Learning Preferences for Collaboration

Eva Armengol

Artificial Intelligence Research Institute (IIIA - CSIC),
Campus de la UAB, 08193 Bellaterra, Catalonia, Spain
eva@iiia.csic.es

Abstract. In this paper we propose the acquisition of a set of preferences of collaboration between classifiers based on decision trees. A classifier uses a well-known algorithm (k-NN with leaf-one-out) on its own knowledge base to generate a set of tuples with information about the object to be classified, the number of similar precedents, the maximum similarity, and about if it is a situation of collaboration or not. We considered that a classifier does not collaborate when it is able to reach by itself the correct classification for an object, otherwise it has to collaborate. The mentioned set of tuples is given as input to generate a decision tree from which a set of collaboration preferences is obtained.

Key words: Machine learning, Classification, Learning preferences, Collaboration, Decision trees

1 Introduction

In Machine Learning the idea of cooperation between entities appears with the formation of *ensembles*. An ensemble is composed of several classifiers (using inductive learning methods), each one of them being capable of completely solving a problem. Since classifiers can provide different solutions for the same problem, the key issue of ensembles is how to aggregate the solutions proposed by the different classifiers. Perrone and Cooper [9] proved that aggregating the solutions obtained by independent classifiers improves the accuracy of each classifier on its own. In that approach the cooperation among entities is done by both sharing the results for the same problem and reaching an aggregated solution.

Plaza and Ontañón [10] take the idea of ensemble learning and apply it to multi-agent systems. These authors define a *commitee* as an ensemble of agents where each agent has its own experience and it is capable of completely solving new problems. Each agent in a commitee can solve problems but it can also collaborate with other agents in order to improve its accuracy. The difference between this approach and the most common approaches to multi-agent learning systems (MALS) is that in a commitee each agent is able to completely solve a problem whereas in MALS approaches each agent solves a part of a problem.

Related to the idea of ensemble there is also the idea of *meta-learning* whose aim is to construct a classifier from distributed knowledge bases. The idea is to combine the predictions of an ensemble of classifiers in order to obtain a *global* classifier. This global classifier establishes what could be seen as a set of preferences (since it is not a simple aggregation procedure) to give the final classification. Prodromidis et al. [11] analyzed meta-learning and give a simplified meta-learning scenario composed of the following phases:

1. the base classifiers are trained from the data,
2. each classifier generates independently a prediction for the data on a separate test set,
3. a meta-level training set is constructed from the test set and the predictions generated by each classifier on the test set,
4. the *meta-classifier* is trained from the meta-level training set.

What we propose in this paper is similar to both, ensembles and meta-learning. As in ensembles, our goal is to solve a new problem and we take the approach proposed by Plaza and Ontañón [10], that is to say, each classifier does not solve a part of a problem (like in the most common ensemble approaches) but it can solve completely the problem. The metaphor of our approach is the following: let us suppose that a physician has to diagnose a patient but he has not enough experience for this. The most usual behavior could be that this physician asks other colleagues for advice in diagnosing that patient. As long as the physician interacts with others for solving problems that initially were outside of his experience, he acquires in turn, experience on this kind of problems and, consequently, the interaction with other experts will be reduced.

In a previous work [1] we implemented this scenario and we proposed that the agents can take benefit from the collaboration with other agents by learning domain knowledge. Our point was that if agents are able to justify the solution they provide, then agents receiving these justifications could use them as new domain knowledge (like domain rules). The idea of taking benefit from cooperation between learners was pointed out by Provost and Hennessy [12]. These authors shown how individual learners can share domain rules in order to build a common domain theory from distributed data and how this improves the performance of the whole system.

In the current paper we are interested to show how one individual agent can improve its own domain knowledge from the collaboration with other agents. To do that we analyze the answer of a question that has to be taken into account before to start the machinery described in [1]: when an agent prefers to ask for collaboration instead to give the classification it has reached using its own experience? In the previous work we assumed that this collaboration is done when the classification has not enough support. However now we take a closer look on the agent's own competence and learn situations where the agent prefers the classification it has obtained and when it prefers to ask other agents.

The paper is organized as follows. In Section 2 we present the scenario and introduce the elements that will be used as input for learning. In Section 3 we describe the procedure to construct the preference rules for collaboration.

2 Scenario

Let us suppose a classifier capable to solve problems of a given domain. Domain objects are described by sets of attribute-value pairs and each object has associated a class label belonging to a set $\mathcal{C} = \{C_1 \ldots C_n\}$. We assume that all the domain objects are described using the same set of attributes. The experience of the classifier is formed by a knowledge base containing domain objects with its class label, i.e., $\langle O_i, C_j \rangle$. Given a problem p to classify, the classifier uses the *k-Nearest Neighbor* (k-NN) algorithm [5, 4] on the knowledge base to obtain the class label for p. The k-NN algorithm uses a similarity measure to assess the similarity between the object p and each one of the domain objects in the knowledge base. The outcome of k-NN is the set of the k objects most similar to p.

What the classifier knows about its own knowledge are the problems in its knowledge base. Therefore the knowledge base is the only source from which it can learn about its own competence. The procedure we propose now is similar to the one described in [13], but here the classifier tries to solve its own problems. In other words, for each $\langle O_i, C_i \rangle$ in its knowledge base, the classifier takes O_i and uses k-NN for classifying it. Let us suppose that k-NN proposes as classification for O_i the class C_j. In such situation there are three possible scenarios:

1. $C_j = C_i$, i.e, O_i has been classified correctly.
2. $C_j \neq C_i$, i.e, O_i has been classified incorrectly.
3. k-NN proposes more than one class for the object.

This procedure can be done either for all the objects of the knowledge base (using leave-one-out) or for a selected subset of objects. Each object O_i is a domain object described by a set of attributes $\mathcal{A} = \{a_1, \ldots, a_n\}$ each one with a value that may be either numeric or symbolic. For each object O_i in the knowledge base the classifier generates a tuple as follows:

$$\langle O_i.a_1, \ldots, O_i.a_n, C_j, \nu, sim, action \rangle$$

where the notation $O_i.a_l$ stands for the value that the object O_i takes in the attribute a_l; C_j is the classification of O_i using the majority rule; ν is the number of examples used by k-NN to reach the solution; *sim* is the maximum similarity among O_i and the ν examples; and *action* is either *collaboration* or *no-collaboration*. When O_i is solved correctly, the action is *no-collaboration* (meaning that the classifier is able to solve correctly the problem O_i with its own knowledge), otherwise the action is *collaboration*, (meaning that the classifier is not able to solve correctly the problem O_i with its own knowledge).

From the set of all these tuples, the classifier constructs a decision tree that allows to learn preferences about two situations: *a*) when to collaborate with other classifiers, and *b*) when the classifier prefers its own solution. This approach is similar to the one proposed in [8] where the authors use a decision tree to compute the confidence degree on the classification given for an object. In [8] the examples used to construct the decision tree are tuples of three elements,

Table 1. Similarities between the objects p_1 and p_2 and each one of the objects in the knowledge base.

object1	object2	similarity	object1	object2	similarity
p_1	O_1	0.67	p_2	O_1	0.91
p_1	O_2	0.85	p_2	O_2	0.89
p_1	O_3	0.74	p_2	O_3	0.80
p_1	O_4	0.90	p_2	O_4	0.78

therefore the tree has as maximum three levels meaning that the leaves have elements belonging to both classes. In our approach, the tuples we use to construct the decision three have $n + 3$ components (the n attributes plus ν, *sim* and *action*). Also, the preferences we obtain are (or may be) in terms of some of the attributes describing the objects.

2.1 Similarity between objects

The k-NN algorithm uses a similarity measure to retrieve k objects that are the more similar ones to a given object O_i. The most common similarity measures used in k-NN when objects are represented as a set of attribute-value pairs and the values are numerical is the Euclidean distance (although other measures are also used, see for instance [7]). When the values of the attributes are symbolic, the most common similarity measure is the following one:

$$sim(O_i.a_l, O_j.a_l) = \begin{cases} 1, & \text{if} \quad O_i.a_l = O_j.a_l \\ 0, & \text{otherwise} \end{cases}$$

Therefore the similarity between objects O_i and O_j is computed as follows:

$$sim(O_i, O_j) = \frac{\sum_{l=1}^{n} sim(O_i.a_l, O_j.a_l)}{n}$$

in other words, the similarity of both objects is the number of attributes taking the same value in both objects and normalized by the number of attributes describing the objects.

2.2 Retrieving a subset of similar objects

Commonly, the k-NN algorithm retrieves k objects similar to a given one. However what we propose is to retrieve *all* objects with similarity equal or bigger than a given threshold of similarity h. When a problem p_i is given to the classifier, h is the threshold under which the objects are not considered similar enough to p_i and they are rejected. This means that the number of retrieved objects may be different for each input problem. The closer to 1 h is, the more confident is the classification. For instance, let us suppose a knowledge base containing the objects O_1, O_2, O_3 and O_4, and the problems p_1 and p_2 to be classified. Table 1 shows the similarity between p_1 and p_2 and the objects of the knowledge base.

If we take $h = 0.80$, when solving p_1 the classifier retrieves $\nu = 2$ similar objects (only O_2 and O_4 have similarity equal or greater than h); however, when solving p_2 the classifier retrieves $\nu = 3$ similar objects (only O_4 has similarity lower than h).

Let $\mathcal{P}$ be the set of objects in the knowledge base whose similarity to p_i is greater or equal than h. There are four possible situations concerning the elements of $\mathcal{P}$:

1. For *all* object $o_i \in \mathcal{P}$ the solution class is the same. If the class is the correct one, then there is a situation of *no collaboration*, otherwise the algorithm has retrieved similar objects but the classification proposed is not the correct one. These cases are specially useful since they belong to a region where the classes are similar. Therefore, when the problem to be solved belongs to these regions the classifier has to prefer to collaborate with other classifiers since its own classification may be incorrect.
2. The *majority* of the objects in $\mathcal{P}$ belongs to the same class. In this situation the classifier has to collaborate when the majority class is not the correct one. Both *sim* and ν give an idea of how strong is this classification.
3. There is a tie between two solution classes, i.e., there are two classes with the same number of elements in $\mathcal{P}$. This is a situation of collaboration because the classifier has not enough information for classifying p_i.
4. The algorithm does not retrieve any object, meaning that there are not objects in the knowledge base similar enough to the new problem to be solved. In this situation, the classifier also prefers to collaborate with others than give its own classification based on objects that are not similar enough.

3 Learning Preferences for Collaboration

From the procedure described in the previous section, the classifier acquires a set of tuples describing situations for *collaboration/no collaboration*. We assume that only when a problem has been solved correctly, it gives a situation of no collaboration, otherwise the classifier should collaborate. The tuples can be used to construct a decision tree with the goal to induce a general model of collaboration.

As we already have explained in Section 2, the tuples have the form

$$\langle O_i.a_1, \ldots, O_i.a_n, C_j, \nu, sim, action\rangle.$$

Notice that, because we assumed that all the objects are described using the same set of attributes $\mathcal{A}$, all the tuples have the same length. For convenience, we suppose that there are not attributes with unknown values. However, when an object has *unknown* value in an attribute (say $O_i.a_j$) we can take the option that the corresponding position of the tuple (i.e., the position j) will hold the value *unknown*. In other words, *unknown* is plays the same role than any other value. Let us analyze these elements in more detail. The first n elements of the tuple, $O_i.a_1, \ldots, O_i.a_n$, are the values that the object O_i takes in each one of its attributes.

The element C_j of the tuple is the class to which belong the majority of elements in $\mathcal{P}$. Therefore the tuple contains the class to which O_i is classified using the k-NN algorithm with the majority rule. When there is a tie between two classes then we consider that $C_j = \emptyset$.

The element ν of the tuple is the cardinality of $\mathcal{P}$, i.e., the number of elements in the knowledge base that have a similarity greater or equal than the given threshold h. The number ν is related with the threshold h and gives information about the knowledge base. For instance, if h has to be low in order to obtain $\nu \neq 0$, this means that the object O_i is not much similar to any of the elements in the knowledge base.

The element *sim* is the maximum similarity of O_i and the objects of the knowledge base. Although we give a threshold, it is possible that the object O_i has the highest similarity with some of the objects in the knowledge base. We want to take into account this fact, especially when the classification has been incorrect. Notice that these cases (high similarity and incorrect classification) mean that the knowledge base has not enough objects to clearly distinguish the classes involved. In the example shown in Table 1 taking $h = 0.85$, the maximum similarity of the objects retrieved when solving o_1 is 0.90 and when solving o_2 is 0.91.

The element *action* plays the role of class label. It can take two values: *collaborate* or *no collaborate.* As we have already mentioned, the classifier will prefer to collaborate when the classification reached for an object has been either incorrect or a tie. Also it prefers to collaborate when there are no objects in the knowledge base similar enough to the problem O_i.

From the set $\mathcal{T}$ of tuples obtained from the procedure described above, we propose to construct a decision tree to induce rules describing preferences of collaboration/no collaboration. In the next section we describe in some detail the process of construction of a decision tree.

3.1 Construction of Decision Trees

A *Decision Tree* (DT) is a kind of directed acyclic graph in the form of a tree. The root of the tree has not incoming edges and the remaining ones have exactly one incoming edge. Nodes without outgoing edges are called *leaf* nodes and the remaining ones are *internal* nodes. A DT is commonly used to create a domain model predictive enough to classify future unseen domain objects.

The construction of a decision tree is performed by splitting the source set of examples. This process is repeated on each derived subset in a recursive manner called *recursive partitioning.* Figure 1 shows the algorithm (for more details see [14, 15]) commonly used to construct decision trees. It is assumed that domain objects are represented by means of a set of pairs attribute-value. For instance, a mushroom can be described using the attributes texture, form and, size and each one of these attributes can take a value. Therefore, the following is a description of a particular mushroom using a set of attribute-values:

$$((\mathsf{texture} = \mathit{spots})(\mathsf{form} = \mathit{planar})(\mathsf{size} = \mathit{big}))$$

```
ID3 (examples, attributes)
  create a node
  if all examples belong to the same class return class as the label for the node
  otherwise
      A ← best attribute
      for each possible value $v_i$ of A
          add a new tree branch below node
          $examples_{vi}$ ← subset of examples such that A = $v_i$
          ID3($examples_{vi}$, attributes - {A})
return node
```

Fig. 1. Algorithm for growing a decision tree.

The values of the attributes may be continuous-valued or categorical. The description of the mushroom above is categorical (i.e., the values of the attributes are labels). Examples of continuous-valued attributes are the heigh and the weight of a person. Each tree node represents an attribute a_i selected by some criteria and each arch is followed according to the value of a_i. For instance, Fig. 2 shows an example classifying mushrooms as *eatable* or *poisonous*. Attributes describing a mushroom are texture, form and, size. The most relevant attribute for classifying a mushroom is texture since if it is *smooth* the mushroom can be classified as eatable. Otherwise the node has to be expanded. Next relevant attribute is form with two possible values: *planar* corresponding only to poisonous mushrooms; and *round* that is a characteristic shared by both classes of mushrooms. Finally, the attribute size allows a perfect classification of all the known mushrooms.

Each node of a tree has associated a set of examples that are those satisfying the path from the root to that node. For instance, the node size of the tree shown in Fig. 2 has associated all the examples having texture = *spots* and form = *round*.

From a decision tree we can extract rules giving descriptions of classes. For instance, some eatable mushrooms are described by means of the rule:

if texture=*spots* and form=*round* and size=*small* *then* eatable.

A key issue of the construction of decision trees is the selection of the most relevant attribute to split a node. This selection is made by means of a distance measure. Each measure uses a different criteria, therefore the selected attribute could be different depending on it and, thus the whole tree could also be different. The most common measures are based on the *degree of impurity* of a node. That is to say, they compute the proportion of examples of each class contained in a node. The goal is to obtain nodes (the leaves of the tree) having examples of only one class, that is to say, with impurity zero. Intermediate nodes are more pure as closer they are to the leaves, meaning that they are able to differentiate the classes.

Impurity measures compare the impurity of a node, say t, with the impurity of the children nodes $t_1 \dots t_k$ generated by an attibute a_i. This comparison is done for each one of the attributes used to represent the domain objects. The

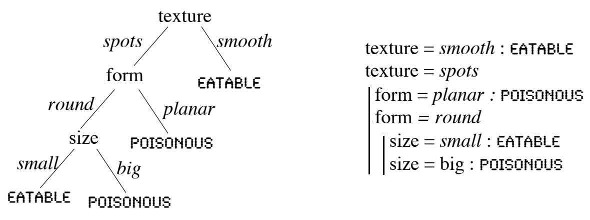

Fig. 2. Example of two different representations of the same decision tree for mushroom classification. These representations are equivalent.

general expression to calculate the *gain* Δ associated to an attribute a_i is the following:

$$\Delta(a_i) = I(t) - \sum_{j=1}^{k} \frac{N(t_j)}{N} \cdot I(t_j)$$

where $I(\cdot)$ is an impurity measure, N is the total number of examples associated to the parent node t, k is the number of different values taken by a_i and $N(t_j)$ is the number of examples associated with the child node t_j.

3.2 Example

We have performed some preliminary experiments using the procedure described in previous sections on the data set *Soybean* from the UCI Machine Learning Repository [2]. The Soybean dataset contains around 300 domain objects distributed on 18 solution classes and described by means of 35 categorical attributes without unknown values.

The first step of our approach is to generate a model of the classifier capabilities using the *leaf-one-out method.* For each object $\langle O_i, C_i \rangle$ in the knowledge base the leave-one-out process is an evaluation technique, commonly used in machine learning, with the following procedure:

1. Take only the description O_i.
2. Use the classifier to achieve a classification for O_i using the remaining objects of the knowledge base.
3. Let C_j be the classification proposed for O_i. If $C_j = C_i$ then the classification for the object O_i is correct; otherwise the classification is incorrect.

In our approach, when an object O_i is classified correctly, it is labelled as belonging to the class *no collaboration* otherwise it is classified as *collaboration.* For instance, a tuple generated in this process is the following:

$$\langle \textit{April,LT-normal,GT-normal,no-hail,}\ldots\textit{,normal,}\ 11,\ 0.88,\ \textit{collaboration} \rangle \quad (1)$$

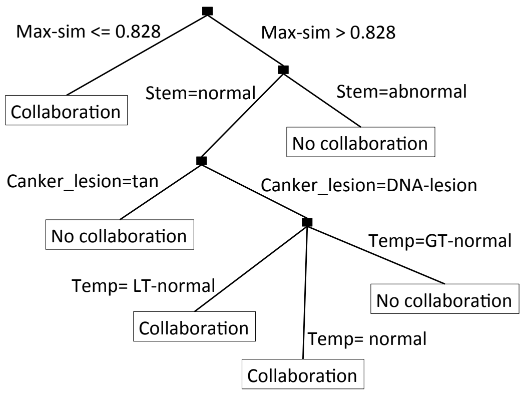

Fig. 3. Example of decision tree characterizing the preferences of collaboration for a classifier on the *Soybean* dataset.

where the first part is composed of 38 values corresponding to the 38 attributes of the description of the domain objects, and the three last values indicate that: 1) the classifier has based its classification on 11 objects of the knowledge base, 2) the maximum similarity between the new object and the most similar retrieved object is 0.88, and 3) the classification has been incorrect, therefore the classifier has labelled the object as *collaboration*.

Since the knowledge base has 300 objects, we have obtained 300 tuples as the one shown in (1) at the end of the leave-one-out process. These 300 tuples have been given as input to a decision tree. We used the J48 algorithm [3] implemented in Weka [6] to generate a decision tree. The J48 algorithm is, in fact, the ID3 algorithm proposed by Quinlan [15] evolved to be able to deal with both categorical and continuous attributes without previous discretization.

We have experimented with different similarity thresholds. The decision tree shown in Fig. 3 is the one corresponding to the threshold of similarity $h = 0.80$. This tree shows that the classifier prefers to collaborate when: 1) the most similar object is under 0.828 (in fact between 0.80 and 0.82 since h is the lower threshold given as input) or, 2) when the similarity is higher than 0.828 and the object to be classified has Stem= *normal*, canker_lesion=*DNA-lesion* and either Temp=*LT_normal* or Temp=*normal*. That is, in addition to the similarity, the description of the object is also taken into account to decide when to collaborate.

4 Future Work

The work introduced in the current paper opens several interesting lines of future research. First of all, we plan to integrate a classifier as the one described in this paper into a system formed by n other classifiers. Each classifier forming the system is capable to completely solve a problem on a given domain and use objects described by means of a common representation (i.e., with the same set of attributes). Moreover, each classifier has a model of its own capability in solving a problem. The general idea is that when one of the classifiers, say Cl_k has to solve a problem p, the first step is to use the tree of preferences in order to detect if Cl_k is capable to solve p using its own knowledge. If the model labels p as *collaboration* Cl_k will ask all other classifiers for collaboration. The easy case is to assume that all the classifiers propose a class and that the classification for p is the class proposed by the majority of the classifiers. This approach should be similar to the one used in ensemble learning [11]. More complicated cases could occur when only a few (or any) of the rest of classifiers are able to propose a classification for p. These cases should be analyzed in detail. We also plan to deeply evaluate a system as the one described above in order to check its accuracy. In particular, we expect that solving problems in collaboration between classifiers produces higher accuracy than having only one.

Another interesting issue is that each classifier has a model of the capabilities of each one of the classifiers of the system. This model could be constructed in the same way described in the paper and its utility could be twofold: 1) a classifier should knowns in advance which classifier will most probably give a correct classification and, consequently, 2) it will not be necessary ask all the classifiers in the system. The second issue is especially interesting when the system is formed by a high number of classifiers since a filtering like this will reduce the communication load between the classifiers.

A third line of research is that each classifier gives, in addition to the classification for p, the confidence in that classification. Such confidence could be obtained from the parameters ν and *sim* of the tuple. High values of both ν and *sim* mean that p has been classified taking into account many known examples having all them high similarity with p, therefore the classification is highly confident. In such situation, the final classification for p could be obtained by means of a weighted aggregation of the classifications proposed by the system classifiers. This same confidence could also be used by a classifier to assess its own capability in classifying p.

Acknowledgements

The author thanks Àngel García-Cerdaña, Pilar Dellunde and the anonymous reviewers their helpful comments and suggestions. The author also acknowledges support by the Spanish MICINN projects EdeTRI (TIN2012-39348-C02-01) and COGNITIO (TIN2012-38450-C03-03) and the grant 2014SGR-118 from the Generalitat de Catalunya.

References

1. E. Armengol and E. Puertas. Learning from cooperation using justifications. In M. Polit, T. Talbert, B. López, and J. Melendez, editors, *Artificial Intelligence Research and Development*, pages 47–54. IOS Press, 2006.

2. A. Asuncion and D. Newman. UCI machine learning repository, 2007.
3. N. Bhargava, G. Sharma, R. Bhargava, and M. Mathuria. Decision tree analysis on J48 algorithm for data mining. *International Journal of Advanced Research in Computer Science and Software Engineering*, 3(6):1114–1119, June 2013.
4. B. V. Dasarathy. *Handbook of Data Mining and Knowledge Discovery*, chapter Data Mining Tasks and Methods: Classification: Nearest-neighbor Approaches, pages 288–298. Oxford University Press, Inc., New York, NY, USA, 2002.
5. D. Dasarathy. *Nearest Neighbor Norms: NN Pattern Classification Techniques.* IEEE Press, 1991.
6. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The Weka data mining software: An update. *SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
7. T. W. Liao, Z. Zhang, and C. Mount. Similarity measures for retrieval in case-based reasoning systems. *Applied Artificial Intelligence*, 12(4):267–288, 1998.
8. S. Ontañón and E. Plaza. Learning when to collaborate among learning agents. In L. D. Raedt and P. A. Flach, editors, *ECML*, volume 2167 of *Lecture Notes in Computer Science*, pages 394–405. Springer, 2001.
9. M. P. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In R. J. Mammone, editor, *Neural Networks for Speech and Image Processing*, pages 126–142. Chapman-Hall, 1993.
10. E. Plaza and S. Ontañón. Ensemble case-based reasoning: Colaboration policies for multiagent cooperative CBR. In I. Watson and Q. Yang, editors, *CBR Research and Development: ICCBR-2001*, volume 2080, pages 437–451, 2001.
11. A. Prodromidis, P. Chan, and S. Stolfo. Meta-learning in distributed data mining systems: Issues and approaches. In H. Kargupta and P. Chan, editors, *Book on Advances of Distributed Data Mining.* AAAI press, 2000.
12. F. J. Provost and D. N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the 13th AAAI/IAAI, Volume 1*, pages 74–79, 1996.
13. E. Puertas and E. Armengol. Inducing domain theory from problem solving in a multi-agent system. In J. Vitrià, P. Radeva, and I. Aguiló, editors, *Recent Advances in Artificial Intelligence Research and Development*, pages 325–332. IOS Press, 2004.
14. J. Quinlan. Discovering rules by induction from large collection of examples. In *Expert Systems in the Microelectronic Age. D. Michie (Ed.)*, pages 168–201. Edimburg University Press, 1979.
15. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.