Argumentation-based Distributed Induction

Santiago Ontañón[1] and Enric Plaza[2]

[1] CCL, Cognitive Computing Lab,
Georgia Institute of Technology, Atlanta, GA 303322/0280,
santi@cc.gatech.edu
[2] IIIA-CSIC, Artificial Intelligence Research Institute
Campus UAB, 08193 Bellaterra, Catalonia (Spain),
enric@iiia.csic.es

Abstract. Argumentation can be used by a group of agents to discuss about the validity of hypotheses. In this paper we propose an argumentation-based framework for distributed induction, where two agents learn separately from individual training sets, and then engage in an argumentation process in order to converge to a common hypothesis about the data. The result is a distributed induction strategy in which the agents minimize the set of examples that they have to share in order to converge to a common hypothesis. The proposed strategy works for any induction algorithm which expresses the hypothesis as a disjunction of rules. We show that the strategy converges to a hypothesis indistinguishable in training set accuracy from that learned by a centralized strategy.

Keywords: Distributed Induction, Argumentation, Classification.

1 Introduction

Distributed induction is the problem of learning a hypothesis or model (like a set of rules, or a decision true) from data when the data is distributed among different sources. Some real-life domains involve such forms of distributed data, where data cannot be centralized due to one or several of the following reasons: storage size, bandwidth, privacy, or management issues. Storage size and bandwidth are less and less a problem nowadays, however in large data sets they might still be an issue. In this paper we will propose a framework in which agents will use a limited form of argumentation in order to arrive to a hypothesis of all the data while minimizing the communication, and specially minimizing the amount of examples exchanged, and ensuring that the hypothesis found is exactly as good as if centralized induction with all the data was used.

Argumentation frameworks can be used in multi-agent systems for different purposes like joint deliberation, persuasion, negotiation, and conflict resolution [12]. In a previous work [8] we have shown how argumentation can be used by agents that use lazy learning techniques to solve classification tasks. In this paper we will introduce a framework where agents use argumentation to argue about hypotheses. In this framework, agents will generate hypotheses locally, and then argue about them until both agents agree. During the argumentation process, agents might exchange some small number of examples.

Formalizing agent communication as argumentation allows the distributed induction strategies to abstract away from the induction algorithm used by the agents. Thus, all the strategies presented in this paper can work with any induction algorithm that satisfies certain requirements. In particular, we require that the hypotheses learnt can be expressed as a disjunction of independent rules. So, for instance, algorithms such as FOIL [11], or ID3 [10] (since a tree can be easily flattened into a set of rules), or other rule learners can be used. Algorithms such as CN2 [4] that learn an *ordered* set of rules also fit in this framework, but rules require some preprocessing to remove the dependencies that the ordering introduces (as elaborated in Section 2). Moreover, the framework is also agnostic in regards to the representation formalism, in the experiments section we will show results with both relational as well as flat feature-vector representations.

The remainder of this paper is organized as follows. Section 2 presents our multi-agent learning framework, including our formalism for argumentation. Section 3 presents two strategies for distributed induction based on argumentation, and Section 4 empirically evaluates them, comparing them to other distributed induction strategies in the literature. Section 5 provides a quick overview of the related work, and finally the paper closes with conclusions and future work.

2 A Framework for Multi-Agent Learning

Let A_1 and A_2 be two agents who are completely autonomous and have access only to their individual and private training sets T_1, and T_2. A training set $T_i = \{e_1, ..., e_n\}$ is a collection of examples. Agents are able to individually apply induction in order to learn a hypothesis (or model) of the data and solve problems using the induced model, but they can also collaborate with other agents for both induction and problem solving. In the rest of this paper we will use the terms model and hypothesis indistinguishably.

2.1 Examples, Hypotheses and Rules

Examples, *hypotheses* and *rules* are the three key concepts of the learning framework proposed in this paper.

We will restrict ourselves to analytical tasks, i.e. tasks like classification, where the solution of a problem is achieved by selecting a solution class from an enumerated set of solution classes. In the following we will note the set of all the solution classes by $\mathcal{S} = \{S_1, ..., S_K\}$. Therefore, an *example* $e = \langle P, S \rangle$ is a pair containing a problem P and a solution class $S \in \mathcal{S}$. In the remainder of this paper, we will use the dot notation to refer to elements inside a tuple; e.g., to refer to the solution class of an example e, we will write $e.S$.

Our framework is restricted to hypotheses H that can be represented as a disjunctive set of *rules*: $H = \{r_1, ..., r_m\}$. A rule $r = \langle H, S \rangle$ is composed of a *head* $r.H$, and a *solution*, $r.S$. When a problem P matches the head $r.H$ of a

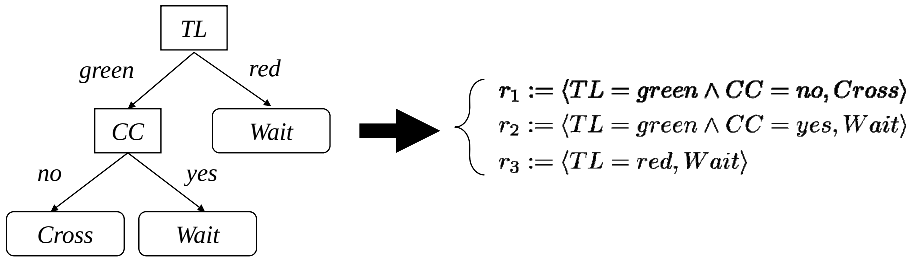

Fig. 1. Conversion of a decision tree to a set of rules.

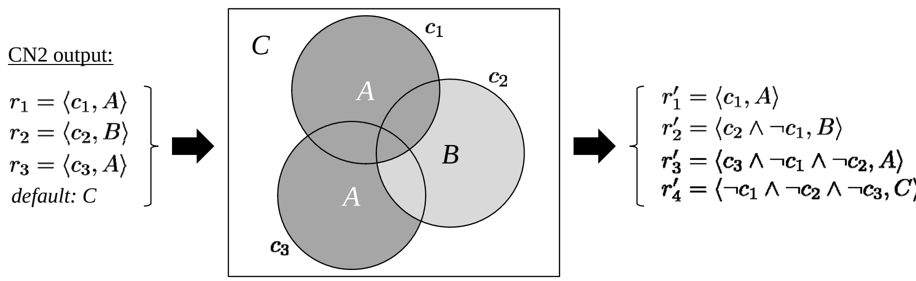

Fig. 2. Postprocessing of the rules generated by CN2 in order to remove the order dependencies, and thus fit in our argumentation framework.

particular rule r, the rule predicts that the solution to the problem P is $r.S$. When a problem matches the head of a rule $r.H$, we say that the head *subsumes* the problem: $r.H \sqsubseteq P$. A large number of induction algorithms can generate hypothesis that can be represented using this formalism. In particular, in our experimental section we have selected to use INDIE [1], which is a heuristic relational inductive learner, ID3, and CN2, but other algorithms could have been used such as AQR [6], or FOIL [11].

The framework introduced in this paper does not specify which induction algorithm or representation formalism agents use. In principle, any induction algorithm could be used, and any data representation (propositional, relational, or any other) could be used. The restriction on the hypothesis representation is imposed because agents will argue about each one of these rules independently.

To illustrate how different induction algorithms can represent their hypothesis using our formalism, Figure 1 shows he equivalence between a decision tree and a set of rules. In the example, a simple decision tree with only two features (TL and CC) is shown, and three rules equivalent to the tree are constructed.

Further, as we mentioned earlier, when using algorithms such as CN2, that produce an ordered set of rules, the rules produced have to be postprocessed in order to remove the order relationship among them. The left hand side of Figure 2 shows a set of three rules generated by CN2 (plus the default class assigned by

CN2 when no rule covers a problem). The center of Figure 2 shows a graphical representation of the way these rules partition the problem space among the three different solution classes A, B, and C (the three circles represent the subset of problems that are subsumed by each of the three conditions in the head of the rules: c_1, c_2 and c_3). Notice for instance that rule r_2 states that all the problems that are subsumed by c_2 have solution B. However, r_2 is only considered if r_1 is not fired. Therefore, that rule is postprocessed and converted into rule r'_2, which states that all examples that are subsumed by c_2, but not by c_1 have solution B. In general, a rule is postprocessed by adding the negations of all the previous rules to its head. Finally, the default solution computed by CN2 is converted also into a rule containing the conjunction of the negation of the head of all the rules generated by CN2. The result of this process is a set of independent rules, which can be used in our framework.

As illustrated by the previous two examples, a large collection of induction algorithms can represent their hypotheses in the form of rules.

2.2 Arguments and Counterarguments

In order to use argumentation, two elements must be defined: the *argument language* (that defines the set of arguments that can be generated), and a *preference relation* (that determines which arguments are stronger than others). In our framework, the argument language is composed of two kinds of arguments:

- A *rule argument* $\alpha = \langle A, r \rangle$, is an argument generated by an agent $\alpha.A$ stating that the rule $\alpha.r$ is true.
- A *counterexample argument* $\beta = \langle A, e, \alpha \rangle$, is an argument generated by an agent $\beta.A$ stating that a particular argument $\beta.\alpha$ is incorrect, because the example $\beta.e$ is a counterexample of such argument.

To define the relation among arguments, we have to take into account all the possible different situations that can arise while comparing two arguments consisting of rules or examples. Figure 3 shows all these situations. The top row of Figure 3 considers all the possible comparisons of two rule arguments, r_1 and r_2 such that $r_1.S = r_2.S$. Only three situations might arise: a) r_1 and r_2 are totally unrelated, b) the set of problems covered by r_1 and r_2 has non empty intersection, and c) one is more general than the other. The middle row of Figure 3 considers all the possible comparisons of two rule arguments, r_1 and r_2 but this time $r_1.S \neq r_2.S$. The same three situations arise (unrelated, non-empty intersection, and one more general than another). Notice that in the non-empty intersection situation we also require that no rule is more general than another (we don't include the extra restriction in the figure for clarity). Thus, when comparing any two rule arguments, only 6 situations might arise. Situations a), b), c) and d) represent rule arguments that are *compatible*, whereas situations e) and f) represent *conflicting* arguments. Moreover situation c) is a special situation and we say that r_1 *subsumes* r_2.

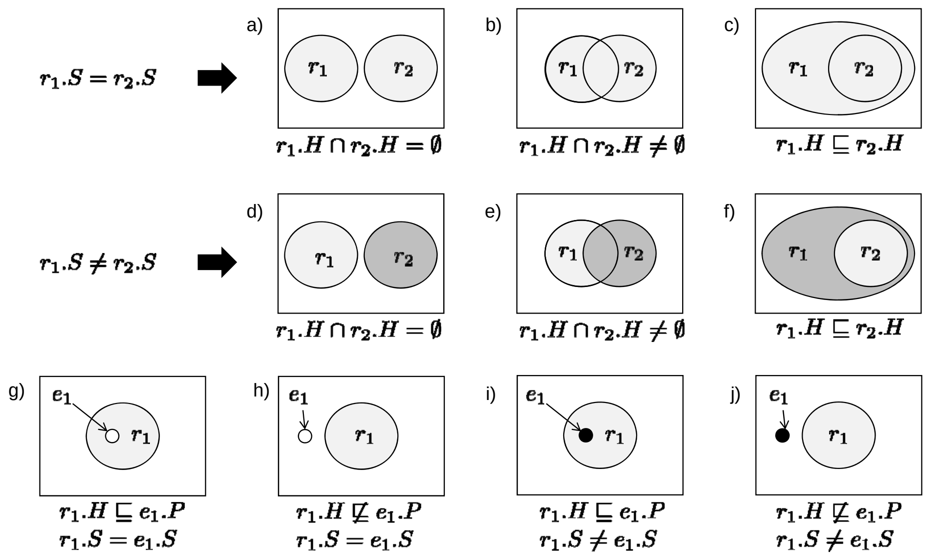

Fig. 3. All the possible different situations that can arise while comparing two arguments consisting of rules or counterexamples.

The third row of Figure 3 shows all the possible situations that arise when comparing a rule argument with a counterexample argument: g) both the counterexample and the rule support the same class (in which case the counterexample is not such, and both arguments endorse each other), h) in which the counterexample, although supporting the same class, is not covered by the rule, i) where the counterexample supports a different solution than the rule, and the rule covers the counterexample, j) in which the counterexample, although supporting a different class, is not covered by the rule. In our framework, we assume that a counterexample cannot be defeated, and thus only the rule arguments can be defeated. Out of the four situations, the counterexample argument only defeats the rule in situation i), in all the other situations, they are compatible. The counterexample in situation i) is called a *defeating counterexample* of r_1.

Using these two types of arguments and the *compatible*, *conflicting*, *subsumed*, and *defeated* relations among arguments, next section introduces two different distributed induction strategies.

3 Argumentation-based Distributed Induction

In this section we will present two strategies, ADI (Argumentation-based Distributed Induction) and RADI (Reduced Argumentation-based Distributed Induction), based on argumentation for distributed induction. Since the strategies involve communication among agents, they will be presented as communication protocols. Both strategies are based on the same idea, and share the same high level structure.

1. A_1 and A_2 use induction locally with their respective training sets, T_1 and T_2, and obtain initial hypotheses H_1 and H_2 respectively.
2. A_1 and A_2 argue about H_1, obtaining a new H_1^* derived from H_1 that is consistent with both A_1 and A_2's data.
3. A_1 and A_2 argue about H_2, obtaining a new H_2^* derived from H_2 that is consistent with both A_1 and A_2's data.
4. A_1 and A_2 obtain a final hypothesis $H^* = H_1^* \cup H_2^*$. Remove all the rules that are subsumed by any other rule (situation c) in Figure 3).

Basically, the intuitive idea is the following. In step 1 both agents perform induction individually. Then in steps 2 and 3 (which are symmetric, and can actually be performed in parallel), the agents use argumentation to refine the individually obtained hypotheses and make them compatible with the data known to both agents. Finally, when both hypothesis are compatible, a final global hypothesis H^* is obtained by just computing the union of all the rules learned by both agents, removing all the rules that are subsumed by some other rule (since those would be redundant). Notice that, unless the induction algorithms are not able to learn rules with 100% accuracy in the training set, there should not be any conflicting rules in H^* (at least not conflicting in the classification of the examples of the training set). However, there might be rules that conflict in the classification of problems outside of the training set. If the learning algorithm computes confidence levels for rules, those can be used to arbitrate, otherwise random arbitration can be used. ADI and RADI only differ in the way steps 2 and 3 are performed.

Step 2 in ADI works as follows

1. Let $H_1^0 = H_1$, and $t = 0$.
2. If there is any rule $r \in H_1^t$ that still has not been *accepted* by A_2, then send the argument $\alpha = \langle A_1, r \rangle$ to A_2. Otherwise, if all the rules in H_1^t have been accepted the protocol goes to step 5.
3. A_2 analyzes $\alpha.r$ and tries to find a counterexample that defeats it (situation i) in Figure 3). If A_2 can find such counterexample e, then A_2 sends the counterargument $\beta = \langle A_2, e, \alpha \rangle$ it to A_1. Otherwise, r is accepted and the protocol goes to step 2 again.
4. When A_1 receives a counterexample β, it appends $\beta.e$ to its training set T_1, and updates its hypothesis. If the induction algorithm of A_1 is not incremental, then A_1 can simply use induction from scratch with the new extended set of examples that includes e. A_1 updates the hypothesis obtaining H_1^{t+1}. The protocol goes to step 2 again, and $t = t + 1$.
5. The protocol returns H_1^t.

The main idea is that A_1 will generate hypotheses according to his local training set T_1, and A_2 evaluates them, trying to generate counterarguments to the hypotheses that do not agree with his own local data T_2. Step 3 in ADI is just the reversed, where it is A_2 that generates hypotheses, and A_1 that tries to rebut them with counterexamples. One characteristic of ADI is that at each step, only one counterexample is exchanged.

If the induction algorithm of A_1 and A_2 was capable of achieving 100% accuracy if it was given the complete collection of examples that both A_1 and A_2 have, then the protocol always ends up converging to a hypothesis that also has 100% accuracy in both T_1 and T_2. Moreover, in order to prevent infinite iterations in the case of noisy data, the agents are not allowed to send the same counterexample twice during the protocol. This ensures that the protocol will eventually end.

The second strategy, RADI, improves over ADI in trying to minimize the number of times the hypothesis has to be updated while keeping the number of counterexamples exchanged low. Step 2 in RADI works as follows:

1. Let $H_1^0 = H_1$, and $t = 0$.
2. Let $R^t \subseteq H_1^t$ be the set of rules in the hypothesis of A_1 not yet *accepted* by A_2. If such set is empty, then the protocol goes to step 5. Otherwise, A_1 sends the set of arguments $\mathcal{R}^t = \{\alpha = \langle A_1, r\rangle | r \in R^t\}$ to A_2.
3. For each $\alpha \in \mathcal{R}^t$, A_2 computes the set of examples C_α in its training set that are counterexamples that defeat $\alpha.r$: $C_\alpha = \{e \in T_2 | \alpha.r.H \sqsubseteq e.P \wedge \alpha.r.S \neq e.S\}$. For each argument $\alpha \in \mathcal{R}^t$ such that $C_\alpha = \emptyset$, $\alpha.r$ is accepted by A_2. Let $I^t \subseteq \mathcal{R}^t$ be the subset of arguments for which A_2 could find defeating counterexamples. A_2 computes the minimum set of counterexamples B^t such that $\forall_{\alpha \in I^t} C_\alpha \cap B^t \neq \emptyset$. That is the minimum subset of examples such that there is at least one counterexample that can defeat each argument in I^t. A_2 sends the set of counterexample arguments $\mathcal{B}^t$ consisting of a counterexample argument $\beta = \langle A_2, e, \alpha\rangle$ for each pair e, α such that $e \in B^t$, $\alpha \in \mathcal{R}^t$, and β defeats α.
4. When A_1 receives a set counterexample arguments $\mathcal{B}^t$, it appends all the examples on them to its training set T_1, and updates its hypothesis. If the induction algorithm of A_1 is not incremental, then A_1 can simply use induction from scratch with the new extended set of examples. A_1 updates the hypothesis obtaining H_1^{t+1}. The protocol goes to step 2 again, and $t = t + 1$.
5. The protocol returns H_1^t.

The idea behind RADI is that an example can be a defeating counterexample of more than one rule at the same time, thus, by selecting the minimum set of counterexamples, the number of examples exchanged is reduced. Also, by sending all the counterexample arguments at once, the number of times the hypothesis has to be updated is also reduced. This results in an efficient strategy for distributed induction that minimizes the number of examples being exchanged, and that can be used with any induction algorithm. Notice that finding such minimum subset of counterexamples is NP, however approximate methods to compute such minimum subset can be easily defined.

Figure 4 illustrates the first cycle of the RADI protocol. In the figure, agent A_1 has learnt a hypothesis H_1^0 from its original training set. A_1 sends the set

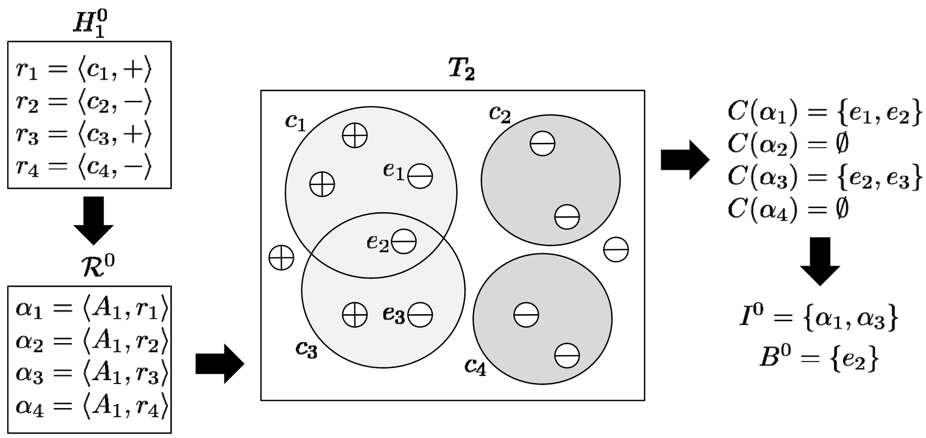

Fig. 4. An illustration of one step in the RADI strategy.

$\mathcal{R}^0$ of rule arguments to A_2. In the middle part of Figure 4, we can see the training set of A_2 (T_2). In this example, there are only two classes, + and -. A_2 evaluates all the arguments in $\mathcal{R}^0$ with his training set T_2. In particular, in this example, A_2 finds counterexamples for two of the arguments, namely α_1 and α_2. Thus, the set $B^0 = \{\alpha_1, \alpha_2\}$. Then, A_2 constructs the set B^0 consisting of the minimum set of examples that contain counterexamples for all the arguments in I^0. In this case, there is a particular example, e_2, which is a counterexample of both arguments, so e_2 is enough to contradict both. Therefore, A_2 will construct the set of counterarguments $\mathcal{B}^t = \{\langle A_2, e_2, \alpha_1 \rangle, \langle A_2, e_2, \alpha_3 \rangle\}$, and send it to A_1, which will update his hypothesis by appending e_2 to its training set T_1, and will generate a new updated hypothesis H_1^1, with which the next round of the protocol will start.

As explained in Section 6, it is part of our future work to investigate how the inclusion of additional types of counterarguments, such as rule counterarguments, can further reduce the amount of information exchanged during the distributed induction process.

4 Experimental Evaluation

In order to evaluate our approach, we tested the distributed induction strategies in four different data sets: three propositional data sets from the Irvine machine learning repository (soybean, zoology, cars), and a complex relational data set (sponges). Moreover, we tested it using three different induction algorithms: ID3, CN2 and INDIE (a relational inductive learner [1]). We also compared the results against centralized induction and also three other distributed induction strategies: individual (where agents just do induction individually), union (where agents do induction individually, and then they put together all the rules they learn into one common hypothesis), and DAGGER [5] (the only other distributed

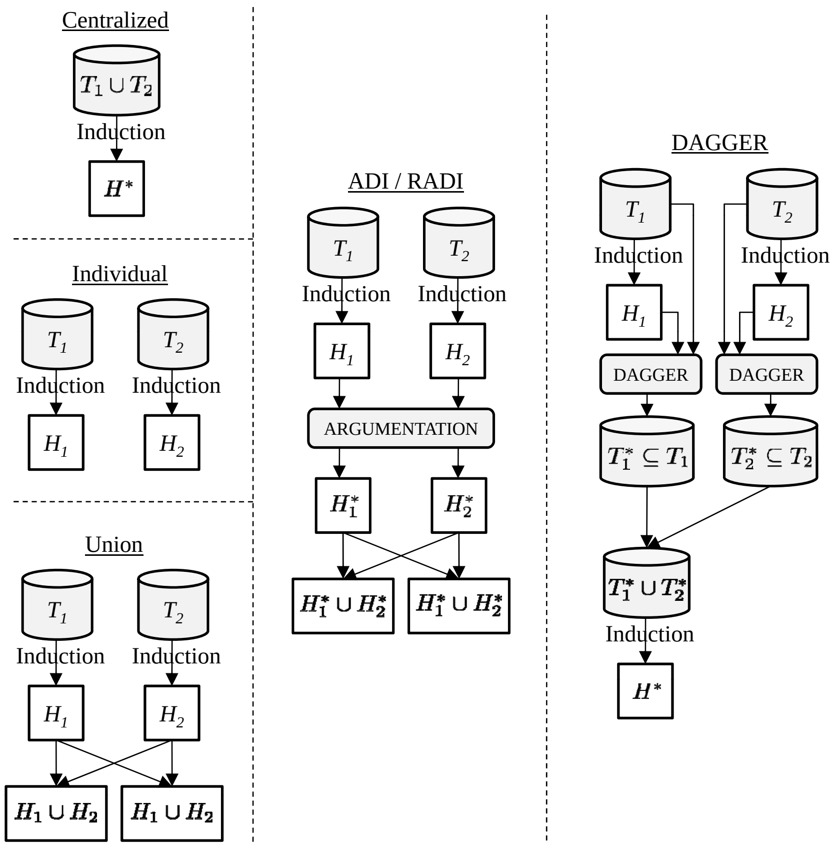

Fig. 5. An illustration of all the different distributed induction strategies evaluated in our experiments.

induction technique independent of the learning algorithm to the best of our knowledge, see Section 5 for a brief explanation of DAGGER). Figure 5 presents a visual overview of the different strategies used in our evaluation. We will evaluate convergence, time, number of examples exchanged, number of rules exchanged, number of induction calls, and both training and test set accuracy. All the results presented are the average of 10 fold cross validation runs.

Since sponge is a relational data set (introduced by Armengol and Plaza [1]) it has to be converted to propositional so that ID3 and CN2 can use it. In the sponge data set, examples are represented as trees, and the size of the trees varies greatly from an example to another. In order to convert it to a propositional representation, we computed the set of all possible different branches that the examples have, and each one is converted to a feature (70 different features are defined in this way). Each example consists of about 30 to 50 features each, so there is a large amount of missing values in the resulting propositional repre-

Table 1. Training and test accuracy measurements of different distributed induction strategies combined with different induction algorithms.

	Training				Test			
	Soybean	Zoology	Cars	Sponges	Soybean	Zoology	Cars	Sponges
ID3-ADI	100.00	100.00	100.00	99.70	**88.50**	**99.00**	88.95	58.21
ID3-RADI	100.00	100.00	100.00	99.74	87.67	**99.00**	89.24	58.21
ID3-centralized	100.00	100.00	100.00	99.44	85.00	**99.00**	88.95	58.57
ID3-individual	85.67	93.85	93.84	80.20	76.50	90.00	86.84	55.54
ID3-union	90.25	94.73	97.73	94.05	81.00	94.00	**90.99**	60.36
ID3-DAGGER	99.57	100.00	76.36	99.76	80.67	92.50	68.95	**62.50**
CN2-ADI	100.00	100.00	100.00	100.00	**84.90**	**93.50**	**80.61**	**79.11**
CN2-RADI	100.00	100.00	100.00	100.00	**84.66**	**93.50**	80.17	**78.93**
CN2-centralized	100.00	100.00	100.00	100.00	**84.66**	94.00	**80.64**	**78.57**
CN2-individual	87.82	94.62	89.90	88.29	77.83	87.50	**80.84**	74.46
CN2-union	54.91	91.65	80.41	70.71	63.66	86.00	80.00	68.20
CN2-DAGGER	99.49	99.65	95.86	99.88	79.33	92.50	75.34	**78.93**
INDIE-ADI	99.64	100.00	100.00	100.00	**84.33**	93.00	**91.25**	**95.89**
INDIE-RADI	99.64	100.00	100.00	100.00	**84.50**	**94.00**	**91.37**	94.11
INDIE-centralized	99.64	100.00	100.00	100.00	83.00	**94.00**	81.80	95.00
INDIE-individual	89.21	94.07	93.93	96.45	77.50	85.50	87.76	54.11
INDIE-union	91.44	96.48	97.42	97.90	78.00	90.00	**91.80**	94.29
INDIE-DAGGER	-	-	-	-	-	-	-	-

sentation. Thus, both ID3 and CN2 have troubles learning in this domain. CN2 does, in fact, a better job, but ID3 achieves a very low classification accuracy. Additionally, since the basic ID3 cannot handle missing values, all missing values where considered to have the special value "missing" when the data set was used by ID3. For CN2, a beam size of 3 was used in all the experiments.

Table 1 presents the classification accuracy (both measured in training set and in test set). We ran each combination of induction algorithm (ID3, CN2, INDIE) with distributed induction strategy (centralized, individual, union, DAGGER, ADI and RADI) with all the data sets (except the combination of INDIE-DAGGER, that is not possible, since DAGGER assumes propositional data sets, and INDIE requires them in relational form). In each experimental run the data set was split in two sets, a training set containing 90% of the examples, and a test set containing 10% of the examples. The training set was further split among two agents (except in the case of the centralized strategy, where there was only one agent). Accuracy is measured in the original training set (with 90% of the examples), and also in the remaining 10%, that forms the test set. The left hand side of Table 1 shows accuracy in the training set, and the right hand side shows accuracy in the test set.

Looking at the training accuracy, the first thing that the experimental results confirm is that the hypotheses learnt by ADI and RADI is indistinguishable in training set accuracy from the one learnt by using centralized induction.

Table 2. Time (in seconds) required to complete the induction process per agent, number of examples shared per agent (as a percentage of the number of examples owned by an agent), number of rules sent per agent, and number of times the base induction algorithm had to be invoked per agent (notice that in the "centralized" case, there is only one agent). All the results are average over all the induction algorithms and all the data sets.

	time	Examples shared	Rules sent	Induction calls
centralized	2.8	100.00%	0.00	1.00
individual	1.5	0.00%	0.00	1.00
union	1.5	0.00%	67.63	1.00
DAGGER	3.5	68.56%	64.75	1.50
ADI	155.4	19.04%	3748.70	58.90
RADI	18.2	21.52%	679.34	5.77

Achieving a 100% accuracy all the times where centralized induction also does. When agents perform individual induction, of course, training accuracy diminishes (since agents only learn with 50% of the data in the training set), agents using the union strategy improve their accuracy, but still it is not guaranteed to be as good as that of centralized accuracy (and in the case of CN2, where the order of the rules matter, the accuracy drops drastically). DAGGER shows good accuracy (although not guaranteeing that of centralized induction).

Analyzing test set accuracy, we observe that, except in a few cases where DAGGER achieves higher accuracy (and one where surprisingly union does [3]), ADI and RADI achieve same or higher accuracy than the centralized approach. Table 1 shows the highest results for each induction algorithm in boldface (when the difference was not statistically significant, more than one result is highlighted). The explanation is that when agents use ADI or RADI, two different hypothesis of the data are learnt (one per agent), and, after inconsistencies are fixed, they are merged. Therefore, the resulting hypothesis has potentially several rules that cover the same examples, but that were derived from different training sets (thus having different biases). This, alleviates overfitting, and thus increases classification accuracy in unseen problems. The effect achieved is similar to that of ensemble methods, but with the advantage of having a single hypothesis.

Another effect that can be seen is that ID3 and CN2 cannot properly handle the complexity of the sponges data set, since, although they can achieve high training set accuracy, the rules they learn do not generalize and achieve very low test set accuracy. INDIE, however, being a relational learner, can handle sponges in its native representation formalism, and thus learn much more general rules, that generalize properly, achieving high test set accuracy.

[3] We repeated that experiment several times, with identical result, we are still in the process of analyzing why union achieves such good result in the cars data set with ID3.

Classification accuracy, however, is only one side of the coin. Table 2 shows the amount of time used by each of the different distributed induction strategies per agent (averaged over all the data sets and induction algorithms), also the percentage of the examples owned by each agent that had to be shared, the number of rules exchanged, and also the number of times that the agents had to call the base induction algorithm. Notice, that time is dominated by the slower learning algorithm (CN2) and the most complex data set (sponges), the fastest algorithm (ID3) required less than a tenth of a second for any strategy except ADI and RADI (where it still required less than a second for any data set). Table 2 shows that ADI and RADI, are the most computationally expensive strategies, ADI taking 155.4 seconds and RADI 18.2, while centralized accuracy required only 2.8 seconds. Moreover, most of the time consumed by ADI and RADI corresponds to invocations to the base induction algorithm after receiving new examples. If an incremental induction algorithm such as ID5R or ITI [15] the amount of time consumed would be reduced drastically.

Table 1 shows that among all the distributed induction strategies, DAGGER is the one that requires exchanging the highest percentage of examples, 68.56%, while ADI and RADI exchange only 19.04% and 21.52% respectively. The union strategy, of course, does not force agents to exchange any example. However, ADI and RADI require the exchange of a large amount of rules, where as other strategies, such as DAGGER, or union only require sharing the final hypothesis. While ADI shares slightly a lower amount of examples, RADI requires only a tenth of the time, a fifth of the rules, and a tenth of the number of induction calls.

Summarizing the results, we can conclude that different distributed induction strategies have different strengths and weaknesses. Performing centralized induction has the problem of having to share all the examples, but achieves a high accuracy at the minimum computational cost. Next in line is DAGGER, which forces the agents to share most of their examples, but achieves a high accuracy also (although not guaranteed to be as high as centralized). On the other extreme, we have the individual and union strategies, that have the minimum computational cost, zero example exchange, but also the lowest classification accuracies. ADI and RADI sit in the middle, requiring the agents to share a small percentage of examples (around 20%), while ensuring the same or higher classification accuracy than centralized induction (especially in the test set, where they hypotheses learn by ADI or RADI have less overfitting). However, ADI and RADI have a higher computational cost. In between ADI and RADI, we can conclude that RADI is the most well balanced strategy, since it requires about a tenth of the computational cost, while only sharing a very small number of additional examples.

Moreover, notice that nothing stops ADI or RADI from working even if each of the agents had a different base induction algorithm.

5 Related Work

Two areas of work are related to our work, namely distributed induction and incremental learning. Distributed induction has been attempted mainly from four different approaches: computing statistics in a distributed fashion and then aggregating, sharing example, sharing hypotheses or viewing induction as search and distributing the search process. Let us present examples of each one of the approaches.

Caragea et al. [3] present a framework for induction from a set of distributed sources based on computing a collection of statistics locally in each one of the sources, and then aggregating them to learn a model. They prove that some learning algorithms, such as ID3, can be distributed in this way while still guaranteeing finding the same exact tree that would be found if all the data was centralized. Their framework restricts to feature value representations. The main difference with out work is that in their framework they assume a single agent trying to learn from scratch from a collection of distributed sources, while in our framework we assume a multi-agent system with agents that already have an initial hypothesis and improve it by interacting with other agents. A similar framework was presented by Bhatnagar and Srinivasan [2], but where they allow each agent to have a completely different of attributes, as long as all the tables owned by the agents can be put together using a *join* data base operation if they were copied in a centralized repository. Another difference of both these frameworks with ours is that both attempt at providing a framework for defining distributed induction algorithms. So, using those frameworks, algorithms such as ID3 or CN2 have to be adapted to work in a distributed fashion. In contrast, our research focuses on finding distributed induction strategies that can be built around standard induction algorithms without modifying them.

A different approach is DAGGER, presented by Davies [5] (and used in our experiments for comparison purposes). Davies proposes to perform distributed induction by selecting a reduced set of informative examples from each one of the distributed sources, and then perform centralized induction with the union of the reduced sets of examples. Davies' method had the goal of being an alternative to voting mechanisms for aggregating hypothesis, which is a different goal than the work presented in this paper. Thus Davies' approach is a one shot approach that does not ensure preserving classification accuracy, while our strategies do.

Another approach to distributed induction is that of Shaw and Sikora [14], where they propose to learn individual models in each one of the sources, and then combine them by using a genetic algorithm that uses specialized mutation and crossover operators for being able to merge the hypothesis. The goal with Shaw and Sikora's approach is just to distribute the induction task among agents, so that it becomes more efficient and parallelizable. Our goal is not to make the induction process more efficient, but to allow groups of agents to perform distributed induction by using their own induction algorithm, and ensuring high quality of the hypotheses found.

Another example of distributing induction for efficiency is that of distributing the search process of finding rules among a series of distributed processors.

Provost and Hennessy [9] propose to perform distributed search for rule learning, where each individual processor only searches with a subset of the data and proposes each candidate rule to the rest for verification.

Also related is the work on incremental learning, such as the incremental versions of the ID3 algorithm ID4 and $\widehat{\text{ID4}}$ by Schlimmer and Fisher [13], or ID5R and ITI by Utgoff [15], which allow learning a decision tree from a set of examples, and then update it with new examples at a lower cost than learning it from scratch again. These algorithms can be used to increase the efficiency of relearning the hypotheses in the techniques presented in this paper. Experiments to validate this claim are part of our future work.

Finally, the argumentation framework presented in this paper is complementary to that introduced in [8], where an argumentation framework for lazy learning called AMAL was presented. The idea behind AMAL is complementary to that of ADI and RADI. While in ADI and RADI agents perform induction in a collaborative fashion, and then they solve problems individually (by using the hypothesis learnt collaboratively), in AMAL, agents learn separately, and only collaborate during problem solving. Thus, AMAL is an argumentation model of multi-agent learning based on "solution merging", where as ADI and RADI is based on "hypothesis merging".

6 Conclusions and Future Work

In this paper we have introduced two different distributed induction strategies, ADI and RADI, that can be used on top of any induction algorithm capable of learning hypotheses that can be represented using an independent set of rules. ADI and RADI ensure that the hypothesis learnt will be undistinguishable in terms of training set accuracy from that produced by the base induction algorithm when learning from all the data. The main idea behind ADI and RADI is to let each agent perform induction individually, then argue about the learnt hypotheses to remove inconsistencies, and finally merge both hypotheses.

Experimental results show that, in addition to achieve the same training set accuracy than a centralized method, ADI and RADI have the advantage of obtaining hypotheses that are less prone to overfitting, and thus achieve higher test set accuracy. Moreover, ADI and RADI require sharing only about 20% of the examples of each agent in order to converge to the common hypothesis. ADI and RADI also require that the agents perform several calls to the base induction algorithm, and thus are better suited to be paired with incremental learning algorithms (but this is not required).

ADI and RADI use counterexamples as the only form of counterargument. However, we plan to investigate more complex argumentation protocols that let agents use rule also as counterarguments. The problem of that, is that the base learning algorithms would have to be modified to be able to take rules into account, in addition to the examples in the training set. This is related to the research in "argument based machine learning" by Možina et al. [7] where they modify the CN2 algorithm to take into account specific rules (arguments) in addition to examples for learning purposes. Our ultimate goal is to design distributed induction strategies that could be paired with any induction algorithm, and get as close as possible to not requiring the exchange of any example, while converging to the same hypothesis of a centralized learner. Additionally, we want to extend ADI and RADI to work with an arbitrary number of agents, and go beyond classification tasks.

Acknowledgements Support for this work came from the project MID-CBR TIN2006-15140-C03-01.

References

[1] E. Armengol and E. Plaza. Bottom-up induction of feature terms. *Machine Learning Journal*, 41(1):259–294, 2000.

[2] Raj Bhatnagar and Sriram Srinivasan. Pattern discovery in distributed databases. In *AAAI/IAAI*, pages 503–508, 1997.

[3] Doina Caragea, Adrian Silvescu, and Vasant Honavar. Decision tree induction from distributed, heterogeneous, autonomous data sources. In *In Proceedings of the Conference on Intelligent Systems Design and Applications (ISDA 03)*, pages 341–350. Springer Verlag, 2003.

[4] Peter Clark and Tim Niblett. The CN2 induction algorithm. In *Machine Learning*, pages 261–283, 1989.

[5] Winston H. E. Davies. *The Communication of Inductive Inference*. PhD thesis, University of Aberdeen, 2001.

[6] Ryszard S. Michalski, Igor Mozetic, Jiarong Hong, and Nada Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *AAAI*, pages 1041–1047, 1986.

[7] Martin Možina, Jure Žabkar, and Ivan Bratko. Argument based machine learning. *Artificial Intelligence*, 171(10-15):922–937, 2007.

[8] Santiago Ontañón and Enric Plaza. Learning and joint deliberation through argumentation in multiagent systems. In *AAMAS*, page 159, 2007.

[9] Foster John Provost and Daniel N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *In Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 74–79. AAAI Press, 1996.

[10] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[11] J. R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[12] Iyad Rahwan, Simon Parsons, and Chris Reed, editors. *Argumentation in Multi-Agent Systems, 4th International Workshop, ArgMAS 2007, Honolulu, HI, USA, May 15, 2007, Revised Selected and Invited Papers*, volume 4946 of *Lecture Notes in Computer Science*. Springer, 2008.

[13] Jeffrey C. Schlimmer and Douglas H. Fisher. A case study of incremental concept induction. In *AAAI*, pages 496–501, 1986.

[14] Michael J. Shaw and Riyaz Sikora. A distributed problem-solving approach to rule induction: Learning in distributed artificial intelligence systems. Technical Report ADA232822, Carnagie-Mellon University, 1990.

[15] Paul E. Utgoff. An improved algorithm for incremental induction of decision trees. Technical Report 94-072, UMass, 1994.