Effects of Faulty Knowledge Engineering on Structured Classification Learning *

Joshua Jones
University of Maryland, Baltimore County
Baltimore, MD 21230
jkj@umbc.edu

Ashok Goel
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
goel@cc.gatech.edu

Abstract

Past research has shown that when tree-structured background knowledge is available, it can be exploited to increase the efficiency of classification learning. When this kind of background knowledge is available, the problem becomes one of compositional classification. Of course, if the background knowledge contains errors, the quality of the learned hypothesis will suffer. In this paper we study the effect of faulty knowledge engineering on compositional classification learning. We present and analyze empirical results that show the impact on the quality of compositional classification learning as the quality of knowledge engineering is degraded.

Introduction

In order to enable efficient and practical learning in the context of complex, high-dimensional learning problems, the machine learning community has increasingly embraced the use of various kinds of knowledge-based bias (e.g. (Dietterich 2000) (Marthi, Russell, and Latham 2005) (Ulam, Jones, and Goel 2008) (Mitchell and Thrun 1993) (Towell and Shavlik 1994) (Whiteson et al. 2005) (Pearl 1988)). While the use of prior knowledge to bias learning can be highly effective in increasing the generalization power and efficiency of a learning algorithm, the risk is that errors in this background knowledge will hamper the learning process. Faulty background knowledge is likely to decrease the quality of the final hypothesis selected by the learning algorithm and/or the efficiency of the learning, as improper generalizations may be made and allowable generalizations may be missed. Our goal in the research described here is to develop a better understanding of the degree of risk involved in biasing a learning process with potentially fallible background knowledge.

Here we are specifically concerned with compositional classification (Jones and Goel 2009), where prior knowledge in the form of a tree structure of classification subproblems is given, and where both the top level class label and the output values of the subproblems can be obtained during learning (but are not known to the knowledge engineer, and thus cannot be hard-coded). Past research has shown that when tree-structured background knowledge is available, it can be exploited to increase the efficiency of learning (Tadepalli and Russell 1998). Below we first describe compositional classification more formally. We then explain the framework for compositional classification learning that we have implemented, and with which the experiments described in this paper were performed (Jones and Goel 2008). We call this framework for compositional classification learning *abstraction networks* (ANs) and refer to our implemented system as Augur. We then turn to a discussion of two experiments performed to test the effects of faulty knowledge engineering in this setting.

Compositional Classification

Informally, a compositional classification problem is one in which a hierarchy of intermediate abstractions mediating between raw, observable state (the input feature vector) and the target classification can reasonably be elaborated. These intermediate abstractions are higher level concepts that are more abstract than raw state but less abstract than the target classification. In this work we require that the values of all of these intermediate abstractions be obtainable during learning. The structure of these intermediate abstractions forms the background knowledge (provided by a knowledge engineer) that will be provided to our learner. This type of background knowledge is similar to that provided in other learning techniques, such as Bayesian Networks (Pearl 1988) and Tree-Structured Bias (Russell 1988) (Tadepalli and Russell 1998). Further, research in knowledge-based artificial intelligence has indicated that this type of background knowledge is frequently available in classification (Goel and Bylander 1989) (Bylander, Johnson, and Goel 1991) (Chandrasekaran 1993).

To more formally describe compositional classification, let T be a discrete random variable representing the class label. Let $S = \{s : s \text{ is empirically determinable and } h[T] > h[T|s]\}$, where $h[x]$ denotes the entropy of x. S is a set of discrete random variables that have nonzero mutual information with the class label and are *empirically determinable*, meaning that there is some way to interact with the environment to determine which value has been taken by each member of S, though in general this interaction will not be possible until some time after the classification has been

*Please cite as: Josh Jones & Ashok Goel. Effects of Faulty Knowledge Engineering on Structured Classification Learning. In AAAI-2010 Workshop on Abstraction, Reformulation and Approximation, Atlanta, July 2010, pg 32-37.

produced. This means that the environment acts as an oracle for these intermediate classifications, allowing the learner to obtain their true values during learning. Each member s of S represents a related set of equivalence classes, where each value taken by s is a unique equivalence class. A task instance is generated by jointly sampling the variables in $S \cup T$. *Compositional classification* is the problem of predicting T given some subset of S (the same subset from one problem instance to the next). In order to make such predictions, our learner will make use of a structured knowledge representation called an *abstraction network*, defined in the next section. This representation will capture knowledge about the relationships between variables in S. Learning is required if the distributions $\mathcal{P}(s|K), s \in S \cup T, K \subseteq S$ are not known, but must instead be inferred from experience.

Abstraction Networks

Representation

Here we formally define the knowledge representation used for the compositional classification task. The knowledge structure contains a node for each $s \in S \cup T$. These nodes are connected in a hierarchy reflecting direct dependence relationships organized according to background knowledge. Each node will handle the subproblem of predicting the value of the variable with which it is associated given the values of its children.

Definition 1 *Here, we will define a* supervised classification learner *as a tuple $\langle I, O, F, U \rangle$, where I is a set of input strings (input space), O is a set of output symbols (output space), F is a function from I to O, and U is a function from $(i, o) : i \in I, o \in O$ to the set of supervised classification learners that share the same input space I and output space O. U is an update function that has the effect of changing F based upon a training example. This update function alters the hypothesis represented by the learner, implementing an incremental learning procedure that will progressively change the learner's hypothesis as examples are presented.*

Definition 2 *An* empirical verification procedure *(EVP) is a pair $\langle E, O \rangle$ where O is a set of output symbols (output space) and E is an arbitrary, possibly branching sequence of actions in the environment and observations from the environment concluding with the selection of an $o \in O$.*

EVPs define interactions with the environment that are used to obtain the true values of intermediate abstractions during learning. Any output space O of an empirical verification procedure is an empirically determinable set of equivalence classes. So, a set of equivalence classes is empirically determinable, as required of members of S in the definition of compositional classification, if an empirical verification procedure can be defined with an output space equal to that set of classes.

Definition 3 *An* abstraction network *is recursively defined as follows. A tuple $\langle \emptyset, O, L, P, last\_input, last\_value \rangle$ is an Abstraction Network, where O is a set of output symbols, L is a supervised classification learner, and P is an Empirical Verification Procedure. $last\_input$ and $last\_value$ are used to cache input and return values at AN nodes in order to support the learning procedure (detailed below). A tuple $\langle N, O, L, P, last\_input, last\_value \rangle$ is an abstraction network, where N is a set of abstraction networks. Let I be the set of strings formable by imposing a fixed order on the members of N and choosing exactly one output symbol from each $n \in N$ according to this order. The supervised classification learner L has input space I and output space O, and the Empirical Verification Procedure P has output space O.*

When N is empty, L is trivial and has no use as the input space is empty. In these cases (the leaves of the AN), a value determination must always be made by invoking P, which must be possible before classification in the case of AN leaves. Thus, the leaves of the AN form the subset of S that will be provided as raw state inputs during classification.

Reasoning

In a given task instance, the values of the leaf nodes are fixed by observation. Each node with fixed inputs then produces its prediction. This is repeated until the value of the class label is predicted by the root of the hierarchy. This procedure is described in detail in Table 1.

Learning

At some time after classification, the true value of the class label is obtained by the learner. If the value produced by the hierarchy was correct, no further action is taken. Otherwise, a diagnosis and learning procedure is followed. The specifics of this procedure are dependent upon the characteristics of the learner types that are used within nodes and the specific problem setting. For the empirical results detailed in this paper, the following procedure is used:

1. The true value of each child of the AN root is obtained by executing the associated EVPs.
2. If the predictions of all children were correct, modify local knowledge at the current node by invoking the local supervised learner.
3. Otherwise, recursively repeat this procedure for each child sub-AN that was found to have produced an incorrect prediction.

This procedure is described more formally in Table 2. The procedure has a base case when the leaves are reached, as their true values were obtained before classification, and thus cannot be found to be incorrect. The procedure is optimized to localize blame for classification errors using as few probes as possible under certain assumptions about error (no compensating faults) and the problem setting/learner type.

Experiments

We have performed two sets of experiments in a synthetic domain intended to provide insight into the performance of compositional classification learners when knowledge engineering is imperfect. The environment in this domain consists of a fixed abstraction network, over which no learning will occur, that represents the correct, target content (and

Table 1: Reasoning procedure used to produce a predictive classification from an abstraction network a.

```
/* Values from Definition 3:
 * a.N              - a set of ANs.  The children of 'a'.
 * a.P              - the EVP for 'a'.
 * a.last_input     - the last input sequence provided to 'a'.
 * a.last_value     - the last value produced by 'a'.
 * a.L              - the learner associated with 'a'.
 *
 * Values from Definition 1:
 * L.F              - the learner's inference function.
 *
 * Subfunctions used:
 *   push_back(Vector i, Value V):
 *                  Appends the value provided as the second argument
 *                  to the vector provided as the first.
 */
begin AN-reasoning(abstraction network a)
      Vector i ← {}

      /* If we are at a leaf, return the result of executing the local
       * EVP. These values are the ``inputs" to the AN inference process.*/
      if a.N = ∅, return a.P

      /* Otherwise, build the input vector for the local learner
       * and return the result of applying it.  */
      forall n ∈ a.N:
                push_back(i,AN-reasoning(n))
      a.last_input ← i
      a.last_value ← a.L.F(i)
      return a.last_value
end
```

Table 2: Diagnosis and repair procedure used to correct knowledge stored in an abstraction network a.

```
/* Values from Definition 3:
 * a.P              - the EVP for 'a'.
 * a.last_value     - the last value produced by 'a'.
 * a.N              - a set of ANs.  The children of 'a'.
 * a.L              - the learner associated with 'a'.
 * a.last_input     - the last input sequence provided to 'a'.
 *
 * Values from Definition 1:
 * L.U              - the learner's update (learning) function.
*/
begin AN-learning(abstraction network a)
      Bool flag ← true
      if a.P() = a.last_value, return true
      forall n ∈ a.N
                if AN-learning(n) = false, flag ← false
      if !flag, return false
      a.L ← a.L.U((a.last_input, a.P()))
      return false.
end
```

structure) for the problem. Given this fixed AN, we then create a separate *learner* AN that will be initialized with incorrect knowledge content (and in some cases, incorrect structure) and expected to learn to produce the same top-level classifications as the fixed AN. This is implemented by initializing the knowledge content of both the fixed and learner AN nodes separately with pseudo-random values. The randomly-generated content of the fixed AN forms the target knowledge for the learner AN. Training proceeds by repeating the following steps:

1. Generating a pseudo-random sequence of floating point numbers to serve as the observations for the input nodes of the ANs.
2. Performing inference with the fixed AN, saving the values produced at each node.
3. Performing inference with the learner AN.
4. Performing EVP-based diagnosis and learning over the learner AN as in the procedure described in Table 2.

In this synthetic domain, EVPs within the inputs of both (fixed and learner) ANs are set up to quantize the floating point observations. EVPs are not needed at non-leaf nodes in the fixed AN, since no learning will occur. EVPs at non-leaf nodes in the learning AN are set up to examine the saved output value from the corresponding node in the fixed AN.

In all of these experiments, a binary AN hierarchy was used, with level sizes 16-8-4-2-1. We allowed each node in the hierarchy to produce 4 output values. Each non-leaf node in the learner AN contained a kNN learner with a k-value of 1. The results shown in this section are an average of 20 randomized trials, each consisting of sequences of randomly selected examples split into blocks of 100 for graphing purposes. For the purposes of comparison, we also run a baseline consisting of an unstructured kNN learner working on the same classification problem – the unstructured learner receives the 16 quantized input values and learns to produce the same target values as the AN learner. In the first of these experiments, specific nodes are ablated from within the learner AN, connecting the child nodes of the removed node to the parent node of the removed node. In these experiments, no input information is lost through the node removals (inputs are never ablated), but we expect the hypothesis space restriction imposed by the AN structure to be diminished, and thus the efficiency of learning to decrease. This expectation is indeed borne out by the experiments, summarized in Figure 1. In these experiments, we still reach or approach zero error, as expected because, when learners capable of expressing any function (such as kNN learners) are used within nodes, the correct hypothesis is never eliminated from those expressible by the AN through this kind of ablation. However, the learning rate is negatively impacted as the restriction bias imposed by the AN is reduced. The keys for the graphs in this section refer to the location of nodes ablated by *level*. We consider leaf nodes to be level 0, the direct parents of leaf nodes to be level 1, etc. This notation is possible because of the balanced binary structure used in these experiments. An interesting note about these results is that, when ablating a single node, it appears to make no significant difference at which level of the hierarchy the node is removed. This suggests that impact on overall hypothesis space size is not dependent upon a concept's level of abstraction.

In the second set of experiments, whole subtrees beneath a selected node (or nodes) are pruned from the learner AN. This kind of ablation actually has the effect of *increasing* the restriction bias of the AN, as all hypotheses dependent upon the inputs beneath the ablated node are no longer express-

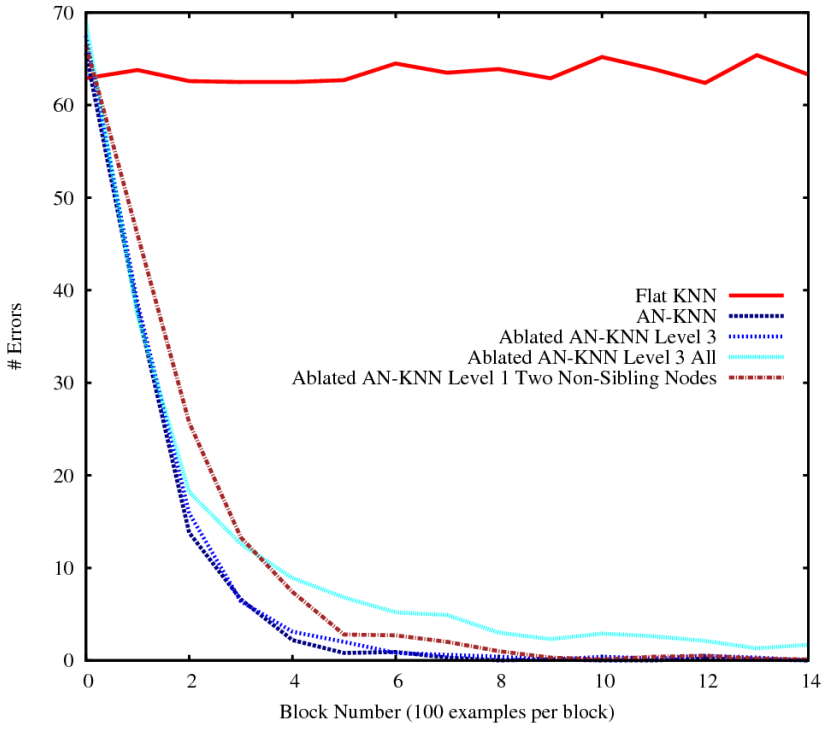

Figure 1: Results of ablating (groups of) individual nodes from an AN learner.

ible at all. The problem, of course, is that the restriction bias is likely to have now excluded the correct hypothesis, as inputs that may be needed for discrimination between two states could have been removed. These induced deficiencies are much more severe than those of the first set of experiments. As expected, the ability of the learner to correctly match the target function are more severely hampered, as illustrated in Figure 2. However, the final error reached is still below that of an unstructured kNN learner after 1000 training examples – illustrating that, if *any* reliable structural information is available about a domain, there is substantial benefit to its exploitation if few training examples are available. Of course, over time the unstructured kNN learner would reach zero error in this synthetic domain, once it has seen and memorized by rote each problem instance. However, this would require a massive training set. If it is known that some inputs are or may be pertinent, one can always feed them directly into the root node of an AN hierarchy even if intervening structure is not known. But it is interesting to note that in some sense, a designer is better off knowing about only a subset of the inputs relevant to a classification problem and having some good knowledge about an intervening abstraction structure than having full knowledge of the relevant inputs but no knowledge of the structure. While the latter scenario allows the designer to produce a learner that *theoretically* can express the correct hypothesis and thus would eventually reach zero error, in practical terms for large problems it will not be possible to gather enough training examples to get there. On the other hand while in the former scenario zero error will never be reached, some level of useful generalization can be made after relatively few input examples. In the trial where we ablated two non-sibling level 2 nodes, we have literally removed half of the problem inputs and still get a better error rate after 1000 examples have been seen!

Conclusions

We have presented a set of experiments in a synthetic domain that explore the impact of faulty knowledge engineering on compositional classification learners. The key finding in these experiments is that as knowledge engineering quality degrades, there is a corresponding gradual degradation in the benefit obtained from using the structural background knowledge. Of course, here we have tested only two kinds of incorrectness in knowledge engineering (missing inputs and missing intermediate abstractions). One could imagine many other kinds of errors, such as wiring nodes into the wrong location in an AN. In this case, one would expect the AN to learn to ignore information that is not pertinent to a particular classification. This would slow learning but should not impact final error beyond the effect of not having the information available in the correct location. Thus, the effect of such an error could be expected to be similar to that of ablating the subtree beneath the miswired node. In any case, it is not the intent of this paper to experiment with, or even identify an exhaustive taxonomy of conceivable errors in knowledge engineering. However, the experiments described here do provide some sense of the kinds of degradation in learning rate (when intermediate abstractions are

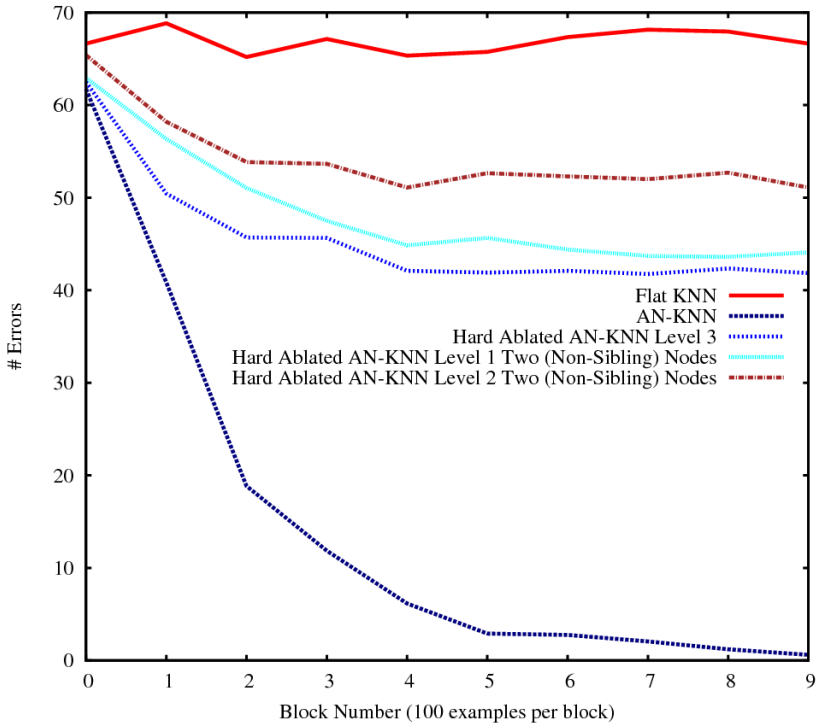

Figure 2: Results of ablating (groups of) subtrees from an AN learner.

missed but all needed inputs are intact) and final error levels (when needed inputs are not present) that one can expect under two kinds of faulty knowledge engineering that seem likely to occur in practice when designing classification hierarchies. Since compositional classification learning still performs better than flat classification even with these knowledge engineering errors, the takeaway finding is that structural background knowledge is well worth using in compositional classification settings even when it is prone to error.

References

Bylander, T.; Johnson, T.; and Goel, A. 1991. Structured matching: a task-specific technique for making decisions. *Knowledge Acquisition* 3:1–20.

Chandrasekaran, B. 1993. Generic tasks in knowledge-based reasoning: high-level building blocks for expert system design. 170–177.

Dietterich, T. 2000. An overview of MAXQ hierarchical reinforcement learning. *Lecture Notes in Computer Science* 1864.

Goel, A., and Bylander, T. 1989. Computational feasibility of structured matching. *IEEE Trans. Pattern Anal. Mach. Intell.* 11(12):1312–1316.

Jones, J., and Goel, A. K. 2008. Retrospective self-adaptation of an agents domain knowledge: Perceptually-grounded semantics for structural credit assignment. In *In Proceedings of the AAAI-08 Workshop on Metareasoning*.

Jones, J., and Goel, A. 2009. Metareasoning for adaptation of classification knowledge. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, 1145–1146. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Marthi, B.; Russell, S.; and Latham, D. 2005. Writing Stratagus-playing agents in concurrent ALisp. In *Proceedings of the IJCAI Workshop on Reasoning, Representation and Learning in Computer Games, Edinburgh, UK*.

Mitchell, T. M., and Thrun, S. B. 1993. Explanation-based neural network learning for robot control. In Giles, C. L.; Hanson, S. J.; and Cowan, J. D., eds., *Advances in Neural Information Processing Systems 5, Proceedings of the IEEE Conference in Denver*. San Mateo, CA: Morgan Kaufmann.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Russell, S. J. 1988. Tree-structured bias. In *AAAI*, 641–645.

Tadepalli, P., and Russell, S. J. 1998. Learning from examples and membership queries with structured determinations. In *Machine Learning*, volume 32, 245–295.

Towell, G. G., and Shavlik, J. W. 1994. Knowledge-based artificial neural networks. *Artificial Intelligence* 70(1-2):119–165.

Ulam, P.; Jones, J.; and Goel, A. K. 2008. Using model-based reflection to guide reinforcement learning. In *Fourth*

AAAI Conference on AI in Interactive Digital Entertainment (AIIDE-08).

Whiteson, S.; Kohl, N.; Miikkulainen, R.; and Stone, P. 2005. Evolving keepaway soccer players through task decomposition. *Machine Learning* 59(1):5–30.