The minimum entropy value increases when the noise increases, but for all noise levels the curve steeply declines to a sharp minimum at A_I = a_c, independent of the noise level. For the example of a linear class boundary (figure 3) the overall entropy value is higher. In this case, there is a minimum as well, and we see it is lower if there is less noise. There is, however, no sharp cusp at the minimum.

Filling formula (2) with partial entropies, it can be proven that the entropy behaviour B_{Ai} cannot be differentiated towards A_i in A_i = a_c. This proof is omitted here due to space constraints.

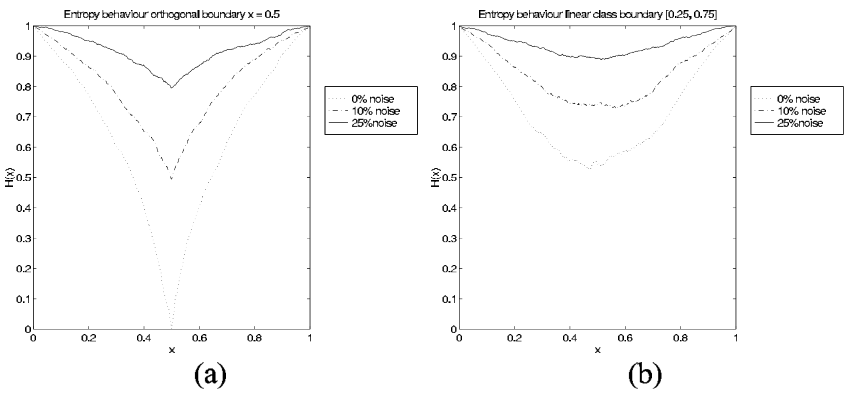


Figure 4. Entropy behaviour of a data sample from the class distributions of figure 1 with 0%, 10% and 25% noise. (a) orthogonal class boundary (b) linear class distribution.

We note, however, that for $\lim_{a_i \uparrow a_c}$ and $\lim_{a_i \downarrow a_c}$ the derivative results in -κ and κ respectively, with κ depending on the noise level.

Figure 3 shows the entropy behaviours for the linear class boundary of figure 1, over three different values $[a_{i,1}, a_{i,2}]$ for A_i. Filling in the partial entropies in (2) is more complex. The resulting entropy behaviour consists of three parts: $A_i \le a_{i,1}$, $A_i > a_{i,2}$, with a behaviour that follows the entropy behaviour for orthogonal class distributions, and $A_i \in < a_{i,1}, a_{i,2}]$. The latter shows a gradual curve. It can be proven that at $A_i = a_{i,1}$ and $A_i = a_{i,2}$, the entropy behaviour is differentiable.

1.3 Characteristics of entropy behaviour

In principle, when the underlying class distribution is known, the entropy behaviour can be modelled, as shown in the above sections. Figure 2 and 3 depict the theoretical behaviour of the orthogonal and linear class distributions. Orthogonal class boundaries are characterised by a steep cusp in the entropy behaviour, which is characterised by a discontinuity in the first derivative.

In the real world the underlying class distributions for a data set are unknown. The data set itself serves as sample from the underlying class distribution, and class boundaries reflect in entropy behaviour. To characterise the underlying class distribution, we can scan the entropy behaviour for the presence of cusps in the entropy behaviour.

Data sets of 2000 points are drawn from a 2-dimensional data space. Points are uniformly distributed over the data space. All cases belong to one of two classes c_1 (i.e. '+') or c_2 (i.e. '-'). The data space is split by a class boundary, dividing the space into two distributions. Probabilities are for class c_1 being $p(C=c_1) = n$ and $p(C=c_1) = 1-n$ respectively. Consequently the probabilities for class c_2 are $p(C=c_2) = 1-n$ and $p(C=c_2) = n$. The accuracy level n defines the probability that a record is classified correctly, and is supposed to be uniformly distributed over the data space. From this data set the entropy behaviour is derived. Attribute axes are scanned by defining partition points at the end of dyadic intervals (#intervals = 2^d, here d = 12). Every interval boundary $A=a_i$ is used to calculate $H(D_{A=ai})$. Figure 4 shows the entropy behaviour for orthogonal and linear class distributions at different noise levels.

The entropy behaviour of these data sets, as in the case of real world data, is not smooth. This is both due to noise in real-world data and irregularities in data distributions as a result of the sampling. Until now, we have discussed simple patterns in simulated data. Subtle wrinkles in the entropy behaviour can be indications for class boundaries at a deeper level in a concept hierarchy, or indications for multiple orthogonal class boundaries for a variable. We therefore need an analysis technique that can help to detect and quantify singularities in a noisy signal.

2 Continuous Wavelet Transform

Continuous Wavelet Transform (CWT; Mallat, 1999) is a mathematical instrument to analyse the local behaviour of signals at various levels of detail. The basic idea behind wavelet analyses is to describe a complex signal relative to a constant prototype signal, the *mother wavelet*. To gain local insight, this wavelet is translated over the total space of the complex signal to cover the total variable range, and dilated/contracted to cover the relevant frequency ranges. At each frequency, and at each translation, this results in a similarity

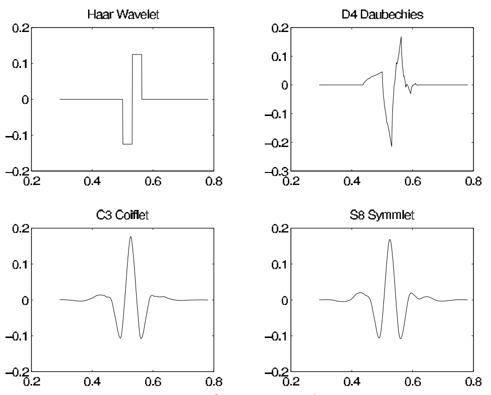


Figure 5. Four mother wavelets

measure between the complex signal and the wavelet. In other words, a CWT is a 2-dimensional data structure that describes at each point over the variable range, and at each scale level, the match between the signal at hand and the mother wavelet. The mother wavelet is chosen from a specific set of mother wavelets, whose analytical properties are readily accessible. This way the CWT reveals a wealth of information on the original signal. CWT is used for various applications, including signal analysis, denoising and signal compression. It is especially applicable to non-stationary signals (in which it differs from Fourier analysis).

2.1 CWT Basics

A wavelet is a function $\psi(x)$ with as two main properties that it integrates to zero ($\int \psi(x)dx = 0$, *Waving above and below the x-axis*) and that it has a compact support, that is, if $x \notin [a,b]$, with $a<b$, then $\psi(x) = 0$. Numerous mother wavelets $\psi(x)$ have been proposed. Wavelab, a wavelet library (Stanford 1996) for Matlab, counts about ten different mother wavelets. Figure 5 presents some often-used mother wavelets.

In accordance with Mallat (1999, pp 80), we adopt $\psi(x)$ to be the second derivative of the Gaussian kernel, the Mexican hat function:

$$\psi(t) = \frac{2}{\pi^{\frac{1}{4}}\sqrt{3\sigma}}\left(\frac{t^2}{\sigma^2} - 1\right)e^{\frac{-t^2}{2\sigma^2}} \qquad (3)$$

not shown in figure 5, here normalised for mean π and standard deviation σ. A signal f(x) is analysed by convoluting it with a contracted and dilated version of this mother wavelet. The result is a multidimensional numerical representation: the continuous wavelet transform $Wf(\tau,s)$:

$$Wf(\tau,s) = \frac{1}{\sqrt{s}}\int f(x)\psi\left(\frac{x-\tau}{s}\right)dx, \qquad (4)$$

with τ being the translation over the x-axis, and s the scale of analysis.

Figure 1 depicts an orthogonal class distribution. The entropy behaviour for a sample, drawn from this distribution, is shown in figure 4. The resulting continuous wavelet transform is shown in figure 6a and 6b. Figure 6a contains a surface plot. Large absolute values for cwt coefficients are coloured white, small absolute values are coloured black. Figure 6b also represents that CWT, but now, a curve for each scale level s is drawn. The curve with minimum and maximum values and the gradual curvature represent the lowest CWT scale (top of figure 6a), while the relative flat, but turbulent curves represent the finer scales (bottom of figure 6a).

2.2 Detecting singularities

The task is to detect discontinuities in the derivative of entropy behaviours. In the wavelet vocabulary, these discontinuities are characterised as singularities, or cusps. The Lipschitz α describes the regularity of signal. A singularity is characterised by a $\alpha<1$. A function f(x) is Lipschitz α in x_0 if:

$$|f(x_0+h) - f(x_0)| \le K|h|^{\alpha} \qquad (5)$$

for some positive value of K if $h \to 0$. The quantity α can be understood as the continuous equivalent of *times differentiable*. A bounded discontinuity at x_0 has a Lipschitz α of 0 in x_0. For points x_0 with a local $0 < \alpha < 1$, the function is not differentiable in x_0.

Mallat (1999, pp 169-173) shows that the local regularity of a signal at a specific point x_0 depends on the decay at fine scales of $|Wf(\tau,s)|$ in the neighborhood of x_0. This decay can be measured directly from $Wf(\tau,s)$, but this is unnecessarily complex. It can also be obtained (4)
from the local absolute maximum values.

In the CWT, at each scale s, strict local maxima are called *modulus maxima*. A *maxima line* is a line across scales, for which all points are modulus maxima. The collection of all maxima lines for a CWT Wf(u,s) is called the Wavelet Transform Modulus Maxima (WTMM). Figure 6c shows the maxima lines for the CWT (the numbered lines) of figure 6a.

One point of caution in drawing conclusions from a WTMM is maxima line interference. CWT analyses a signal with a dilated and translated version of the mother wavelet. At each scale, signal effects in the environment of the point of analysis, influence the CWT coefficients. In order to draw conclusions from the maxima lines, they may not be *too closely* related, where *too closely* is defined in terms of the *Cone of Influence*. This Cone of Influence (COI) is the area around a ridge where other ridges, if present, may influence the coefficient values. When analysing ridges, the section of the COI where no other ridges interfere is prone for analysis. Figure 6c gives the COI for the significant ridges 1 and 2 (unnumbered lines). Mallat shows that a local singularity ($\alpha < 1$) only exists for x_0 when there is a maxima line converging to x_0 at finer scales.
Mallat (1999) further shows that for an isolated singularity in x_0 the following holds for a cusp singularity:

$$|Wf(u,s)| \le As^{\alpha+\frac{1}{2}} \Leftrightarrow \log_2|Wf(u,s)| \le \log_2 A + (\alpha + \frac{1}{2})\log_2(s) \qquad \textbf{(6)}$$

Now, cusp singularities can be detected and their α can be estimated from the decay of the CWT coefficients along the modulus maxima line. More specifically, the local Lipschitz regularity of a function at a singularity is estimated from the maximum slope a of the ${}^{2}log|Wf(u,s)|$ as a function of ${}^{2}log(s)$ along the maxima lines by:

$$\alpha + 0.5 = a \qquad (7)$$

At coarse scales, the maxima related to the signal behaviour (including cusps) dominate. In the finest scales the wavelet maxima become unstable, due to noise induced interactions between fine scale maxima. Consequently, determining α requires selection of the appropriate s_{crit}, such that the largest scale J is smaller than the distance between to maxima. In practice, the value for s_{crit} are fixed (e.g. max(s) –5). For the coarsest level, two options exist. Mallat proposes a small range from s_{crit} upwards. Struzik (2001) suggests taking it up to the coarsest scale. We follow Struzik in case of stable slope. When the slope is unstable, we determine the maximum slope over a smaller range:

$$\alpha = \frac{{}^{2}\log|Wf(s_{lo})| - {}^{2}\log|Wf(s_{crit})|}{{}^{2}\log|s_{lo}| - {}^{2}\log|s_{crit}|}, \qquad \textbf{(8)}$$

with s_{lo} begin the coarsest scale in case of stable behaviour, and the scale range for the maximum slope in the case of unstable behaviour. In all, this analysis is restricted to that scale range in the COI that is free of other curves. By following the WTMM line towards the finest scale, the location of a singularity can be detected.

2.3 CWT analysis of Entropy behaviour

With this definition of ψ entropy behaviour is analysed. Given a signal and the mother wavelet, the CWT (**3**) and WTMM are calculated. Significant ridges are identified, and for each of the significant ridges the α value is calculated (**7**) to assess whether the ridge describes a cusp in the data, or a differentiable extreme. The free interval in the COI is assessed, and if the free interval fulfils a selection criterion, a ridge is supposed to identify an orthogonal class boundary. The criterion assesses:

- *Cusp scale persistency (persist)*, the scale interval where the ridge lies free in the COI. We use values of around 0.5.
- *Dynamic amplitude range (dynrange)*, the relative amplitudes of a cusp. These parameters lead to relatively small (opposed to the norm of all cusps) wrinkles being ignored. Typical values are in [0.7,0.75].
- *Statistical significance threshold*, the significance of the entropy gain. This measures whether the entropy gain at the actual location is significant at all. This threshold is set at 0.1 %.

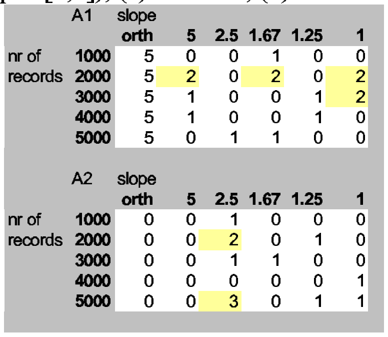
3 Evaluation

Generated data

This section assesses in two experiments the potential of the CWT approach for identifying orthogonal class boundaries on artificial data. A first experiment uses data with known underlying class distributions. Goal of this exploration is to explore the limitations of distinguishing orthogonal class boundaries from linear boundaries. In line with figure 1, 2-dimensional data sets (x, y $\in$ [0,1]) are generated with orthogonal and linear class boundaries. The slopes of the linear boundaries vary from 1 to 5. Class boundaries cross the point (0.5, 0.5). Set sizes vary from 1000 to

5000 records to assess the influence of the number of records. For each setting 5 repeats are drawn. Noise levels are set at 0% and 25%. Records are drawn randomly with a homogeneous distribution.
In table 1 the results are presented. Attribute A1 contains sharp cusps in the orthogonal case, and gradual minima in the case of slopes $\in$ [1,5]. With dynrange=0.7 and persist= 0.5 all cusps in A1 are detected when noise is 0%. With 25% noise, some cusps are omitted, especially in small data sets where the impact of random fluctuations in the data distribution increases. When looking at the various linear data sets it shows that for several slope-#records combinations relatively high numbers of cusps are detected (yellow cells in table 1; e.g. 25% noise, slope=2.5, #records =3000: 3 cusps). The A_1-values for assumed cusps however, are fairly constant in the case of an orthogonal class boundary (σ_{Ai}<0.5% of the variable range) but highly variable for the linear class boundaries (up to σ_{Ai}>10% of the variable range, with a minimum of σ_{Ai}> 2%).
In a second experiment a data set with 5 attributes $[A_1..A_5] \in [0,1]$ was generated. Class labels were assigned according to the following schema:

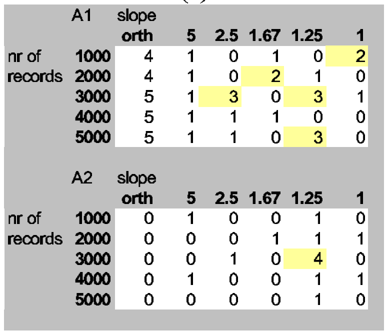
Table 1. Number of identified cusps in 5-fold 2-dimensional orthogonal (orth) and linear data sets with (slope$\in$[1,5]), (a) 0%noise, (b) 25% noise.



A1		slope					
		orth	5	2.5	1.67	1.25	1
nr of	1000	5	0	0	1	0	0
records	2000	5	2	0	2	0	2
	3000	5	1	0	0	1	2
	4000	5	1	0	0	1	0
	5000	5	0	1	1	0	0

A2		slope					
		orth	5	2.5	1.67	1.25	1
nr of	1000	0	0	1	0	0	0
records	2000	0	0	2	0	1	0
	3000	0	0	1	1	0	0
	4000	0	0	0	0	0	1
	5000	0	0	3	0	1	1

(a)



A1		slope					
		orth	5	2.5	1.67	1.25	1
nr of	1000	4	1	0	1	0	2
records	2000	4	1	0	2	1	0
	3000	5	1	3	0	3	1
	4000	5	1	1	1	0	0
	5000	5	1	1	0	3	0

A2		slope					
		orth	5	2.5	1.67	1.25	1
nr of	1000	0	1	0	0	1	0
records	2000	0	0	0	1	1	1
	3000	0	0	1	0	4	0
	4000	0	1	0	0	1	1
	5000	0	0	0	0	1	0

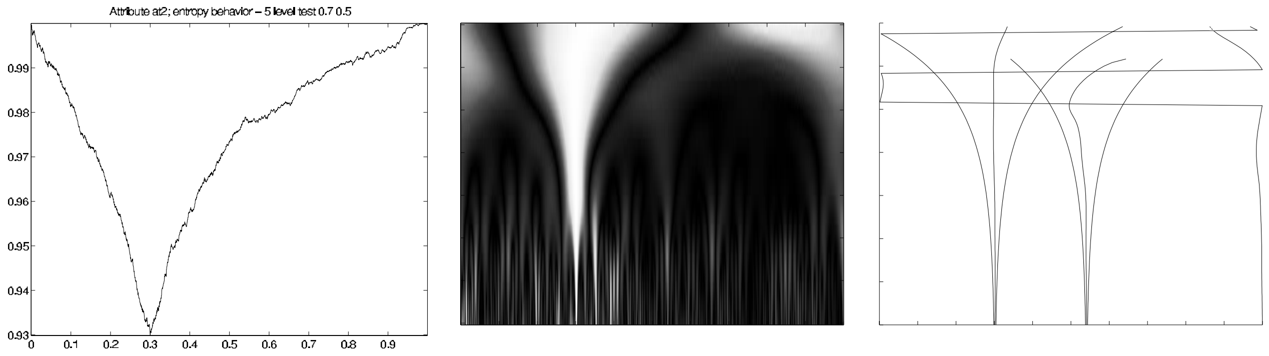
(b)

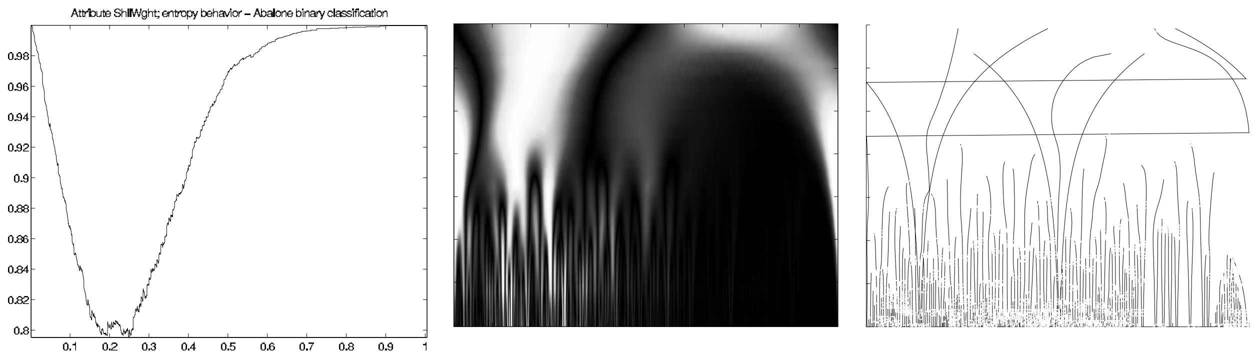
```
Class C1:
   A1> 0.5 AND A2≤0.3, OR
   A1> 0.5 AND A3>0.7, OR
   A1≤ 0.5 AND A4≤0.3, OR
   A1≤ 0.5 AND A5>0.7
Class C2:
   Otherwise
```

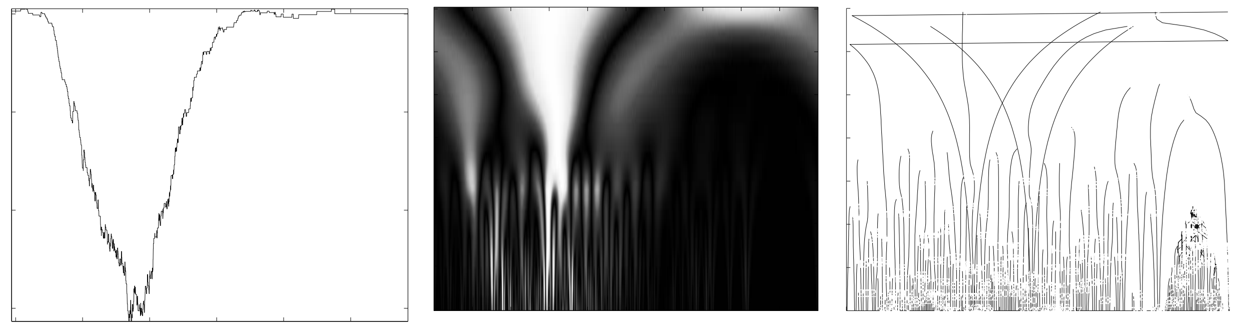
The class distribution over A_1 is fairly constant (cf. XOR distributions) and consequently exhibits no expressive entropy behaviour. The other attributes show cusps. All cusps are detected with parameter values dynrange=0.75 and presists=0.5. Figure 6 gives the entropy behaviour, CWT and WTMM for attribute A_2. Cusps 1 and 3 are tested on significance (α<0.5), which only applies to cusp 1.

UCI data

From the UCI machine learning repository we extracted two data sets with numeric attributes with large numbers of different values: *Abalone* and the *Wisconsin breast cancer (wdbc)*. The data set abalone with 29 classes was transformed to a 2-class data set. The class labels *small* and *large* are determined such that the set splits in portions of equal size (2096 and 2081 respectively). Moreover, the nominal attribute sex was removed from the set. In both data sets cusps are detected, but the largest entropy reductions are not coinciding with these cusps. In a univariate decision tree learner, abundant partitions may occur. Evaluation was done with the j48 decision tree learner in WEKA (Witten and Frank, 1999) and the linear classification tree generator Quest (Loh and Shih, 1997), both with standard settings. The results on these data sets are listed in table 2.
In the abalone case, there are three significant cusps in three related weight parameters (correlations of .93, .91 and .88). It seems justified to state that there is one clear cusp in this data set, but that this attribute does not deliver the maximum entropy gain. The resulting univariate decision tree counts 191 branches and 96 leaf nodes. Each variable occurs 20 to 38 times. A lack of cusps suggests that a linear classification tree may be more appropriate. This is confirmed by the results, especially by the reduced model size.
For the wdbc data, the situation is somewhat different. The data set consists of mean values (mn), standard measurement errors (se) and worst values (ws) for 10 measurements. The







data is linearly separable using all 30 parameters. Moreover, best predictive accuracy results from a linear hyper-plane of 3 variables. Cusp analysis shows that 12 variables are likely to have cusp-like minima, while 18 variables have non-cusp minima. Moreover, se-variables show a typical frequency pattern that may require normalisation (not yet included in our approach), undermining cusp detection. The decision tree for wdbc data counts 9 attributes. For 5 of these attributes, cusp analysis shows that linear tests could have been appropriate. The performance of this decision tree (94.2%) is not much less then a linear classification tree (96.8%). These results suggest that univariate decision points already give a reliable prediction for class membership, which is in line with both the statements in the data description and the cuspy character of the data. Figure 7 presents entropy behaviour, CWT and WTMM for two attributes. Shell weight is the attribute that is partitioned in the top decision node of the abalone decision tree. It's entropy behaviour has no significant cusps. Mean texture is an attribute in wdbc, where the partition point corresponds (almost exactly) with a partition point in the decision tree.

Table 2. Results for Abalone and WDBC data set. For explanation: see text

	Abalone	**WDBC**
Data set		
#attributes	8	31
#records	4177	569
#cusps	3	13
Decision Tree		
#nodes	191	25
#leaf nodes	96	13
Minimal repeat per attribute	20	1
Maximal repeat per attribute	38	2
Maximal depth	14	6
Accuracy(%)	77.68	94.2
#nodes on cusps	0	2(3)
Linear Classification Tree		
#nodes	27	3
#leaf nodes	14	2
Accuracy(%)	79.1	96.8

Discussion and conclusion

We presented an approach to detect orthogonal class boundaries in pre-classified data sets. One of the goals of our work is to evaluate the model class of a method before actually applying the method. This is useful because it guides design decisions by insight in underlying patterns and because it may gain speed.

Detecting orthogonal class boundaries is non-trivial problem. Even when we know the model class of a learning method, using this knowledge may be more expensive then simply trying out the method. An early example of this type of analysis is the observation that 2-layer perceptrons cannot learn non-linear functions. This knowledge does not give us a method for detecting if a pattern has a linear structure that enables to decide about 2-layer perceptrons. We focused here on hyper rectangles as model class.

The presence of numerous orthogonal class boundaries may lad to the induction of univariate concept hierarchies as the appropriate method. Absence of orthogonal class boundaries on the other hand suggests that univariate concept hierarchies may not be the best representation. CWT are used as tool to detect cusps in data. The experiments show that if the density of the data is not very high, spurious cusps can be found. The analysis still proves sensitive for parameter adjustments, especially *cusp scale persistency* and *dynamic amplitude range.*

Our approach detects cusps in individual variables. A further problem is what the presence or absence of cusps says about the overall pattern. We are considering two approaches to this problem. One is an application of cusp detection in hybrid methods such as linear classification trees (Loh and Shih, 1997) as in Van der Ham (2002). Cusp analysis of entropy behaviour for the maximum entropy gain may reveal whether a linear or orthogonal split is appropriate.

The other approach is to combine information about cusps with correlations between variables and the information gained by constructing intervals at cusps to assess if a hyperrectanglular model class is appropriate.

Acknowledgements

Our thanks go to Zbigniew R. Struzik, Bob Wielinga and Ferenc v.d. Ham for their contributions, comments and discussions. This work co-funded by DWK program 391 of the Dutch Ministry of Agriculture, Nature Management and Fisheries.

References

P. Clark and T. Niblett (1989), The CN2 Induction Algorithm. *Machine Learning*, 3(4):261-283

U. Fayyad, K.B. Irani (1993). Multi-Interval discretization of continuos attributes as pre-processing for classification learning. In Proceedings of the 13th international Join Conference on Artificial Intelligence, Morgan Kaufmann, pp. 1022-1027.

F. van den Ham (2002), *Induction o multivariate decision trees based on orthogonal class boundaries* (in Dutch), MSc thesis, Amsterdam

W.L. Loh and Y-S Shih (1997), Split Selection Methods for Classification Trees, in: *Statistica Sinica*, Vol. 7, pp. 815-840

S. Mallat (1999), *A wavelet tour of signal processing* (2^{nd} edition), Academic Press, San Diego (CA)

T.M. Mitchell (1997), *Machine Learning*, McGraw Hill

J.R. Quinlan (1993), *C4.5, Programs for Machine Learning*, Morgan Kaufmann, San Mateo (CA)

Stanford University (1996), Wavelab .701, at: http://playfair.stanford.edu/~wavelab

I.H. Witten and E. Frank (1999), WEKA Machine Learning Algorithms in Java, in: I.H. Witten & E. Frank, *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*, Morgan Kaufman,

Supervised Learning of Bayesian Network Parameters Made Easy

Hannes Wettig⋆, Peter Grünwald◦, Teemu Roos⋆, Petri Myllymäki⋆, and Henry Tirri⋆

⋆ Complex Systems Computation Group
Helsinki Inst. for Inf. Tech. (HIIT)
University of Helsinki
& Helsinki University of Technology
P.O. Box 9800, FIN-02015 HUT, Finland
{Firstname}.{Lastname}@hiit.fi

◦ CWI
P.O. Box 94079
NL-1090 GB Amsterdam, The Netherlands.
Peter.Grunwald@cwi.nl

Abstract

Bayesian network models are widely used for supervised prediction tasks such as classification. Usually the parameters of such models are determined using 'unsupervised' methods such as maximization of the joint likelihood. In many cases, the reason is that it is not clear how to find the parameters maximizing the supervised (conditional) likelihood. We show how the supervised learning problem can be solved efficiently for a large class of Bayesian network models, including the Naive Bayes (NB) and tree-augmented NB (TAN) classifiers. We do this by showing that under a certain general condition on the network structure, the supervised learning problem is exactly equivalent to logistic regression. Hitherto this was known only for Naive Bayes models. Since logistic regression models have a concave log-likelihood surface, the global maximum can be easily found by local optimization methods.

1 Introduction

In recent years it has been recognized that for supervised prediction tasks such as classification, we should use a supervised learning algorithm such as supervised (conditional) likelihood maximization (Friedman et al., 1997; Greiner et al., 1997; Ng and Jordan, 2001; Kontkanen et al., 2001; Greiner and Zhou, 2002). Nevertheless, in most related applications the model parameters are still determined using unsupervised methods such as maximization of the unsupervised (joint) likelihood or (ordinary, unsupervised) Bayesian methods. One of the main reasons for this discrepancy is the difficulty in finding the global maximum of the supervised likelihood. In this paper, we show that this problem can be solved for Bayesian network models, as long as they satisfy a particular additional condition. The condition is satisfied for many existing Bayesian-network based classifiers including the Naive Bayes (NB), TAN (tree-augmented NB) and 'diagnostic' classifiers (Kontkanen et al., 2001).

We find the maximum supervised likelihood parameters by parametrizing our models in a different manner; roughly speaking, the parameters in our parametrization correspond to logarithms of parameters in the standard Bayesian network parametrization. In this way, each conditional Bayesian network model is mapped to a logistic regression model. However, in some cases the parameters of this logistic regression model are not allowed to vary freely. In other words, the Bayesian network model corresponds to a subset of a logistic regression model rather than a 'full' logistic regression model.

We provide a general condition on the network structure under which, as we prove, the Bayesian network model is mapped to a full logistic regression model with freely varying parameters. The supervised log-likelihood for logistic regression models is a concave function of the parameters. Since, under our condition, these parameters are allowed to vary freely over $\mathbb{R}^k$ for some k, our condition implies that in the new parametrization the supervised log-likelihood becomes a concave function of parameters in a convex set. This implies that we can find the global maximum supervised likelihood parameters by simple local optimization techniques such as hill climbing.

We remark that viewing Bayesian network classifiers as logistic regression models is not new; it was used earlier in papers such as (Heckerman and Meek, 1997a; Ng and Jordan, 2001; Greiner and Zhou, 2002). Also, the concavity of the log-likelihood surface for logistic regression is known. Our main contribution is to supply the condition under which Bayesian network models correspond to logistic regression with *completely freely varying parameters.* Only in this latter case can we guarantee that there are no local maxima in the likelihood surface. As a direct consequence of our result, we show for the first time that the supervised likelihood of, for instance, the tree-augmented Naive Bayes (TAN) model has no local maxima.

This paper is organized as follows. In Section 2 we introduce Bayesian networks and an alternative so-called L-parametrization. In Section 3 we show that the L-parametrization allows us to consider Bayesian network classifiers as logistic regression models. Based on earlier results in logistic regression, we conclude that in the L-parametrization the supervised log-likelihood is a concave function. In Section 4 we present our main result, giving conditions under which the two parametrizations correspond to exactly the same conditional distributions and the L-parametrization preserves all the independence assumptions encoded by the network structure. Conclusions are summarized in Section 5.

2 Bayesian Networks

We assume that the reader is familiar with the basics of the theory of Bayesian networks see, e.g., (Pearl, 1988).

Consider a random vector $X = (X_0, X_1, \ldots, X_{M'})$, where each X_i takes values in $\{1, \ldots, n_i\}$. Let $\mathcal{B}$ be a Bayesian network structure over X, which factorizes $P(X)$ into

$$P(X) = \prod_{i=0}^{M'} P(X_i \mid Pa_i), \qquad (1)$$

where $Pa_i \subseteq \{X_0, \ldots, X_{M'}\}$ is the parent set of variable X_i in $\mathcal{B}$.

We are interested in predicting some class variable X_m for some $m \in \{0, \ldots, M'\}$ conditioned on all X_i, $i \neq m$. Without loss of generality we may assume that $m = 0$ (i.e., X_0 is the class variable) and that the children of X_0 in $\mathcal{B}$ are $\{X_1, \ldots, X_M\}$ for some $M \leq M'$. For instance, in the so-called Naive Bayes model (leftmost picture in Figure 1), we have $M = M'$ and the children of the class variable X_0 are independent given the value of X_0. The Bayesian network model corresponding to $\mathcal{B}$ is the set of all distributions satisfying the conditional independencies encoded in $\mathcal{B}$. It is usually parametrized by vectors $\Theta^{\mathcal{B}}$ with components of the form $\theta^{\mathcal{B}}_{x_i|pa_i}$ defined by

$$\theta^{\mathcal{B}}_{x_i|pa_i} := P(X_i = x_i \mid Pa_i = pa_i), \qquad (2)$$

where pa_i is any configuration (set of values) for the parents Pa_i of X_i. Whenever we want to emphasize that each pa_i is determined by the complete data vector $x = (x_0, \ldots, x_{M'})$, we write $pa_i(x)$ to denote the configuration of Pa_i in $\mathcal{B}$ given by the vector x. For a given data vector $x = (x_0, x_1, \ldots, x_{M'})$, we sometimes need to consider a modified vector where x_0 is replaced by x'_0 and the other entries remain the same. We then write $pa_i(x'_0, x)$ for the same configuration given by $(x'_0, x_1, \ldots, x_{M'})$.

We let $\mathcal{M}^{\mathcal{B}}$ be the set of *conditional* distributions $P(X_0 \mid X_1, \ldots, X_{M'}, \Theta^{\mathcal{B}})$ corresponding to distributions $P(X_0, \ldots, X_{M'} \mid \Theta^{\mathcal{B}})$ satisfying the conditional independencies encoded in $\mathcal{B}$. The conditional distributions in $\mathcal{M}^{\mathcal{B}}$ can be written as

$$P(x_0 \mid x_1, \ldots, x_{M'}, \Theta^{\mathcal{B}}) = \frac{\theta^{\mathcal{B}}_{x_0|pa_0(x)} \prod_{i=1}^{M'} \theta^{\mathcal{B}}_{x_i|pa_i(x)}}{\sum_{x'_0=1}^{n_0} \theta^{\mathcal{B}}_{x'_0|pa_0(x)} \prod_{i=1}^{M'} \theta^{\mathcal{B}}_{x_i|pa_i(x'_0,x)}}, \qquad (3)$$

extended to N outcomes by independence.

Given a complete data-matrix $D = (x^1, \ldots, x^N)$, the *supervised log-likelihood*, $S^{\mathcal{B}}(D; \Theta^{\mathcal{B}})$, of parameters $\Theta^{\mathcal{B}}$ is given by

$$S^{\mathcal{B}}(D; \Theta^{\mathcal{B}}) := \sum_{j=1}^{N} S^{\mathcal{B}}(x^j; \Theta^{\mathcal{B}}), \qquad (4)$$

where

$$S^{\mathcal{B}}(x; \Theta^{\mathcal{B}}) := \log P(x_0 \mid x_1, \ldots, x_{M'}, \Theta^{\mathcal{B}}). \qquad (5)$$

Note that in (3), and hence also in (4), all $\theta^{\mathcal{B}}_{x_i|pa_i}$ with $i > M$ (standing for nodes that are neither the class variable nor any of its children) cancel out, since for these terms we have $pa_i(x) \equiv pa_i(x'_0, x)$ for all x'_0. Thus the only relevant parameters for determining the conditional likelihood are of the form $\theta^{\mathcal{B}}_{x_i|pa_i}$ with $i \in \{0..M\}$, $x_i \in \{1..n_i\}$ and pa_i any configuration of values of Pa_i. We order these parameters lexicographically and define $\mathbf{\Theta}^{\mathcal{B}}$ to be the set of vectors constructed this way, with $\theta^{\mathcal{B}}_{x_i|pa_i} > 0$ and $\sum_{x_i=1}^{n_i} \theta^{\mathcal{B}}_{x_i|pa_i} = 1$ for all $i \in \{0,\ldots,M\}$, x_i and all values (configurations) of pa_i. Note that we require all parameters to be strictly positive.

The model $\mathcal{M}^{\mathcal{B}}$ does not contain any notion of the 'unsupervised' distributions: Probabilities such as $P(X_i \mid Pa_i)$, where $M < i \leq M'$, are undefined, and neither are we interested in them. Our task is prediction of X_0 *given* $X_1,\ldots,X_{M'}$. Heckerman and Meek call models such as $\mathcal{M}^{\mathcal{B}}$ *Bayesian regression/classification* (BRC) models (Heckerman and Meek, 1997a; Heckerman and Meek, 1997b).

For an arbitrary supervised Bayesian network model $\mathcal{M}^{\mathcal{B}}$, we now define the so-called L-model, another set of conditional distributions $P(X_0 \mid X_1,\ldots,X_{M'})$. This model, which we denote $\mathcal{M}^L$, is parametrized by vectors Θ^L in some set $\mathbf{\Theta^L}$ that closely resembles $\mathbf{\Theta}^{\mathcal{B}}$. Each different $\mathcal{M}^{\mathcal{B}}$ will give rise to a corresponding $\mathcal{M}^L$, although we do not necessarily have $\mathcal{M}^{\mathcal{B}} = \mathcal{M}^L$. For each component $\theta^{\mathcal{B}}_{x_i|pa_i}$ of each vector $\Theta^{\mathcal{B}} \in \mathbf{\Theta}^{\mathcal{B}}$, there is a corresponding component $\theta^L_{x_i|pa_i}$ of the vectors $\Theta^L \in \mathbf{\Theta^L}$. The components $\theta^L_{x_i|pa_i}$ take values in the range $(-\infty,\infty)$ rather than $(0,1)$. Each vector $\Theta^L \in \mathbf{\Theta^L}$ defines the following conditional distribution:

$$P(x_0 \mid x_1,\ldots,x_{M'},\Theta^L) := \frac{\exp\theta^L_{x_0|pa_0(x)} \prod_{i=1}^{M} \exp\theta^L_{x_i|pa_i(x)}}{\sum_{x_0'=1}^{n_0} \exp\theta^L_{x_0'|pa_0(x)} \prod_{i=1}^{M} \exp\theta^L_{x_i|pa_i(x_0',x)}}. \quad (6)$$

The model $\mathcal{M}^L$ is the set of conditional distributions $P(X_0 \mid X_1,\ldots,X_{M'},\Theta^L)$ indexed by $\Theta^L \in \mathbf{\Theta^L}$, extended to N outcomes by independence. Given a data-matrix D, let $S^L(D;\Theta^L)$ be the supervised log-likelihood of parameters Θ^L, defined analogously to (4) with (6) in place of (3).

Theorem 1. *$\mathcal{M}^{\mathcal{B}} \subseteq \mathcal{M}^L$.*

Proof. The theorem is immediate from doing the log-parameter transformation, i.e., setting $\theta^L_{x_i|pa_i} = \log\theta^{\mathcal{B}}_{x_i|pa_i}$ for all i, x_i and pa_i. ∎

In words, all the conditional distributions that can be represented by parameters $\Theta^B \in \mathbf{\Theta}^{\mathcal{B}}$ can also be represented by parameters $\Theta^L \in \mathbf{\Theta^L}$. However, whereas in the usual parametrization we require $\sum_{x_i=1}^{n_i} \theta^{\mathcal{B}}_{x_i|pa_i} = 1$ for each $i \in \{0,\ldots,M'\}$ and pa_i, there is no corresponding condition for the parameters of the L-model. Consequently, the converse of Theorem 1, i.e., $\mathcal{M}^L \subseteq \mathcal{M}^{\mathcal{B}}$, is true only under some additional conditions on the network structure. We return to this topic in Section 4 but first we take a closer look at the L-model.

3 The L-model Viewed as Logistic Regression

Although the L-model is closely related to and in some cases formally identical to Bayesian network classifiers, it can also be interpreted in terms of *logistic regression*. We can think of the conditional model $\mathcal{M}^L$ as a predictor that combines the information of the attributes using the so-called *softmax* rule (Bishop, 1995; Heckerman and Meek, 1997b; Ng and Jordan, 2001). Figure 1 gives an interpretation of this, depicting a Naive Bayes model and the corresponding L-model in their Bayesian network guises.

In terms of logistic regression, the L-model has one (binary) regressor variable for each configuration of each of the parent sets Pa_i, and one (binary) output variable for each possible value of the class variable. Having established that the L-model is a logistic regression model, we may use a well-known fact that holds for logistic regression models in general. Namely, the supervised log-likelihood in the L-parametrization is a concave function of the parameters:

Theorem 2. *(Santner and Duffy, 1989) The parameter set $\mathbf{\Theta^L}$ is convex, and the supervised log-likelihood $S^L(D;\Theta^L)$ is concave, though not strictly concave.*

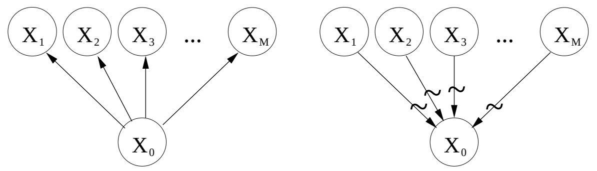


Figure 1: Standard Naive Bayes structure (left) and the corresponding L-model (right). The arcs of the network have been reversed and the resulting product distribution has been replaced by softmax (denoted by tildes).

Proof. The first part is obvious since each parameter can take values in $(-\infty, \infty)$. Concavity of $S^L(D; \Theta^L)$ is a direct consequence of the fact that $\mathcal{M}^L$ is a logistic regression model; see, e.g., (Santner and Duffy, 1989, p. 234). For an example where the supervised log-likelihood is not strictly concave, see (Wettig et al., 2002). ∎

From the theorem we directly obtain the following corollary.

Corollary 1. *There are no local (non-global) maxima in the likelihood surface of an L-model.*

The conditions under which a global maximum exists are discussed in, e.g., (Santner and Duffy, 1989) and references therein. A possible solution in cases where no maximum exists is to assign a prior on the model parameters and maximize the 'supervised posterior' (Grünwald et al., 2002; Wettig et al., 2002) instead of the likelihood.

The global supervised maximum likelihood parameters obtained from training data can be used for prediction of future data. In addition, as discussed in (Heckerman and Meek, 1997a) they can be used to perform model selection among several competing model structures using, e.g., the BIC (Schwarz, 1978) or (approximate) MDL (Rissanen, 1978) criteria. In (Heckerman and Meek, 1997a) it is stated that for general supervised Bayesian network models $\mathcal{M}^{\mathcal{B}}$, "although it may be difficult to determine a global maximum, gradient-based methods [...] can be used to locate local maxima". What is more, Theorem 2 shows that when dealing with L-models even a *global* maximum can be found if it exists.

In the original parametrization, the log-likelihood surface is not necessarily *concave*, as the following example shows.

Example 1. Consider a Bayesian network where the class variable X_0 has only one child, X_1, and both variables take values in $\{1, 2\}$. Let the training data be given by

$$D = \big((1,1), (1,2), (2,1), (2,2)\big).$$

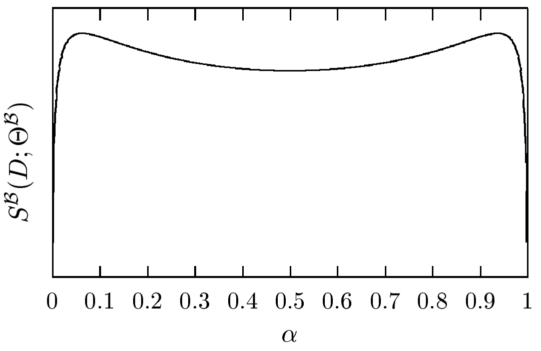


Figure 2: The supervised log-likelihood of Example 1 peaks twice in the original parametrization along a line defined by $\alpha \in (0,1)$.

Set the parameters $\Theta^{\mathcal{B}}$ as follows:

$$\theta^{\mathcal{B}}_{x_0} = \begin{cases} 0.1 & \text{if} \quad x_0 = 1, \\ 0.9 & \text{if} \quad x_0 = 2, \end{cases}$$

$$\theta^{\mathcal{B}}_{x_1|x_0} = \begin{cases} 0.5 & \text{if} \quad x_0 = 1, x_1 = 1, \\ 0.5 & \text{if} \quad x_0 = 1, x_1 = 2, \\ \alpha & \text{if} \quad x_0 = 2, x_1 = 1, \\ 1-\alpha & \text{if} \quad x_0 = 2, x_1 = 2. \end{cases}$$

Figure 2 shows the supervised log-likelihood given data D as a function of α. The figure shows a bimodal curve that clearly violates concavity. ◇

If the network structure $\mathcal{B}$ is such that the two models are equivalent, $\mathcal{M}^{\mathcal{B}} = \mathcal{M}^L$, we can find the global maximum of the supervised likelihood by reparametrizing $\mathcal{M}^{\mathcal{B}}$ in the L-parameterization, and using some local optimization method. Because the log-transformation is continuous, it follows (with some calculus) that in this case all local maxima of the supervised likelihood are global maxima also in the original parametrization $\boldsymbol{\Theta}^{\mathcal{B}}$.

4 Main Result

Theorems 1 and 2 suggest that in order to find parameters maximizing the supervised likelihood for any Bayesian network model, we could use the L-model where optimization is easy. However, the resulting parameters may violate the 'sum-up-to-one constraint', i.e., we may have $\sum_{x_i=1}^{n_i} \exp \theta^L_{x_i|pa_i} \neq 1$ for some $i \in \{0, \ldots, M'\}$ and pa_i. This *could* correspond to

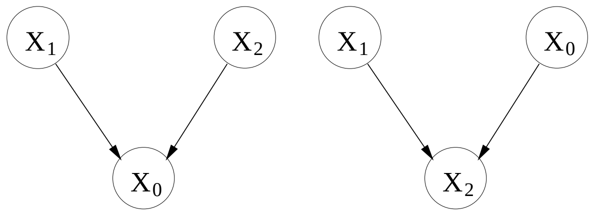


Figure 3: A simple Bayesian network (the class variable is denoted by X_0) satisfying Condition 1 (left); and a network that does not satisfy the condition (right).

a violation of the independence assumptions of the model structure $\mathcal{B}$ as demonstrated by Example 2 below. However, for some structures $\mathcal{B}$, we can show that even if Θ^L is such that $\sum_{x_i=1}^{n_i} \exp \theta^L_{x_i|pa_i} \neq 1$ for some $i \in \{0, \ldots, M'\}$ and pa_i, the conditional distribution indexed by Θ^L is still in $\mathcal{M}^{\mathcal{B}}$.

Our main result is that the independence assumptions encoded by $\mathcal{B}$ are guaranteed to be preserved in the L-model if $\mathcal{B}$ satisfies the following condition:

Condition 1 For all $j = 1..M$, there exists $X_i \in Pa_j \cap \{X_0, \ldots, X_M\}$ such that $Pa_j \subseteq Pa_i \cup \{X_i\}$.

Remark. Condition 1 holds for $\mathcal{B}$ (as can be seen by induction) if and only if any parent set Pa_j of a child X_j of the class X_0 is 'conditionally fully connected', i.e., fully connected modulo arcs (between parents of X_0) that have no effect on the conditional $P(X_0 \mid Pa_j \setminus \{X_0\})$. A necessary but not sufficient condition is that the class X_0 must be a 'moral node', i.e., it cannot have a common child with a node it is not directly connected with; see Figure 4. $\diamond$

Example 2. Consider the Bayesian networks depicted in Figure 3. The leftmost network, $\mathcal{B}_1$, satisfies Condition 1, unlike the rightmost network, $\mathcal{B}_2$. Let $\mathcal{M}_2^{\mathcal{B}}$ denote the 'usual' model corresponding to $\mathcal{B}_2$ indexed by parameters in $\mathbf{\Theta}^{\mathcal{B}}$, and let $\mathcal{M}_2^L$ denote the corresponding L-model. The parameters used to define $\mathcal{M}_2^L$ are $\theta^L_{x_0}$ and $\theta^L_{x_2|x_0,x_1}$ where each x_i varies in $\{1, \ldots, n_i\}$. Consider the distributions in $\mathcal{M}_2^L$ indexed by a Θ^L such that, for some x_0 and x_1, $\sum_{x_2=1}^{n_2} \exp \theta^L_{x_2|x_0,x_1} \neq 1$. Some of these distributions violate the independence assumptions of $\mathcal{B}_2$, and therefore, are not in $\mathcal{M}_2^{\mathcal{B}}$. For instance, suppose that for $x_0 = x_1 = 1$, summing over the values of X_2 gives something *more* than one, whereas for $x_0 = 2, x_1 = 1$, summing over the values of X_2 gives something *less* than one. This would correspond to the situation where given $X_1 = 1$ it is more probable to have $X_0 = 1$ than $X_0 = 2$, i.e., X_1 and X_0 would be dependent *without* X_2 being given. It follows that in this case $\mathcal{M}_2^L$ contains some conditional distributions that do not satisfy the independence assumptions encoded by $\mathcal{B}_2$. This can not happen in the leftmost network $\mathcal{B}_1$ since the structure allows all conditional distributions of X_0 given X_1 and X_2. $\diamond$

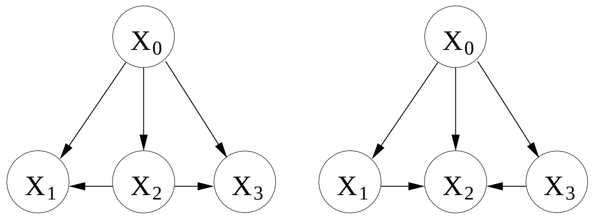


Figure 4: A tree-augmented Naive Bayes (TAN) model satisfying Condition 1 (left); and a network that is not TAN (right). Note that even though in both cases the class variable X_0 is a moral node, the network on the right does not satisfy Condition 1.

As examples of network structures that satisfy Condition 1, we mention the Naive Bayes (NB) and the tree-augmented Naive Bayes (TAN) models (Friedman et al., 1997). The latter is a generalization of the former in which the children of the class variable are allowed to form tree-structures; see Figure 4.

Proposition 1. *Condition 1 is satisfied by the Naive Bayes and the tree-augmented Naive Bayes structures.*

Proof. For Naive Bayes, we have $Pa_j \subseteq \{X_0\}$ for all $j \in \{1, \ldots, M\}$. For TAN models, all children of the class variable have either one or two parents. For children with only one parent (the class variable), we can use the same argument as in the NB case. For any child X_j with two parents, let X_i be the parent that is not the class variable. Because X_i is also a child of the class variable, we have $Pa_j \subseteq Pa_i \cup \{X_i\}$. ∎

Condition 1 is also automatically satisfied if

X_0 only has incoming arcs[1] ('diagnostic' classifiers, see (Kontkanen et al., 2001)). For Bayesian network structures for which the condition does not hold, we can always add some arrows to arrive at a structure $\mathcal{B}'$ for which the condition does hold (for instance, add an arrow from X_1 to X_3 in the rightmost network in Figure 4). Therefore, the model $\mathcal{M}^{\mathcal{B}}$ is always a submodel of a larger model $\mathcal{M}^{\mathcal{B}'}$ for which the condition holds. For these reasons, we regard Condition 1 as relatively mild.

We are now ready to present our main result:

Theorem 3. *If $\mathcal{B}$ satisfies Condition 1, then $\mathcal{M}^{\mathcal{B}} = \mathcal{M}^L$.*

Together with Corollary 1, this shows that if Condition 1 holds for $\mathcal{B}$, then the supervised likelihood surface of $\mathcal{M}^{\mathcal{B}}$ has no local (non-global) maxima. Proposition 1 now implies that, for example, the supervised likelihood surface for the TAN classifiers has no local (non-global) maxima. Therefore, this maximum can be found by local optimization techniques.

Proof. In the following, we will often speak of the parent configuration pa_0 of X_0. In case X_0 has no parents (i.e., $M = M'$), Pa_0 is the empty set and $pa_0(x)$ is constant with respect to $x = (x_0, \ldots, x_{M'})$.

We introduce some more notation. For $j \in \{1, \ldots, M\}$, let m_j be the maximum number in $\{0, \ldots, M\}$ such that $X_{m_j} \in Pa_j$, $Pa_j \subseteq Pa_{m_j} \cup \{X_{m_j}\}$. Such an m_j exists by Condition 1. Condition 1 implies that pa_j is completely determined by the pair (x_{m_j}, pa_{m_j}). We can therefore introduce functions Q_j mapping (x_{m_j}, pa_{m_j}) to the corresponding pa_j. Hence, for all $x = (x_0, \ldots, x_{M'})$ and $j \in \{1, \ldots, M\}$ we have

$$pa_j = Q_j(x_{m_j}, pa_{m_j}). \tag{7}$$

We introduce, for all $i \in \{0, \ldots, M\}$ and for each configuration pa_i of Pa_i, a constant $c_{i|pa_i}$ and define, for any $\Theta^L \in \mathbf{\Theta^L}$,

$$\theta^{(c)}_{x_i|pa_i} := \theta^L_{x_i|pa_i} + c_{i|pa_i} - \sum_{j:m_j=i} c_{j|Q_j(x_i,pa_i)}. \tag{8}$$

[1]It is easy to see that in that case the maximum supervised likelihood parameters may even be determined analytically.

The parameters $\theta^{(c)}_{x_i|pa_i}$ constructed this way are combined to a vector $\Theta^{(c)}$ which is clearly a member of $\mathbf{\Theta^L}$.

After introducing this additional notation, we proceed to the first stage of the proof.

Stage 1 In this stage of the proof, we show that no matter how we choose the constants $c_{i|pa_i}$, for all Θ^L and corresponding $\Theta^{(c)}$ we have $S^L(D; \Theta^{(c)}) = S^L(D; \Theta^L)$.

We first show that, for all possible vectors x and the corresponding parent configurations, no matter how the $c_{i|pa_i}$ are chosen, it holds that

$$\sum_{i=0}^{M} \theta^{(c)}_{x_i|pa_i} = \sum_{i=0}^{M} \theta^L_{x_i|pa_i} + c_{0|pa_0}. \tag{9}$$

To derive (9) we substitute all terms of $\sum_{i=0}^{M} \theta^{(c)}_{x_i|pa_i}$ by their definition (8). Clearly, for all $j \in \{1, \ldots, M\}$, there is exactly one term of the form $c_{j|pa_j}$ that appears in the sum with a positive sign. Since for each $j \in \{1, \ldots, M\}$ there exists exactly one $i \in \{0, \ldots, M\}$ with $p_j = i$, it must be the case that for all $j \in \{1, \ldots, M\}$, a term of the form $c_{j|Q_j(x_i,pa_i)}$ appears exactly once in the sum with a negative sign. By (7) we have $c_{j|Q_j(x_i,pa_i)} = c_{j|pa_j}$. Therefore all terms $c_{j|pa_j}$ that appear once with a positive sign also appear once with a negative sign. It follows that, except for $c_{0|pa_0}$, all terms $c_{j|pa_j}$ cancel. This establishes (9). By plugging in (9) into (6), it follows that $S^L(D; \Theta^{(c)}) = S^L(D; \Theta^L)$ for all D. This concludes Stage 1 of the proof.

Stage 2 Set, for all x_i and pa_i,

$$\theta^{\mathcal{B}}_{x_i|pa_i} = \exp \theta^{(c)}_{x_i|pa_i}. \tag{10}$$

In this stage we show that we can determine the constants $c_{i|pa_i}$ such that for all $i \in \{0, \ldots, M\}$ and pa_i, the 'sum up to one' constraint is satisfied, i.e., we have

$$\sum_{x_i=1}^{n_i} \theta^{\mathcal{B}}_{x_i|pa_i} = 1. \tag{11}$$

We will achieve this by sequentially determining values for $c_{i|pa_i}$ in a particular order. We now need some terminology: we say 'c_i *is determined*' if for all configurations pa_i of Pa_i, we have already determined $c_{i|pa_i}$. We say 'c_i *is undetermined'* if we have determined $c_{i|pa_i}$ for *no* configuration pa_i of Pa_i. We say 'c_i is *ready to be determined'* if c_i is undetermined and at the same time all c_j with $p_j = i$ have been determined.

We first note that as long as some c_i with $i \in \{0, \dots, M\}$ are undetermined, there must exist $c_{i'}$ that are ready to be determined. To see this, first take any $i \in \{0, \dots, M\}$ with c_i undetermined. Either c_i itself is ready to be determined (in which case we are done), or there exists $j \in \{1, \dots M\}$ with $p_j = i$ (and hence $X_i \in Pa_j$) such that c_j is undetermined. If c_j is ready to be determined, we are done. Otherwise, there must exist some k with $X_j \in Pa_k$ such that c_k is undetermined. We can now repeat the argument, and move forward in the Bayesian network structure $\mathcal{B}$ restricted to $\{X_0, \dots, X_M\}$ until we find a c_l that is ready to be determined. Because $\mathcal{B}$ is acyclic, we must find such a c_l (within $M + 1$ steps).

We now describe an algorithm that sequentially assigns values to $c_{i|pa_i}$ such that (11) will be satisfied. We start with all c_i undetermined and repeat the following steps:

```
WHILE there exists $i \in \{0, \dots, M\}$ such that $c_i$ is undetermined
DO
    1. Pick the largest $i$ such that $c_i$ is ready to be determined.
    2. Set, for all configurations $pa_i$ of $Pa_i$, $c_{i|pa_i}$ such that $\sum_{x_i=1}^{n_i} \theta^{\mathcal{B}}_{x_i|pa_i} = 1$ holds.
DONE
```

The algorithm will loop $M + 1$ times and then halt. Step 2 does not affect the values of $c_{j|pa_j}$ for any j, pa_j such that $c_{j|pa_j}$ has already been determined. Therefore, after the algorithm halts, (11) holds. This concludes Stage 2 of the proof.

Let $\Theta^L \in \mathbf{\Theta^L}$. Each choice of constants $c_{i|pa_i}$ determines a corresponding vector $\Theta^{(c)}$ with components given by (8). This in turn determines a corresponding vector $\Theta^{\mathcal{B}}$ with components given by (10). In Stage 2 we showed that we can take the $c_{i|pa_i}$ such that (11) holds. This is the choice of $c_{i|pa_i}$ which we adopt. With this particular choice, $\Theta^{\mathcal{B}}$ indexes a distribution in $\mathcal{M}^{\mathcal{B}}$. By applying the log-transformation to the components of $\Theta^{\mathcal{B}}$ we find that for any D of any length, $S^{\mathcal{B}}(D; \Theta^{\mathcal{B}}) = S^L(D; \Theta^{(c)})$, where $S^{\mathcal{B}}(D; \Theta^{\mathcal{B}})$ denotes the supervised log-likelihood of $\Theta^{\mathcal{B}}$ as given by summing the logarithm of (3). The result of Stage 1 now implies that $\Theta^{\mathcal{B}}$ indexes the same conditional distribution as Θ^L. Since $\Theta^L \in \mathbf{\Theta^L}$ was chosen arbitrarily, this shows that $\mathcal{M}^L \subseteq \mathcal{M}^{\mathcal{B}}$. Together with Theorem 1 this concludes the proof. ∎

5 Concluding Remarks

We showed that by using the parameter transformation described above, one can effectively find the parameters maximizing the supervised (conditional) likelihood of NB, TAN and many other Bayesian network models. For an arbitrary Bayesian network, this transformation may yield a slightly more powerful model class, i.e., remove some of the independence assumptions of the network structure. We also gave a condition under which the transformation does not change the class of models considered. Test runs for the Naive Bayes case in (Wettig et al., 2002) have shown that maximizing the supervised likelihood in contrast to the usual practice of maximizing the unsupervised (joint) likelihood is feasible and yields greatly improved classification. In the future we intend to study more complicated models as well as use the L-parametrization for model selection.

Acknowledgments. This research has been supported by the National Technology Agency, and the Academy of Finland. The authors thank Wray Buntine for many useful comments.

References

C.M. Bishop. 1995. *Neural Networks for Pattern Recognition.* Oxford University Press.

N. Friedman, D. Geiger, and M. Goldszmidt. 1997. Bayesian network classifiers. *Machine Learning*, 29:131–163.

R. Greiner and W. Zhou. 2002. Structural extension to logistic regression: Discriminant parameter learning of belief net classifiers. In *Proceedings of the 18th Annual National Conference on Artificial Intelligence (AAAI-02)*, pages 167–173, Edmonton.

R. Greiner, A. Grove, and D. Schuurmans. 1997. Learning Bayesian nets that perform well. In *Pro-*

ceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97), pages 198–207. Morgan Kaufmann Publishers, San Francisco, CA.

P. Grünwald, P. Kontkanen, P. Myllymäki, T. Roos, H. Tirri, and H. Wettig. 2002. Supervised posterior distributions. Presented at the Seventh Valencia International Meeting on Bayesian Statistics, Tenerife, Spain.

D. Heckerman and C. Meek. 1997a. Embedded bayesian network classifiers. Technical Report MSR-TR-97-06, Microsoft Research.

D. Heckerman and C. Meek. 1997b. Models and selection criteria for regression and classification. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 223–228. Morgan Kaufmann Publishers, San Francisco, CA.

P. Kontkanen, P. Myllymäki, and H. Tirri. 2001. Classifier learning with supervised marginal likelihood. In *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI-01)*, pages 277-284. Morgan Kaufmann Publishers, San Francisco, CA.

A.Y. Ng and M.I. Jordan. 2001. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. *Advances in Neural Information Processing Systems*, 14:605–610.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, San Mateo, CA.

J. Rissanen. 1978. Modeling by shortest data description. *Automatica*, 14:445–471.

T. Santner and D. Duffy. 1989. *The Statistical Analysis of Discrete Data*. Springer Verlag, New York.

G. Schwarz. 1978. Estimating the dimension of a model. *Annals of Statistics*, 6:461–464.

H. Wettig, P. Grünwald, T. Roos, P. Myllymäki, and H. Tirri. 2002. On supervised learning of Bayesian network parameters. Technical Report HIIT-2002–1, Helsinki Institute for Information Technology (HIIT). Available at http://cosco.hiit.fi/Articles/hiit2002-1.ps.

Evolving Causal Neural Networks

Marco A. Wiering
Institute of Information and Computing Sciences
Intelligent Systems Group
Utrecht University
PO Box 80.089, Utrecht
marco@cs.uu.nl

Abstract

We introduce causal neural networks, a generalization of the usual feedforward neural networks which allows input features and target outputs to be represented as input or output units. For inferring the values of target outputs which are represented as input units, we developed a forward-backward propagation algorithm which uses gradient descent to minimize the error of the predicted output features. To deal with the large number of possible structures and feature selection, we use a genetic algorithm. Experiments on a regression problem and 5 classification problems show that the causal neural networks can outperform the usual feedforward architectures for particular problems.

1 Introduction

Bayesian belief networks (Pearl, 1988) are of large interest due to their graphical structure modelling independencies between variables, and the sound Bayesian inference algorithm. Given a set of instantiated values of particular observed features, Bayesian belief networks can be used for inferring the probability distribution of unobserved features. Different neural network algorithms such as Boltzmann machines (Aarts and Korst, 1989) and Sigmoid Belief networks (Neal, 1992) are quite similar to Bayesian belief networks, but exact inference is infeasible in all of these algorithms in case the models become more complex.

In this paper we introduce a much simpler class of neural network architectures which we call causal neural networks, since they also allow for representing causal relations between variables. The structure of the causal neural network allows us to represent target output values in the input layer and we can also represent input feature values in the output layer of a feedforward neural network. The latter has already been studied in (Caruana and de Sa, 1997) and was shown to be effective for particular problems. The causal neural network is a further generalization of the well-known structure of feedforward neural networks. For inferring the value of an output value which is represented as an input unit, we can use backward propagation of error signals of the predicted input feature values represented in the output layer. This backward propagation algorithm is a simple variant of the well known backpropagation algorithm (Rumelhart et al., 1986). The new resulting forward-backward propagation algorithm is used recursively for inference to minimize the error of predicted feature values.

Similarly to Sigmoid Belief networks, causal neural networks try to find those values of target outputs which are useful for generating the desired input feature values. Although there is no direct causal relationship from input feature values represented by output units to target outputs represented as input units, minimizing the reconstruction error causes the system to infer those target outputs which cause the generation of the given input feature values. The causal neural networks can therefore be used to put causes in the input units and the effects of these causes in the output units. This can simplify the learning problem drastically and lead to better generalization.

The learning algorithm for training the causal neural networks is a simple extension of the normal backpropagation algorithm and therefore learning is quite fast. To deal with the large number of possible structures, we employ a genetic algorithm for finding the best structures. In the following, we will use the causal neural networks as a supervised learning algorithm which can deal with missing data and causal relations in a principled way.

Outline of this paper. In section 2 we describe our causal neural networks with the forward-backward algorithm for inference. In section 3 we describe the genetic algorithm for evolving causal neural network structures. In section 4 we describe experimental results. Finally, section 5 concludes this paper.

2 Causal neural networks

The learning task. We study supervised learning on a dataset D consisting of L input vectors X^i and their target output vectors Y^i. The dataset may consist of unknown input features which may make the learning task more difficult. We assume that target outputs in the training data are always known, but could be subject to noise.

In this section, we first describe the architecture, and then the usual forward propagation and backpropagation algorithms for training a causal neural network. To deal with unknown values, we add an additional mean-input for all uninstantiated variables which is used to initialize the values of uninstantiated variables. Finally, we describe the recurrent forward-backward propagation algorithm for inferring the desired output values and unknown input features given the values of instantiated variables.

2.1 The architecture

The causal neural networks could have an arbitrary (modular) structure, but in this paper we only consider fully-connected feedforward neural networks with a single hidden layer. The architecture consist of one input layer with input units[1]: $I_1, \ldots, I_{|I|}$, where $|I|$ is the number of input units, one hidden layer H with hidden units: $H_1, \ldots, H_{|H|}$, and one output layer with output units: $O_1, \ldots, O_{|O|}$. The network has weights: w_{ih} for all input units I_i to hidden units H_h, and weights: w_{ho} for all hidden H_h to output units O_o. Each hidden unit and output unit has a bias b_h or b_o with a constant activation of 1. The hidden units use Sigmoid activation functions, whereas the output units use linear activation functions. Finally, the input units and output units have an additional mean-bias m_i and m_o (different for every unit), which is used to deal with unknown initial values for the uninstantiated variables.

We can represent input features and target outputs inside the input layer or in the output layer. Input features could also not be used at all which allows for feature elimination. The input features which are represented in input (output) units form the set FI (FO), and the target outputs which are represented in the input (output) units form the set TI (TO). Figure 1 shows an example of a causal neural network architecture. The figure also shows the forward and backward propagation algorithms used for inference.

The set FO is usually disjunct from the set FI, since otherwise reconstruction of input feature values in FO would be trivial and not useful for inferring unknown values of target outputs. The input feature values in FO are useful, since the goal is to reconstruct their values from the values of the inputs units. In order to reconstruct these, causal explanations of these values must be found in the (known or unknown) input feature values and the unknown target outputs of the set TI. If the set FI is empty, the algorithm tries to infer the values of target outputs in TI which minimize the reconstruction error.

[1]When we refer to a unit, we also mean its activation.

If FI is not empty, these feature input values serve as a context for this reconstruction.

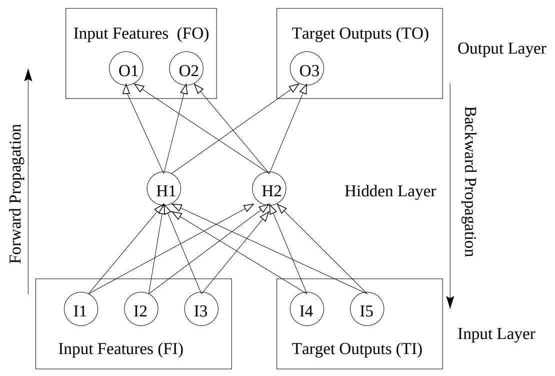


Figure 1: *A causal neural network consisting of one hidden layer. Input features and target outputs can be represented as input or output units.*

Transformation of examples. Given an example (X^i, Y^i) we can represent the input features and target outputs as either input or output units. Depending on the neural network structure which maps input features and target outputs to input and output units, we get a new training example $(\hat{X}^i, \hat{Y}^i)$. To deal with unknown values for variables (e.g. missing data or the target outputs) which are represented as input units, we use the values of their mean-bias m_i learned by the delta rule.

2.2 Forward propagation

Given the values of all input units, we can compute the values for all output units with forward propagation. The forward propagation algorithm for an instance X looks as follows:

1) Clamp the known input feature values $\in FI$ in the input layer: $I_i = X_i$, and clamp the mean-bias for unknown feature values $\in FI$ and for target output values $\in TI$ to the input layer: $I_i = m_i$.

2) Compute the values for all hidden units $H_h \in H$ as follows:

$$H_h = \sigma(\sum_{i \in FI \cup TI} w_{ih} I_i + b_h)$$

Where $\sigma(x)$ is the Sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

3) Compute the values for all output units $O_o \in FO \cup TO$:

$$O_o = \sum_h w_{ho} H_h + b_o$$

2.3 Backpropagation

For training the system we extend the backpropagation algorithm (Rumelhart et al., 1986) in which we also learn the mean-bias values using the delta rule. If feature values in FO are unknown, we use their mean-bias as a target output value. Although leaving them completely out in the learning algorithm would be another possibility, preliminary experimental results indicate that our approach can be more efficient. The learning goal is to learn a mapping from the transformed instance $\hat{X}$ to $\hat{Y}$. For this we first use forward propagation to compute the outputs values O_o and then we use backpropagation to minimize the squared error measure:

$$E = \frac{1}{2} \sum_{o \in FO \cup TO} (\hat{Y}_o - O_o)^2$$

To minimize this error function, we update the weights and biases in the network using gradient descent steps with learning rate α:

$$\Delta w_{ho} = -\alpha \frac{\partial E}{\partial w_{ho}} = \alpha(\hat{Y}_o - O_o)H_h$$

and

$$\Delta w_{ih} = \alpha H_h(1 - H_h)I_i \sum_{o \in FO \cup TO} (\hat{Y}_o - O_o)w_{ho}$$

Update the mean-bias values for input and output units using the delta rule:

$$m_i = m_i + \alpha(\hat{X}_i - m_i); \quad \text{and} \quad m_o = m_o + \alpha(\hat{Y}_o - m_o)$$

2.4 Forward-backward propagation

For inferring the values of all unknown variables given the instantiated values of known input features we introduce the forward-backward propagation algorithm:

1) Clamp the input units to their input feature values. Unknown values for input features and target outputs $\in TI$ will initially have their mean-bias values as input value, after which their activations will be continually updated.

2) Clamp the target output values in $\hat{Y}$ for all input features $\in FO$ to their value in the instance X. For unknown values of input features we use their mean-bias values as target output values in $\hat{Y}$.

3) Use forward propagation to infer the output values $O_i \in FO \cup TO$.

4) Use backward propagation to update the values of uninstantiated input units by gradient descent on the error over the predicted input features $\in FO$:

$$\Delta I_i = \alpha_b \sum_h H_h(1 - H_h)w_{ih} \sum_{o \in FO} (\hat{Y}_o - O_o)w_{ho}$$

5) Repeat Step (3) until some termination condition holds.

Here, α_b is the backward learning rate. We recursively apply the forward-backward algorithm to infer the values of all uninstantiated variables. The algorithm may not always converge, however, since values of uninstantiated variables in the input layer may become infinite to best predict the output features. In the experiments we iterate the algorithm for 400 times which therefore makes inference much slower than with the usual feedforward neural networks. The learning time is exactly the same, since for learning we just make use of the backpropagation algorithm.

3 Evolving causal neural networks

The additional freedom we gain with our causal neural networks can also give us problems, since how can we decide whether we should represent an input feature by an input or output unit, and should the target output(s) be represented as input or output unit(s)? For optimizing the structure of the causal neural networks, we use genetic algorithms (Holland, 1975) and cross-validation. Thus, we divide the total dataset D in three data-sets: the learning dataset D_l, the cross-validation (halting) dataset D_c, and the test dataset D_t. In our experiments, we use the causal neural networks for regression and classification purposes. For the regression task it is possible that some of the target output values are represented as input units and others as output units. Therefore we can evolve mixed architectures. For the classification task there is only one target output value (the classification of an input pattern), and we use the genetic algorithm only with populations of homogeneous structures in which the target output value is always represented by an output unit or by an input unit (we call the corresponding networks forward networks (FN) or backward networks (BN)). Input features can be used as input units, output units, or not at all. When the target outputs are represented as input units, the structure should contain at least one input feature in the output layer. Furthermore, we evolve the number of hidden units and the learning rate.

The genetic algorithm. We use crossover and mutation operators for evolving the population of structures. We use tournament selection with size 3 and the elitist strategy. Given a structure, we train it n_t times on the learning dataset D_l and use cross-validation after each t_c iterations to look whether we should stop the learning process. The best test-error over these n_t train-test trials is kept as fitness value for the structure. After evolving G generations, we compute the error of the best trained individual during the last generation on the test dataset D_t, and return this as a single experimental result.

4 Experiments

We performed a number of experiments to validate the usefulness of our approach compared to the usual feedforward neural networks structures trained with backpropagation. We use two different experiments. The first is a regression task in which there are 3 input features and 3 target outputs. In the second experiment, we use 5 real-world datasets from the UCI repository (Merz et al., 1997).

4.1 The regression problem

The input consists of three inputs X_1, X_2, X_3 which are drawn from a random uniform distribution between 0 and 0.33. The target outputs are computed as:

$$Y_1 = -log(\frac{10.0}{\sigma(X_1)+\sigma(X_2)+X_3} - 1)$$
$$Y_2 = (X_1^2 + X_2^2) \times -log(\frac{10.0}{\sigma(X_1)+\sigma(X_2)+X_3} - 1)$$
$$Y_3 = (\sqrt{X_1} + \sqrt{X_2}) \times -log(\frac{10.0}{\sigma(X_1)+\sigma(X_2)+X_3} - 1)$$

Here $\sigma(x)$ denotes the Sigmoid function. This is an artificial problem which we constructed to show the advantage of using mixed architectures. We can see that Y_1 is a building block for Y_2 and Y_3, which means that we could put Y_1 in the input layer, together with X_1 and X_2 to infer Y_2, Y_3, and also X_3. We will refer to this architecture as a mixed architecture. Another advantage of this mixed architecture is that inferring X_3 from Y_1, X_1 and X_2 may be easier than inferring Y_1 from X_1, X_2, and X_3, since the function is the same as: $X_3 = 10\sigma(Y_1) - \sigma(X_1) - \sigma(X_2)$. Thus, we do not have to approximate the logarithmic function in this way.

Simulation set-up. We compare the normal feedforward network (FN) with no input features in the output layer to the backward network (BN) with all outputs in the input layer and no other inputs (which is of course not suited for this task), and to the mixed architecture with Y_1 and X_1, X_2 in the input layer, and Y_2, Y_3 and X_3 in the output layer. We also evolve mixed architectures using the genetic algorithm with a population size of 30, 50 generations, and $n_t = 1$. For the non-evolving architectures, the number of hidden units is 5. The learning rate $\alpha = 0.2$ and $\alpha_b = 0.1$. We perform 100 simulations with different distributions for the three datasets consisting of 100 examples, for which 10 networks of each type are trained and tested using cross-validation. The trained network with the smallest cross-validation error is used for testing on the test dataset D_t. For this the root of the mean squared error is computed. The results are shown in table 1.

Experimental results. The figure shows that the mixed architecture performs better than the usual feedforward network. Thus, putting target output values inside the input layer of a feedforward neural network can be useful. As expected, the complete backward network is not suited for this task. The genetic algorithm is shown to be able to almost always find the best possible architecture (the Mix-NN architecture).

Table 1: *The Training results (RMSE) on the 3 input 3 output function. Averages are computed over 100 simulations.*

FN	BN	MIX-NN	GA MIX-NN
0.0095±0.0075	0.105±0.012	0.0068±0.0026	0.0071±0.0034

4.2 Experiments on real datasets

We also performed experiments on real datasets from the UCI repository (Merz et al., 1997). We use the following 5 datasets consisting of three medical diagnosis datasets: Hepatitis (155 examples), Liver disease (345), Pima Indians (768), and two other datasets: Chess Kr-vs-kp (3196), and Vote (435). We also make 10% of the input features in the four datasets unknown to examine how unknown (missing) values affect the performance of the different neural network architectures. For these binary classification problems, we threshold at 0 to compute the output class. The results are given as percentage of wrong classifications.

Simulation set-up. We run 10 simulations with different random partitionings of the total dataset into D_l, D_c, and D_t; all datasets have $\frac{1}{3}$ of the number of examples of the total dataset. We use the genetic algorithm on forward (GA-FN) and backward networks (GA-BN) and let it run for 50 generations using a population size of 30 and $n_t = 3$. For the normal forward and backward architectures which are not evolved, we use 4 hidden units for which 500 networks are trained and tested each on the cross-validation dataset. The best was used for testing on the test dataset D_t.

Table 2: *The Training results on the 5 datasets with and without adding missing (unknown) input feature values.*

Data Set	Mis.	FN	BN	GA-FN	GA-BN
Hepatitis	0%	0.46±0.11	0.37±0.04	0.38±0.07	0.37±0.05
Hepatitis	10%	0.51±0.09	0.36±0.06	0.42±0.08	0.36±0.04
Liver Dis.	0%	0.43±0.04	0.39±0.05	0.44±0.05	0.40±0.04
Liver Dis.	10%	0.43±0.05	0.39±0.06	0.43±0.04	0.41±0.07
Pima Ind.	0%	0.25±0.04	0.24±0.03	0.26±0.05	0.24±0.03
Pima Ind.	10%	0.35±0.07	0.24±0.03	0.30±0.05	0.24±0.03
Chess	0%	0.05±0.01	0.25±0.02	0.07±0.02	0.07±0.01
Chess	10%	0.14±0.01	0.25±0.02	0.17±0.02	0.14±0.01
Vote	0%	0.06±0.02	0.11±0.02	0.07±0.02	0.07±0.01
Vote	10%	0.08±0.02	0.11±0.02	0.07±0.01	0.08±0.02
Average		0.276	0.271	0.261	0.238

Experimental results. The results are given in table 2. For the Hepatitis problem the neural networks do not perform very well; this can be explained by the small number of learning examples (there were only 52 examples used for training). Different learning rates or numbers of hidden units did not improve the results. The forward networks perform better than the backward networks on Vote and Chess, but it is interesting to see that the backward networks perform better on the medical diagnosis datasets in which the disease causes the observed symptoms. The backward models also suffer much less from missing data. Using the genetic algorithm can help to find better structures, which is especially clear for the backward networks which can combine input features with target outputs in the input layer to learn correct mappings for Vote and Chess.

5 Conclusion

We described a new neural network architecture which is a generalization of normal feedforward neural networks. These causal neural networks can represent feature inputs and target outputs in the input or output layer of a feedforward network. Although we only used them for supervised learning tasks in this paper, the causal neural networks can also be used for pattern completion, and extended for unsupervised learning purposes. The training algorithm of the causal neural networks is very fast, since it is a simple extension of the normal backpropagation algorithm, although the question may arise whether this is a valid thing to do if there are missing values. For inference in our causal networks we developed the forward-backward propagation algorithm which should be executed recursively to infer all the values of all uninstantiated variables. We want to look closer at stopping conditions for this propagation and the effects of early stopping on the obtained approximation.

Experiments on a regression task and 5 datasets from the UCI repository show that the causal neural networks with target outputs in the input layer can outperform the usual feedforward neural networks for medical datasets in which target outputs (diseases) cause the input features (symptoms). Furthermore, these backward networks are much less sensitive to missing data. For other problems, architectures found by a genetic algorithm which mix input features and target outputs in the input layer performs quite well compared to the usual feedforward neural networks.

The causal neural networks are similar to Sigmoid Belief networks in some way, since they both try to infer causal output variables given instantiated feature values. The difference is that Sigmoid Belief network use a probabilistic setting and try to maximize the likelihood of generating the instantiated feature values, whereas in causal neural networks the goal is to minimize the squared error of the instantiated feature values represented as ouput units.

It is interesting to analyse the meaning of hidden unit representations in the case of having target outputs represented by input units. In the case of non-linear dimensionality reduction (DeMers and Cottrell, 1993) with a four layer feedforward neural network with a bottleneck, the hidden units in the second layer represent a non-linear manifold for reconstructing the input. This may also be applied for supervised learning in which the input and output consist of feature inputs and target outputs. Reconstructing the output could then be done by propagating the values of known feature inputs. A similar technique was applied in (Hancock and Thorpe, 1994) for training a car to follow roads. Here the authors computed the eigenvectors of the generated camera images and a given corresponding steering command to train the system. Afterwards, the system was applied by linearly combining the eigenvectors for the newly generated camera images which also gives the control commands. It remains a question whether such reconstructions can always be used to generate the correct output.

In causal neural networks, we are less interested in dimensionality reduction, but try to evolve an architecture which can use the input units to infer the values of separate output units. In this case the hidden units repesentations should be mixed to infer the values of output units. The mixture is dependent on the values of input units. Therefore if an target output is represented as input unit, forward-backward propagation derives its value which causes the hidden units to be mixed in the best possible way for reconstructing the given values of output units. It should be clear that this cannot be done with all possible architectures, therefore a suitable architecture needs to be evolved.

In future work, we want to study more general graphical structures to model the dependencies between variables. For learning and inference we can employ variations of the backpropagation and forward-backward propagation algorithms.

References

A.H.L. Aarts and J.H.M. Korst. 1989. *Simulated Annealing and Boltzmann Machines.* Wiley, Chichester.

R. Caruana and V.R. de Sa. 1997. Promoting poor features to supervisors: Some inputs work better as outputs. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9.* Morgan Kaufmann, San Mateo, CA.

D. DeMers and G.W. Cottrell. 1993. Non-linear dimensionality reduction. In S.J. Hanson C.L. Giles and J.D. Cowan, editors, *Advances in Neural Information Processing Systems 5*, pages 580–587. Morgan Kaufmann, San Mateo, CA.

J.A. Hancock and C.E. Thorpe. 1994. ELVIS: Eigenvectors for land vehicle image system. In *CMU-RI-TR*.

J.H. Holland. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

C.J. Merz, P.M. Murphy, and D.W. Aha. 1997. UCI repository of machine learning databases.

R.M. Neal. 1992. Connectionist learning of belief networks. *Artificial Intelligence*, 56:71–113.

J. Pearl. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA.

D.E. Rumelhart, G.E. Hinton, and R.J. Williams. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing*, volume 1, pages 318–362. MIT Press.