Seeing the Forest through the Trees
Learning a Comprehensible Model from a First Order Ensemble

Anneleen Van Assche and Hendrik Blockeel

Computer Science Department, Katholieke Universiteit Leuven, Belgium
{anneleen.vanassche,hendrik.blockeel}@cs.kuleuven.be

Abstract. Ensemble methods are popular learning methods that are usually able to increase the predictive accuracy of a classifier. On the other hand, this comes at the cost of interpretability, and insight in the decision process of an ensemble is hard to obtain. This is a major reason why ensemble methods have not been extensively used in the setting of inductive logic programming. In this paper we aim to overcome this issue of comprehensibility by learning a single first order interpretable model that approximates the first order ensemble. The new model is obtained by exploiting the class distributions predicted by the ensemble. These are employed to compute heuristics for deciding which tests are to be used in the new model. As such we obtain a model that is able to give insight in the decision process of the ensemble, while being more accurate than the single model directly learned on the data.
Key words: ensembles, first order decision trees, comprehensibility

1 Introduction

In the process of knowledge discovery, one seeks to extract useful knowledge from data bases. But for knowledge to be useful, predictive accuracy is not sufficient: the extracted patterns also need to be understood by human users in order to trust them and accept them. Moreover users often construct models to gain insight in the problem domain rather than to obtain an accurate classifier only. For this reason, researchers have advocated for machine learning methods, such as decision tree learners and rule learners which yield comprehensible models. In the context of inductive logic programming, comprehensibility is usually even more important than in propositional learning because in the problem domains tackled by ILP algorithms (such as life science, medical domains etc.) end-user acceptance often depends on the learners ability to explain the reasoning behind its decisions.

For quite some years now, a lot of interest has been shown to a class of learning methods called ensembles. The main goal in the design of ensemble methods is to increase the predictive accuracy of the classifier and studies indeed have shown the discrimination power of ensemble methods both theoretically and empirically [1, 4, 7], and in propositional as well as relational learning [12, 10, 6, 18]. Ensemble methods are learning algorithms that first construct a set

of classification models and then classify new data points by combining the predictions of each of these models. Exactly by doing so, they are often able to increase the stability and predictive accuracy significantly over the single models. On the other hand the comprehensibility of the learned hypothesis drops significantly, since the result of an ensemble method is a large set of models with (weighted) votes connected to each of them, which obviously becomes very hard to interpret. This is one of the major reasons why ensemble methods are still less popular in ILP than in propositional learning.

Some authors have pointed out that striving for comprehensibility is one of the important issues in ensemble learning requiring future investigations [1, 15]. Quite some successful research has been carried out already in this area. More in particular researchers have tried to obtain comprehensibility by means of extracting a new interpretable model from an existing ensemble without sacrificing too much accuracy. Craven and Shavlik [5] presented an algorithm TREPAN for extracting comprehensible, symbolic representations from trained neural networks. TREPAN extracts a decision tree using the network as an oracle to answer queries during the extraction process. Domingos [8] proposed Combined Multiple Models (CMM). CMM first builds an ensemble of multiple models and then reapplies the base learner to recover the partitioning implicit in the multiple model ensemble. This is achieved by giving the base learner a new training set, composed of a large number of examples generated and classified according to the ensemble. Zhou et al. [19] utilize neural network ensembles to generate new instances and then extract symbolic rules from those instances. Ferri et al. [9] describe a method to learn a comprehensible model from an ensemble by selecting the single hypothesis from a multi-tree that is most similar to the combined hypothesis according to a similarity measure. In the context of relation learning, Van Assche et al. [17] proposed a method similar to Domingos' to learn a new interpretable model from artificially generated relational data based on a first order ensemble.

The approaches described above all rely on artificially generated data to tackle the problem of finding an interpretable model that approximates the ensemble: either by classifying this new data by the ensemble and constructing a new interpretable model on it, or by using this new data to measure similarity between the ensemble model and candidate interpretable models. As was described in [17] generating relational artificial data is not straightforward (contrary to propositional data), because data distributions are far more complex and examples do not longer have a fixed set of attributes. In [17] new 'partial' examples are constructed by adding more and more constraints (that occur as tests in a tree of the ensemble) to the example. Before a constraint is added, satisfiability needs to be checked with respect to already added constraints. Therefore, one needs to make use of the inherent equivalence relation between a defined concept and its definition. The algorithm relies on the users ability to adequately define this background knowledge in both directions of the equivalence relation. But in Prolog, this becomes almost unfeasible when introducing aggregate or other complex functions.

For this reason, in this paper we aim to learn a single interpretable model from a first order ensemble without the need of generating artificial data nor requiring extra input from the user. Instead of first generating artificial data and computing class distributions for different possible tests on this data, class distributions are estimated directly from the information available from the different models in the ensemble in order to decide which tests are to be used in the new model. We describe the proposed approach in detail in the next section.

2 Proposed Method

In the remainder of the paper we will propose a method to learn a single first order decision tree from a first order decision tree ensemble that is constructed via bagging.

2.1 Constructing a Single Tree Exactly Representing an Ensemble

Assume E is an ensemble of N first order decision trees, which we would like to represent by one single first order decision tree. The ensemble E gives labels $L_E(x)$ to new examples x according to a combination of the predictions of each of its N base trees, in this case the class C_i with the highest average predicted probability:

$$L_E(x) = argmax_{C_i}(\frac{1}{N}\sum_{k}^{N} P_k(C_i|x)) \tag{1}$$

Actually a decision tree is able to represent any function described over the instance space as it can separate the instance space completely if necessary, so there also exist a decision tree that exactly represents the function described by the ensemble (but it cannot necessarily be learned from the training data). Depending on the order of the tests in the nodes, the tree that represents the same concept as the ensemble, can consist of up to 2^d leaves, with d the number of different tests in the ensemble. So although representing the same concept as the ensemble, such a tree would not satisfy our needs, namely be interpretable and give insight in the ensemble's decisions, simply because it is too big. Very often even the smallest tree exactly representing the concept in the ensemble might be far too large to be considered interpretable and an approximation to the ensemble will then be preferred over the exact representation of the ensemble.

2.2 Computing Heuristics from the Ensemble

To construct a first order decision tree from an ensemble we will closely follow the procedure of regular first order tree induction according to TILDE [2] as shown in Table 1. In TILDE, first order decision trees are learned with a divide and conquer algorithm similar to C4.5 [14]. The OPTIMAL_SPLIT procedure returns a query Q_b, which is selected from a set of candidates generated by the refinement operator ρ, by using a heuristic, such as information gain for classification problems, or variance reduction for regression. The refinement operator typically operates

```
procedure GROW_TREE (E: examples, Q: query):
    candidates := ρ(← Q)
    ← Q_b := OPTIMAL_SPLIT(candidates, E)
    if STOP_CRIT (←Q_b, E)
    then
        K := PREDICT(E)
        return leaf(K)
    else
        conj := Q_b − Q
        E_1 := {e ∈ E|←Q_b succeeds in e ∧ B}
        E_2 := {e ∈ E|←Q_b fails in e ∧ B}
        left := GROW_TREE (E_1, Q_b)
        right := GROW_TREE (E_2, Q)
        return node(conj, left, right)
```

Table 1. TILDE algorithm for first order logical decision tree induction [2].

under θ-subsumption and generates candidates by extending the current query Q (the conjunction of all succeeding tests from the root to the leaf that is to be extended) with a number of new literals that are specified in the language bias.

In order to construct a tree that models the ensemble, in each node of the tree the optimal split needs to be chosen based on the ensemble instead of the data. So we need to adapt the OPTIMAL_SPLIT procedure such that the heuristic used (usually information gain or gain ratio) can be computed from the distributions in the ensemble.

Let's assume the heuristic is information gain and we want to compute the optimal split according to the ensemble in a certain node n of the new tree. Suppose B is the conjunction of tests that occurred along the path from the root until node n, then the information gain IG for a certain test T in n is

$$\begin{aligned} IG(T|B) = entropy(B) \quad & -P(T|B)entropy(T,B) \\ & -P(\neg T|B)entropy(\neg T,B) \end{aligned} \tag{2}$$

where

$$entropy(A) = \sum_{i=1}^{c} -P(C_i|A)\log_2 P(C_i|A) \tag{3}$$

with c the total number of classes and C_i the ith class and A any set of conditions. These are the regular formula's for information gain and entropy. But now the distributions needed to calculate these heuristics will be estimated from the ensemble instead of the data.

A decision tree is constructed to model class distributions in the data and thus can be used to estimate these $P(C_i|A)$. Suppose we have a decision tree DT_k in the ensemble E, we can now estimate this $P_k(C_i|A)$ by propagating it through the tree DT_k, applying the law of total probability in each node, until

we end up in the leaves. Then we get:

$$P_k(C_i|A) = \sum_{\text{leaves } l_{kj} \text{ in } DT_k} P(C_i|Y_{kj}, A)P(Y_{kj}|A) \quad (4)$$

where Y_{kj} is the conjunction of tests from the root of tree DT_k until leaf l_{kj}. The class probability estimate $P_E(C_i|A)$ of the ensemble E is then the average over the class probability estimates $P_k(C_i|A)$ of the N trees in E.

In the equation 4 there is one term for each leaf in the tree. Now the probability $P(C_i|Y_{kj}, A)$ corresponds to the probability estimate $P_k(C_i|Y_{kj})$ given by leaf l_{kj}. Indeed, either $A \leftarrow Y_{kj}$ and $P(C_i|Y_{kj}) = P(C_i|Y_{kj}, A)$ or , $A \not\leftarrow Y_{kj}$ and then we can assume that the class C_i is conditionally independent from the tests in A given the tests in Y_{kj}, because if not, the leaf l_{kj} would have been split at least once more on a test $T \in A \setminus Y_{kj}$.

For the other probability $P(Y_{kj}|A)$, we can distinguish 3 possible cases:

- $A \models Y_{kj}$: then $P(Y_{kj}|A) = 1$ and $P_k(C_i|A) = P_k(C_i|Y_{kj})$
- $A \models \neg Y_{kj}$: $P(Y_{kj}|A) = 0$ and leaf l_{kj} of tree DT_k will not contribute in the probability $P_k(C_i|A)$
- $A \not\models Y_{kj}, A \not\models \neg Y_{kj}$: $0 < P(Y_{kj}|A) < 1$ and leaf l_{kj} of tree DT_k partly contributes to the probability $P_k(C_i|A)$

To be able to estimate these probabilities $P(Y_{kj}|A)$, we could make the assumption that Y_{kj} is conditionally independent from A, as such $P(Y_{kj}|A) = P(Y_{kj})$. This assumption is exactly the same as made by Quinlan [13], when classifying instances with missing values by a tree. But as decision trees are not specifically constructed to model distributions among the tests, another, maybe better way to estimate the probabilities $P(Y_{kj}|A)$ is by computing them on the training set (if the test set is already available, using both training and test set might provide an even better estimate). The same holds for the $P(T|B)$, as requested in equation 2.

Using the method described above to compute the information gain for tests according to an ensemble E, a decision tree is built representing the ensemble E, each time replacing a leave n, with B the conjunction of tests occurring on the path from the root to leaf n, in the tree by an internal node as long as we can find a test T where $IG_E(T|B) \geq IG_E(T_i|B)$ for all possible tests T_i and $IG_E(T|B) > 0$. On the other hand if $IG_E(T_i|B) = 0$ for all tests T_i, all examples ending up in n will be labeled the same by the ensemble, and no further splitting is required to represent the ensemble.

2.3 Generation of Candidate Test Queries

In the normal first order decision tree induction algorithm described in Table 1, candidate tests are generated by extending the current query Q using a refinement operator ρ which operates under θ-subsumption. On the other hand, in order to represent the hypothesis of the ensemble by a single tree, it is theoretically sufficient to use the tests that were used in the ensemble. And also

intuitively, it makes sense only to use these tests to construct a new tree as these were probably the most important amongst all possible tests. In a first order decision tree, the tests in the internal nodes of the tree are conjunctions of first order literals. So to select the possible tests to construct the new tree, for each leaf in a tree of the forest, the conjunction of positive first order literals occurring from the root until that leaf is considered. Then this conjunction is split into separate tests by taking literals together that share variables. As such we get a fixed set of tests which can be used to put in the internal nodes of the new tree. This is in contrast with usual ILP hypothesis construction where the search space grows as new literals (that may introduce new variables) are added to the hypothesis. By using this feature selection step, we can construct a new tree efficiently even allowing some kind of lookahead as conjunctions of literals might be added at once.

2.4 Stop Criteria

The tree obtained using the method described above, represents the ensemble but is often very large and as a consequence incomprehensible, also constructing it will be very time consumable. For that reason, it will be necessary to impose some stop criteria to avoid the tree from becoming too large. First we describe a way to do safe prepruning in order to avoid splits to be added that will not have an influence on the predicted class. This will not change the eventual predictions of the tree. Next, we will impose a none equivalence preserving stop criterion, to make the tree more interpretable for the end user.

Safe prepruning At the end of the tree construction, some redundant splits will still be present in the tree, as they might change the class distributions in the leaves but not the eventual classes predicted by the leaves. Usually in decision tree algorithms, these nodes are removed by postpruning. The same can be applied here, but since the tree deduced from an ensemble usually becomes rather large, quite some time might be spend in adding all these redundant splits and it would be desirable to preprune the tree if possible.

Although according to the class distributions in the ensemble there still exist splits that are able to increase the pureness of the distribution in the current node, it makes no sense to split the tree further if all (possible) examples that end up in the current node are labeled the same by the ensemble. We will check whether indeed all examples that end up in a node n will be predicted the same class by the ensemble as follows. Examples that end up in n are examples that succeed D_n, where D_n is the conjunction of tests along the path from the root until node n. For each tree DT_k of the ensemble we keep track of the possible leaves l_{kj} in that tree where these examples fulfilling D_n might end up. Each of these leaves l_{kj} gives a prediction of the probability of a class $P(C_i|Y_{kj})$, with Y_{kj} the tests along the path from the root of tree k to leaf l_{kj} Then we can define a lower bound on the class probability prediction of the ensemble E for examples fulfilling D_n as follows:

$$P_{Emin}(C_i|D_n) = \frac{1}{N}\sum_k \min_{l_{kj}^{D_n}} P(C_i|Y_{kj})$$

and equivalently an upper bound:

$$P_{Emax}(C_i|D_n) = \frac{1}{N}\sum_k \max_{l_{kj}^{D_n}} P(C_i|Y_{kj})$$

where $l_{kj}^{D_n}$ are all the leaves in DT_k where examples satisfying D_n can possibly end up. Then if

$$\exists C \in \mathcal{C}, \forall C_i \in \mathcal{C} \setminus \{C\} : P_{Emin}(C|D_n) > P_{Emax}(C_i|D_n)$$

where $\mathcal{C}$ are all possible class values, all examples in n will be predicted class value C by the ensemble E, and the tree should not be split any further in that node.

Other stop criterion As mentioned before, the tree constructed using safe prepruning, might still be very large. Because of this, we will introduce another stop criterion, such that a more comprehensible approximation to the ensemble is obtained. If a node is pure according to the training data, no further splitting will be performed and the node will become a leaf.

2.5 Practical Implementation in First Order System

The method described above was implemented in the ACE-ilProlog system [3]. This system contains a first order decision tree learner TILDE [2], and some ensemble methods such as bagging and random forests that use TILDE as the base-learner.

As detailed in Section 2.2, each of the preselected candidate tests, are assigned a heuristic value, by propagating them through the ensemble trees, and using the probability estimates of the leaves they end up in. To find the probability that a certain test ends up in a certain leaf, we check the proportion of examples, covered by the leaf, that is also covered by the test. In the implementation, this is done by keeping track of precomputed coverlists for all leaves and candidate tests. These coverlists store for all available examples whether the examples are covered by the corresponding test or not. Deciding which tests end up in which leaves is then simply a matter of taking the intersection of their coverlists, and as such this avoids normal satisfiability testing between queries as was applied in Van Assche et al. [17].

3 Experiments

We performed some preliminary experiments both on a train data set [11] generated with the Random Train Generator from Muggleton[1] according to a specified

[1] The train generator is available at
http://www-users-csyork. ac.uk/∼stephen/progol.html.

	Accuracy			Model size		
	3	11	33	3	11	33
Trains data						
Bagging	0.675	0.707	0.715	136.94	511.76	1529.22
ism(Bagging)	0.683	0.701	0.704	77.82	85.54	82.22
Forf(0,25)	0.673	0.708	0.718	128.8	471.38	1416.42
ism(Forf(0,25))	0.654	0.705	0.707	84.32	76.76	71.36
Tilde		0.689			40.6	
Carcinogenesis data						
Bagging	0.592	0.616	0.620	105.88	389.96	1176.6
ism(Bagging)	0.603	0.619	0.631	80.08	79.3	80.48
Forf(0.25)	0.604	0.623	0.620	104.34	379.51	1137.97
ism(Forf(0.25))	0.601	0.621	0.623	80.64	77.82	73.82
Tilde		0.611			29.88	

Table 2. Accuracy and complexity on both data sets for Bagging and Forf(0.25) with 3-11-33 trees, the ism model that was learned from these ensembles and Tilde.

concept discussed in Van Assche et al. [18], as on the Carcinogenesis data set [16]. We constructed a bagged ensemble with 3, 11 and 33 trees, as well as a random forest where 25% of the features were used at each node. From these ensembles a interpretable single model (ism) was constructed using the method described above. For comparison, we also build a single (pruned) tree directly on the training data. All experiments were done averaging over 10 different 5-fold cross-validations. In table 2 we report testing accuracy and model size in terms of number of nodes in the trees for the different settings. The results show that the method is able to obtain a single model with an accuracy comparable to the ensemble it is learned from, while reducing the complexity substantially.

4 Conclusions and Future Work

In this paper we presented a method to learn a first order decision tree that approximates the decisions made by an ensemble of first order decision trees. The tree is obtained without the need of generating artificial data nor requiring extra input from the user. Instead, first a fixed set of possible candidate tests for the nodes in the new tree are extracted from the ensemble. Next, heuristics are computed for each of the candidate tests by predicting class distributions for these tests using the ensemble. Labels in the eventual leaves of the new tree are the labels predicted by the ensemble. As such, we aim to obtain an interpretable tree that is able to give insight in the predictions of the ensemble, while being more accurate than a single tree directly learned on the data. First experiments are promising but more extensive experiments need to reveal how rewarding this method is compared to learning a tree directly from the data. Currently also no postpruning is performed on the obtained trees, and as a result they are still larger than the pruned trees learned from the data. We need to investigate what the effect on the accuracy is of applying postpruning and/or stop criteria. Next, we would also like to have a look at the stability of the obtained trees.

Acknowledgements

Anneleen Van Assche is supported by the Institute for the Promotion of Innovation by Science and Technology in Flanders (I.W.T.-Vlaanderen). Hendrik Blockeel is Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium) (F.W.O.-Vlaanderen).

References

1. E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105, 1999.
2. H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.
3. H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.
4. L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
5. M. W. Craven. *Extracting Comprehensible Models from Trained Neural Networks*. PhD thesis, University of Wisconsin, Madison, 2003.
6. I. de Castro Dutra, D. Page, V. Costa, and J. Shavlik. An empirical evalutation of bagging in inductive logic programming. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Computer Science*, pages 48–65, 2002.
7. T. Dietterich. Ensemble methods in machine learning. In *Proceedings of the 1th International Workshop on Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15, 2000.
8. P. Domingos. Knowledge discovery via multiple models. *Intelligent Data Analysis*, 2:187–202, 1998.
9. C. Ferri, J. Hernández-Orallo, and M. Ramrez-Quintana. From Ensemble Methods to Comprehensible Models. In *Proceedings of 5th International Conference on Discovery Science (DS 2002)*, volume 2534 of *Lecture Notes in Computer Science*, pages 165–177, 2002.
10. S. Hoche and S. Wrobel. Relational learning using constrained confidence-rated boosting. In C. Rouveirol and M. Sebag, editors, *Proceedings of the Eleventh International Conference on Inductive Logic Programming*, volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 51–64. Springer-Verlag, September 2001.
11. R. Michalski. Pattern Recognition as Rule-Guided Inductive Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2:349–361, 1980.
12. J. Quinlan. Boosting first-order learning. In *Algorithmic Learning Theory, 7th International Workshop (ALT '96)*, 1996.
13. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
14. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in Machine Learning. Morgan Kaufmann, 1993.
15. G. Ridgeway, D. Madigan, and J. Richardson, T. adn O'Kane. Interpretable boosted naive bayes classification. In *Proceedings of the 4th International Conference on Knowledge Discovery in Databases*, pages 101–104. AAAI Press, 1998.
16. A. Srinivasan, R. King, S. Muggleton, and M. Sternberg. Carcinogenesis predictions using ILP. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, pages 273–287. Springer-Verlag, 1997.

17. A. Van Assche, J. Ramon, and H. Blockeel. Learning interpretable models from an ensemble in ILP. In *Proceedings of the 16th International Conference on Inductive Logic Programming – short papers*, pages 210–212, 2006.
18. A. Van Assche, C. Vens, H. Blockeel, and S. Džeroski. First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64(1-3):149–182, 2006.
19. Z. Zhou, Y. Jiang, and S. Chen. Extracting symbolic rules from trained neural network ensembles. *AI Communications*, 16(1):3–15, 2003.