An Agent-to-Agent Framework for Concept Transfer

Sandip Sen and Parijat Kar (sandip,karpa@ens.utulsa.edu)
Mathematical & Computer Sciences Department
University of Tulsa Tulsa, USA

Abstract

Though various interesting research problems have been studied in the context of learning agents, few researchers have addressed the problems of one, knowledgeable, agent teaching another agent. Agents can do more than share training data, problem traces, learned policies. In particular, we investigate how an agent can use its learned knowledge to train another agent with a possibly different internal knowledge representation. We have developed an algorithm that can be used by a concept learning agent, the trainer, to iteratively select training examples for another agent, the trainee, without any assumptions about its internal concept representation or learning algorithm. We present results on artificial and real-life concept learning problems using our Agent Teaching Agent (ATA) framework where instance based and decision tree learners are used as trainer and trainee agents.

Introduction

In this paper, we address the problem of transfer of knowledge between a trainer and a trainee agent. The knowledge being transferred is a concept description. A concept description is a boolean-valued function that classifies input examples as members or non-members of the target concept. We assume that the trainer agent does not have access to the internal knowledge representation of the trainee agent, but can evaluate its concept recognition abilities by asking it to categorize selected exemplars and non-exemplars of the target concept. Though there can be additional research issues that relate to the transfer of concept description knowledge between these two agents, we focus on the incremental selection of training examples by the trainer to expedite the learning of the trainee agent. The primary advantage of concentrating on the iterative selection of training examples from observed performance is that the core process can be independent of the internal knowledge representations and algorithms of the trainer and learner. A secondary benefit is that the developed procedure can:

- be used, in principle, by a software agent to teach a human user.

Though we do not concern ourselves with these aspects, it does open up fruitful avenues for future work. While a number of researchers have looked into the research issues involved in agents learning concurrently or cooperatively, there is little work on a teacher or trainer agent training a trainee agent through an iterative process of knowledge refinement. In this paper, we first present the general architecture of the Agent Teaching Agent (ATA) framework. We assume that the original set of training instances available to a learner is no longer available when it is attempting to teach the concept to another learner. While this assumption can be violated in particular real-life situations, it allows us to develop a general-purpose training framework that does not depend on the availability of original training instances. This assumption is also justified if either of the following holds:

- the original training set size was too large to be carried around by a compact-sized learner, e.g., mobile agents,

- the teacher had no reason to believe that it will need those training instances in the future and hence got rid of them after its own learning was over, e.g., the teacher did not foresee its teaching role.

In this paper, we report on results from experiments using an instance-based learning algorithm, IB2 (Aha, Kibler, & Albert 1991), and a decision-tree based learning algorithm, C4.5 (Quinlan 1993) as trainer and trainee agents. For initial development and evaluation we used a set of artificial concept descriptions. We have also started evaluating our methodology on some data sets from the UCI machine learning repository (Murphy & Aha 1992) and present our initial results here.

ATA Framework

The motivation for our work has been that a teacher agent can guide the learning process of a learner agent by observing the latter's problem solving performance. In the context of concept learning, this means that based on the success and failure of the trainee in classifying given exemplars, the trainer can choose an ap-
appropriate sequence of training examples to guide the learning process of the trainee.

The basic ATA framework architecture is presented in Figure 1. The Trainer agent first acquires the target concept from its interaction with an environment, and using its learning module. This learning process produces a target concept description in the internal knowledge representation format of the trainer agent. These formats can range from logical rules, decision trees, stored instances (Aha, Kibler, & Albert 1991), neural networks, etc. The trainer agent also has a training module which interacts with the trainee module and provides successive training and testing set to train and evaluate the progress in learning of the trainee agent. The trainee learns its own concept description from the set of classified training examples provided by the trainer. It also classifies each of the unclassified test examples provided by the trainer and returns these classified instances to the trainer for evaluation.

We envisage an iterative training procedure in which alternatively the trainer selects a set of training and testing exemplars, the trainee trains using the training set and then classifies the testing set, the trainer observes errors made by the trainee in classifying the instances in the last testing set and accordingly generates the next training and testing sets. This iterative process converges when error of the trainee falls below a given threshold. We present these iterative training steps in an algorithmic form in Figure 2.

The algorithm presented in Figure 2 needs to be further fleshed out to realize an actual implementation. In particular, we have to specify procedures for selection of the initial training and testing sets, \( N_0 \) and \( T_0 \), and the generation of the next test set \( T_{i+1} \) based on the mistakes, \( M_i \), made by the trainee on the current test set.

We first present the underlying principles for designing these procedures. When selecting the initial training and testing instances, the goal is to select the most discriminating examples that help identify regions of the input space that do and do not belong to the target concept. For example, if a hyperplane separates instances of the target concept from non-instances, then points close to and on both sides of that hyperplane should be selected as initial training and testing set members. When selecting the next set of training and testing instances, the goal is to first isolate the mistakes made on the previous test set, and for each of these instances, find a few neighboring points, use some of them as part of the training data and the rest as part of the test data in the following iteration. Note that the true classification of these points will not be known in general, and only their estimated classification, based on the concept description knowledge previously acquired by the trainer, can be used.

The actual procedure for selecting the incremental training and testing sets depend on the internal representation used by the trainer agent. We have developed these procedures for instance based and decision tree learners to work on problems with real-valued attributes. In our experiments, we have normalized all attribute values to lie in the range \([0,1]\).

**Trainer: Instance Based Learner**

An instance based learning algorithm, IB2, is used as our first trainer. IB2 is an incremental learning algorithm that stores a subset of the training examples as its concept description (Aha, Kibler, & Albert 1991). To select \( N_0 \) and \( T_0 \), we first sort, in increasing order, the stored points by the nearest distance to another stored point of the opposite classification. So, the sorting metric for any stored instance \( s \) in the set of stored instances, \( S \), is

\[
\min_{r \in S \land c(s) \neq c(r)} d(r, s),
\]
Procedure newTrainingInstances(M) begin
  \( N \leftarrow \phi \)
  for each \( m \in M \) begin repeat
    \( p \leftarrow \) a point in \( \delta \) neighborhood of \( m \)
  until \( c_{\text{trainer}}(p) \neq c_{\text{trainer}}(m) \)
  \( N \leftarrow N \cup \{p\} \)
  \( r \leftarrow \) nearest stored point of opposite classification to \( m \)
  while \( d(p, r) > \epsilon \) do begin\( q \leftarrow \) midpoint of \( p \) and \( r \)
    if \( c_{\text{trainer}}(q) = c_{\text{trainer}}(p) \)
      \( p \leftarrow q \)
    else \( r \leftarrow q \)
  end
  \( N \leftarrow N \cup \{q, r\} \) end
end

Procedure newTestInstances(M) begin
  \( N \leftarrow \phi \)
  for each \( m \in M \) begin
    \( r \leftarrow \) nearest stored point of opposite classification to \( m \)
    repeat
      \( p \leftarrow \) a point in \( \delta \) neighborhood of \( r \)
    until \( c_{\text{trainer}}(p) \neq c_{\text{trainer}}(r) \)
    \( N \leftarrow N \cup \{p\} \) end
end.

Figure 3: Algorithm for generating new training and testing points to be included in the next iteration.

where \( d(r, s) \) is the distance between the points \( r \) and \( s \) in the input space, and \( c : X \rightarrow \{0, 1\} \) is the learned boolean concept description function that classifies any instance in the input space \( X \) as a member or non-member of the target concept. Let this sorted list, \( L \), be further divided into \( L_1 \) and \( L_0 \), viz., lists containing members and non-members of the target class. In our experiments we started with approximately 30% of stored points as the initial training set and 15% of the stored points as the initial test set. Equal number of instances of both classes were chosen for the initial test and training sets by going down the sorted lists \( L_0 \) and \( L_1 \).

For selecting \( T_{i+1} \), the next set of training examples, for each point \( m \in M_i \), we first randomly generate another point, \( p \), within a hyperrectangle of sides \( \delta \) centered at \( m \) (we have used \( \delta = 0.05 \) in our experiments).

We discard this point if it does not have the same classification as \( m \) according to the the trainer’s learned concept description function \( c_{\text{trainer}} : X \rightarrow \{0, 1\} \). This process is repeated until another point \( p \) of the same classification as \( m \), according to the trainer’s knowledge, is found. Next we find the point \( r \) that is the closest stored point of opposite classification to \( p \). We then use a halving procedure to find two points on the line joining \( p \) and \( r \) that are within a distance \( \epsilon \) of each other and are of opposite classification according to the trainer knowledge. These two points and the point \( p \) is included in the training set for the next iteration. The goal is to present a more detailed depiction of the decision surface by choosing two points of opposite classification close to the surface. The procedure for the selection of new training and test instances is presented in Figure 3. Note that the next test set includes a point within a \( \delta \)-neighborhood of the point \( r \) (see above) and of the same classification as \( r \) according to the trainer’s knowledge.

The process described above is independent of the internal knowledge representation of the trainee. We found that while training a C4.5 based learner, additional improvement can be obtained by presenting point pairs that are within a distance of \( \epsilon \) of each other, are of opposite classes, and the line joining them is axis parallel. To achieve the last condition, we first find the point \( r \) as the nearest point of opposite classification to the mistake point \( p \), and then find a point \( x \) such that the line segment joining \( r \) and \( x \) is axis parallel and \( x \) is in the \( \delta \) neighborhood of \( m \).

Trainer: Decision tree learner

We have also used C4.5, a decision tree based algorithm (Quinlan 1993) as the learning mechanism for the trainer. While using C4.5 as the trainer, we generate a few points from each hyperrectangle corresponding to leaf nodes of the decision tree. For each leaf node, we keep the current bounds of the hyperrectangle given by a set of pairs \( \{x_{il}, x_{ih}\} \) for each \( i \) attributes in the domain, where \( x_{il} \) and \( x_{ih} \) represent the lower and upper bounds of the \( i \)th attribute. If the problem domain has \( n \) attributes then we will have \( n \)-dimensional hyperrectangles as decision regions. For each vertex of such a hyperrectangle, we generate 1 internal point, at a distance \( \delta \) from the vertex and on the hyperdiagonal, and \( n \) external points, one across each of the decision surfaces at that vertex and each at a position of the mirror image of the internal point when reflected about the hyperplane separating them.

If the trainee is an incremental learner, e.g., IB2, it is recommended that a particular order of presenting the training data is followed. We Starting at some vertex, \( C_0 \) of a hyperrectangle corresponding to any leaf node of the decision tree. We first present the internal data point at that vertex followed by each of the external data points. After finishing with one vertex, we move to an adjacent, unvisited vertex. Suppose while moving from the \( C_i \) to the \( C_{i+1} \)th vertex we moved
**Procedure** Train-Agent(Trainer, Trainee, Trainer-knowledge)

begin

Select initial training set $N_0$ and initial testing set $T_0$ from trainer-knowledge

$i ← 0$

repeat

train trainee agent on training set $N_i$

let $M_i$ be the instances in $T_i$ misclassified by trainee after training on $N_i$

$T_{i+1} ← T_i \cup \text{newTestInstances}(M_i)$ and trainer-knowledge

$N_{i+1} ← N_i \cup \text{newTrainingInstances}(M_i)$

$i ← i + 1$

until ($|M_i| < \text{threshold}$)

end.

**Figure 2:** Algorithm for incremental training.

along the axis $j$, i.e., the only attribute value different between the two vertices was the value of the $j$th attribute. At vertex $C_i + 1$, we first consider the outside point found if we traverse along the same dimension that we just moved. If this data point is of the same classification as the last internal data point presented, then we keep presenting other external points at this vertex, in random order, until a point of opposite classification is found. At this point, the internal data point at this vertex is presented followed by the remaining external data points. The goal is to ensure that an incremental learner will consider all of the data points presented, and will not discard any because it could be classified easily based on the previously seen data points. Though such overlooking cannot be eliminated, this order of selection increases the likelihood that an incremental learner, e.g., IB2, will give importance to every instance in the training set. The initial test set is chosen by selecting points randomly at the vertices of each hyperrectangle. The test and training sets at subsequent iterations are formed by adding more points that are close to a mistake point and across a decision surface from it.

**Results**

For the first set of experiments we have chosen an artificial data set as it allows for controlled experimentation and helped us both develop and evaluate the algorithms for selection of incremental test and training data sets. Later we will also present results with larger, real-life data sets to show the general applicability of the ATA approach.

All of the artificial problems we have experimented with are defined on 2 continuous attributes, where the domain of each attribute is $[0, 1]$. The problem set is presented in Figure 4. In each problem, the non-shaded region represents the target concept. We generated 1000 points from a uniform distribution over the input space. We then ran experiments with five-fold cross-validation, i.e., the results were averaged over five training-test set pairs. The size of each training set was 800 and that of each test set was 200.

Now, we present results when IB2 was used as the trainer and C4.5 as the trainee agent and the 2/1 problem is used as the data set. As the hyperplanes identifying the concept regions are not axis-parallel, this problem is somewhat difficult for C4.5 to learn. Basically, C4.5 has to generate a large tree so that it can approximate diagonal lines with steps of axis-parallel lines. We record the classification accuracy of the trainee agent over the entire test set for successive presentations of the incremental training and test sets. The interesting thing to note is that incremental presentation using the ATA framework results in C4.5 doing almost as well as it would have done if the entire training set was available to it (results are not statistically significantly different). Moreover, this performance is achieved only with about 300 data points as compared to the original training set size of 800. In Figure 5 we show how the incremental training improves learning in case of the 2/1 concept.

To ensure that the procedure for selecting incremental training and test points is working effectively we wanted to compare it with some non-directed incremental training scheme. We ran another set of experiments where randomly selected points were classified by the trainer using its learned knowledge and then presented as instances in the iterative training phase. The increase in performance of the trainee with increase in
training instances was significantly worse compared to the ATA framework which chooses instances based on the mistakes made in the previous training iteration.

We next used C4.5 as the trainer agent and IB2 as the trainee agent and the corresponding results are presented in Figure 6. When C4.5 is trained with 800 training instances it learns 27-31 rules for different training sets of 5-fold cross validation. The initial training set for IB2 generated according to our algorithm given in section 2.2 contains 115-125 examples. In all of the 5-fold cross validations, we got an impressive testing accuracy close to 75% after the first training iteration. After 12-17 iterations we could reach the final testing accuracy in the range of 92.5% to 94.5%. The final training set size varies in the range of 415 to 460 where as IB2 itself has an accuracy of 94% over 800 training examples.

As in the case of IB2 teaching C4.5, here also we find that incremental training results in rapid improvement in classification performance of the trainee agent over the entire test set. The final accuracy is a little, but not significantly, less than the accuracy of the trainer’s knowledge or the trainee’s knowledge if it had access to the entire training set. Again, far fewer than the entire training set size of 800 points is used in the incremental version.

The above pair of experiments helped us establish the baseline performance of the ATA framework. To show generality, however, we had to experiment with more data sets. We first tested on the other artificial problems: the 4/1 (a pictorial presentation of this problem can be found at www.mcs.utulsa.edu/~karpa/ATA.html) and the 3/2 data sets. The results are tabulated in Table 1 where IB2 is the trainer and C4.5 is the trainee agent and in Table 2 where C4.5 is the trainer and IB2 is the trainee agent.

In these tables the different columns have the following interpretation: \( TTr \) presents the training accuracy of the training agent. \( TTe \) presents the testing accuracy of the training agent. \( LTe \) presents the testing accuracy of the trainee agent when trained over the entire training data set which was used to train the trainer agent. \( L\text{set}Te \) presents the testing accuracy of the trainee agent when all the points stored by IB2 are given to the trainee in one shot. \( LTe \) presents the testing accuracy of the trainee agent when training examples are presented incrementally.

The general pattern for the results with the 4/1 and the 3/2 data sets are consistent with the results from the 2/1 data set: incremental training produces better testing accuracy by the trainee than when it receives all the stored points in one shot (when IB2 is training...
C4.5). Incremental training performance is slightly inferior to the performance on the test set of the trainer or the trainee when they had access to the entire original training set. The interesting result is that approximately equivalent performance is produced by incremental training with far less training instances. The only weak result is that the incremental trainee results are inferior in the case of the 3/2 data set. Actually, we find that C4.5, as a trainee, performs poorly even when all stored points from IB2 are presented. This level of performance is observed because of the relatively difficult target concept which has many non-axis-parallel boundaries. With few training points per region, some of the decision surfaces are not closely approximated. We expect the results to improve if the training set sizes are significantly increased.

These tables also contain results from testing the ATA framework on larger, real-life data sets. The real-life datasets were selected is the from the Machine Learning Repository hosted by the CS department of the University of California Irvine (http://www1.ics.uci.edu/~mlearn/MLSummary.html). Results from the Pima and the Iris plant databases were similar to the 2/1 and 4/1 data set results in the sense that incremental training produced trainee classification accuracies that were a little inferior to the trainee’s performance when it had access to the entire training set. The incremental training performance were still consistently high.

Particularly interesting results, however, were obtained with the Haberman data set. We performed 5-fold cross validation on the data set. The average test accuracy of IB2 and C4.5 algorithms on this data, when trained on 80% of the instances and averaged over the 5-fold cross-validation, are 56.08% and 68.72% respectively. We obtained an average testing accuracy of about 67.1% with C4.5 as the trainee agent and using the incremental training scheme in the ATA framework. It is interesting to note that IB2 can train C4.5 to classify much better than it can do itself! This may suggest that the bias of the decision tree learner is better suited for this particular concept learning problem. The other significant result is that when IB2 is the trainee, the trainer C4.5 can, through incremental training, train it to perform better (63.52% training accuracy) than when IB2 learned by itself on the entire training set (54.1% testing accuracy). This result indicates that having a competent learner as a trainer can at times lead to better learning than working with raw training examples.

Related Work and Conclusions

Though there have been considerable research on agents learning concurrently or cooperatively (Prasad, Lesser, & Lander 1996), or even learning by observing opponent’s behavior (Carmel & Markovitch 1996), there has been little work in which an agent proactively trains another agent. The most relevant work comes from one learner telling another agents what portions of the search space to ignore (Provost & Hennessy 1996), problem-solving traces or even learned policies (Tan 1993) with another concurrent learner. We need to work on using other knowledge representation schemes to extract decision surfaces learned from the training data. While some recently developed algorithms like Support Vector Machines (Burges 1998) lend itself easily to such introspection, more involved processing would be needed to work with opaque learned knowledge like neural networks. Pool based active learning has been used in SVM instead of randomly selected training set (Tong & Koller 2000). They have shown that the active learning has significantly reduced the need of labeled training instances in both standard inductive and transductive settings. Research in active learning typically target presentation of training points based on a precise understanding of the internal knowledge representation of the learner.

We are currently running experiments on a wider set of problem instances that contain both artificial and real-life data and plan to include the results in the next update of this paper. We plan to use a support-vector machine based agent as trainer and trainee agents in conjunction with the current instance-based agent and decision-tree based agents. We also plan to explore possible combinations of active learning and ATA approaches.

References


Murphy, P., and Aha, D. 1992. UCI repository of machine learning databases [machine-readable data repository].


