Top-Down Pruning in Relational Learning

Johannes Fürnkranz

Austrian Research Institute for Artificial Intelligence
Schottengasse 3
A-1010 Vienna
Austria
juffi@ai.univie.ac.at

Content Areas: Machine Learning, Inductive Logic Programming, Pruning

Abstract

Pruning is an effective method for dealing with noise in Machine Learning. Recently pruning algorithms, in particular *Reduced Error Pruning*, have also attracted interest in the field of *Inductive Logic Programming*. However, it has been shown that these methods can be very inefficient, because most of the time is wasted for generating clauses that explain noisy examples and subsequently pruning these clauses. We introduce a new method which searches for good theories in a top-down fashion to get a better starting point for the pruning algorithm. Experiments show that this approach can significantly lower the complexity of the task as well as increase predictive accuracy.

OEFAI-TR-94-03

1 Introduction

Pruning is a standard way of dealing with noise in Machine Learning. In particular in decision tree learning pruning methods proved to be most effective (see e.g. [Mingers, 1989] or [Esposito *et al.*, 1993]). *Pre-pruning* — heuristically deciding when to stop growing clauses and concepts — has been present in *Inductive Logic Programming* (ILP) in the form of *stopping criteria* for quite some time (see e.g. FOIL [Quinlan, 1990], *m*FOIL [Džeroski and Bratko, 1992] and FOSSIL [Fürnkranz, 1994]). The basic idea behind most *post-pruning* methods is to learn a concept description on one part of the training instances and to subsequently delete several parts of this theory in order to improve performance on the remaining set. The most prominent use of this method in ILP is the adaptation of *Reduced Error Pruning* [Brunk and Pazzani, 1991]. However, it has been shown in [Cohen, 1993] that REP can be very inefficient, because most of the time is wasted for generating clauses that explain noisy examples and subsequently pruning these clauses. We solve this problem by adapting the relational learning algorithm FOSSIL to combine pre-pruning and post-pruning by first performing a general-to-specific search for a good starting theory and then pruning this theory.

2 FOSSIL and the Cutoff Stopping Criterion

FOSSIL [Fürnkranz, 1994] is a FOIL-like ILP system that uses a search heuristic based on statistical correlation. Intuitively, the so-called *correlation coefficient* $corr(c, l)$ measures the congruence of the truth values of the instances covered by the partially grown clause c with the truth values assigned to these instances when extending c with the candidate literal l on a scale from -1 to $+1$. Ideally, l would be `true` for all positive instances covered by c and `false` for all negative instances covered by c. In this case $corr(c, l) = +1$. If l is `false` for all negative instances and `true` for all positive instances, then $corr(c, l) = -1$. This means that $\neg l$ perfectly discriminates between positive and negative examples and should be chosen to extend c. If there is no perfect discrimator among the candidate literals, $corr(c, l)$ is computed for all literals l and the one with the maximum value of $|corr(c, l)|$ will be chosen ($\neg l$ if $corr(c, l) < 0$). $corr(c, l) = 0$ suggests that l is equally likely to

cover positive as well as negative instances, which means that l is no good discriminator and should not be used to extend the clause. Literals that are always `true` or always `false` must be treated separately, because $corr(c, l)$ is undefined on these. Currently they are assigned a correlation value of 0.

Previous research has shown that the correlation heuristic works very well in connection with the simple and efficient *cutoff stopping criterion*: the user is able to specify a certain threshold, the *Cutoff*, such that only literals for which $|corr(c, l)| > Cutoff$ holds are considered for extending the partially grown clause c. If no such literal can be found, the clause is considered to be complete and all covered (positive *and* negative examples) will be removed from the training set. If no new clause can be started with a literal above the treshold, learning stops. Experiments have shown that this method is very noise-tolerant and that a good value for the *Cutoff* is independent of the noise level as well as independent of the size of the training set.

A more detailed description of the algorithm can be found in [Fürnkranz, 1994]. The version of FOSSIL used in the experiments reported here is almost identical to the one reported there, except that it has been sped up considerably and that tuple sets are not extended after the introduction of new variables (which amounts to counting covered instances instead of counting the proofs for the covered instances). In addition, modes, types and symmetries can now be declared for the literals in the background knowledge.

3 Top-Down Pruning

3.1 Pruning in Inductive Logic Programming

Pruning is a standard way of dealing with noise in Decision Tree learning (see e.g. [Mingers, 1989] or [Esposito *et al.*, 1993]). There are two fundamentally distinct approaches to *pruning*:

Pre-Pruning means that during concept generation some training examples are deliberately ignored, so that the final concept description does not classify all training instances correctly.

Post-Pruning means that first a concept description is generated that perfectly explains all training instances. This will be subsequently gener-

alized by cutting off branches of the decision tree (as in [Quinlan, 1987] or [Breiman *et al.*, 1984]).

In ILP, Pre-Pruning has been common in the form of *stopping criteria* as used in FOIL [Quinlan, 1990], *m*FOIL [Džeroski and Bratko, 1992], and FOSSIL (see section 2). Post-Pruning was introduced to ILP with an adaptation of Quinlan's *Reduced Error Pruning* [Brunk and Pazzani, 1991]. First the training set is split into two subsets: a *growing set* and a *pruning set*. A concept description explaining all of the examples in the growing set is generated with a relational learning algorithm. The resulting concept is then generalized by deleting literals and clauses from the theory until all possible deletions would result in a decrease of predictive accuracy, measured on the pruning set.

While this method proved to be very effective in avoiding noise-fitting in several domains, [Cohen, 1993] has shown that REP is a very costly process. Its time complexity on random data is as bad as $\Omega(n^2 \log n)$ for generating a concept description from n examples and $\Omega(n^4 \log n)$ for pruning the resulting set of rules. [Cohen, 1993] has then suggested a more efficient pruning method, which only has a worst-case time complexity of $O(n^2 \log n)$. This algorithm was further improved by adding some pre-pruning methods to speed up the concept generation phase.

In the next sections we will show a way to significantly speed up both the growing and the pruning phase by generating a series of concept descriptions from general to specific and selecting an appropriate starting point for subsequent reduced error pruning.

3.2 Generating a series of concept descriptions

An interesting feature of FOSSIL's cutoff stopping criterion (section 2) — besides its efficiency and stability — is its close relation to the search heuristic. While FOIL (*encoding length restriction*) and *m*FOIL (*significance test*) have to do separate calculations to determine when to stop learning, FOSSIL needs to do a mere comparison between the heuristic value of the best candidate literal and the cutoff value. This allows the design of a very simple algorithm that can generate all theories that could be learned by FOSSIL with any setting of the *Cutoff* parameter (see figure 1).

```
Cutoff = 1.0
Concepts = ∅
while (Cutoff > 0.0) do
        NewConcept = FOSSIL(Examples)
        Cutoff = MaxPrunedCorr(NewConcept)
        Concepts = Concepts ∪ NewConcept
return(Concepts)
```

Figure 1: Algorithm to generate all theories learnable by FOSSIL

The basic idea behind the algorithm given in figure 1 is the following: Assume that we are trying to learn a theory with a Cutoff of 1.0. Unless there is one literal in the background knowledge that perfectly discriminates between positive and negative examples, we will not find a literal with a correlation of 1.0 and thus learn an empty theory. During this run we can remember the literal with the maximum correlation. If we now set the new cutoff to exactly this maximum value, at least one literal (the one that produced this maximum correlation) will be added to the theory.

At this new setting of the cutoff parameter a new theory will be learned and again the maximum correlation of the literals that have been cut off will be remembered. Obviously, for all values between the old cutoff and the new maximum, the same theory would have been learned. Thus we can choose this value as the cutoff for the next run. It can also be expected that the new theory will be more specific than the previous one. This process is repeated until at a certain setting of the *Cutoff* no further literal is cut off (i.e. $MaxPrunedCorr = 0.0$) and thus the most specific theory has been reached.

In figure 2 we see an example how FOSSIL generates a series of theories from 1000 noise free examples in the king-rook-king chess endgame domain.[1] It is interesting to see how the algorithm steadily modifies the theory until it arrives at a 99.32% correct theory (evaluated on a test set of 5000 examples), which explains all of the examples in the training set. Note that lowering the cutoff does not necessarily result in a theory with more clauses, because

[1] A short description of the domain can be found at the beginning of section 4.

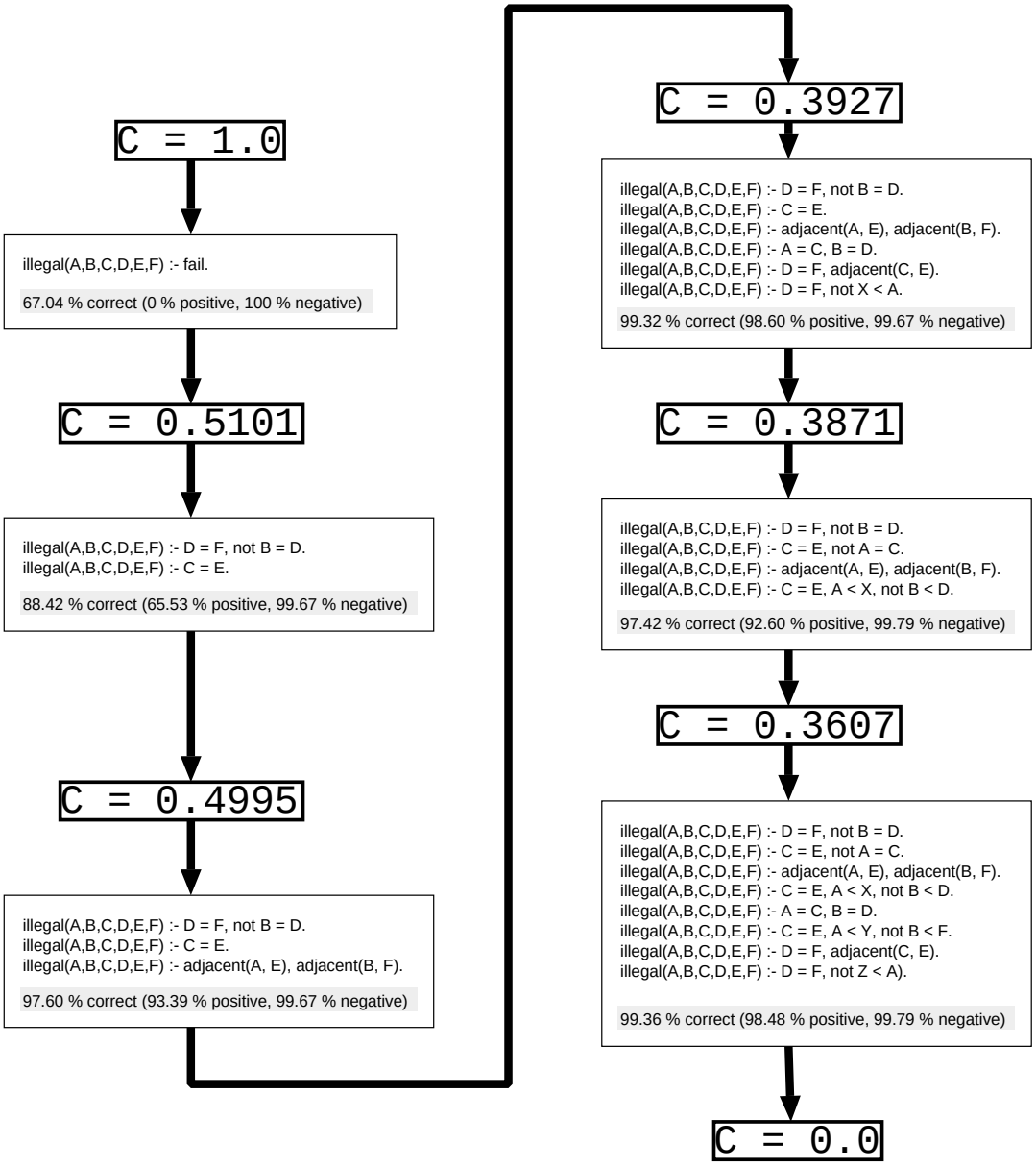

Figure 2: Generating a series of theories in the KRK domain

adding a literal to the current concept definition will change the search space for subsequent literals and thus change the correlation values of the literals at all subsequent choice points.

The next section shows how the simple algorithm of figure 1 can be refined to a powerful algorithm which selects an appropriate starting point for *Reduced Error Pruning*, thus improving over run-time and accuracy of REP.

3.3 Top-Down Pruning

A very nice property of the algorithm of figure 1 is that we get a series of different concept descriptions in a — roughly — general to specific order[2] (*top-down*) as opposed to pruning methods that generate a most specific theory first and then successively generalize it (*bottom-up*). If we could find a way of evaluating the theories as they are generated, and stopping when the generated theories get worse, we could avoid learning the most specific theory, which is usually quite expensive.

Also note that usually several clauses — up to the point where the highest cutoff has occurred — can be reused from the previous run, so that the total cost of generating a series of concept description may not be much more than the cost of generating the most specific theory only.

Based on the above ideas, we have implemented the following simple algorithm. It tries to find the most specific among all reasonably good theories, and subsequently generalizes this theory with *Reduced Error Pruning*. Because of the initial general-to-specific search for a good theory, we have named the method *Top-Down Pruning*.

1. *Split the training set into a* growing set *and a* pruning set *(usually* 2/3 *and* 1/3*).*

2. *Generate a series of concept descriptions from the examples in the growing set.*

3. *While you go, evaluate each theory on the pruning set.*

[2]Note that we use the terms "general" and "specific" in an intuitive way. We consider the empty theory to be most general, because "Everything is false." is a very general statement. However, our "most specific" theory will cover more ground instances than the empty theory, and thus may be considered (extensionally) more general. See [Flach, 1992] for a discussion of related matters.

4. *When the measured accuracy of one the theories falls below the measured accuracy of the best theory so far minus the* Standard Error *for classification*[3]*, stop generating theories and return the last theory within the 1 SE margin.*

5. *Prune the theory obtained in step 4. using* Reduced Error Pruning *as described in [Brunk and Pazzani, 1991].*

We expected this method to be faster than *Reduced Error Pruning*, in particular with high numbers of noisy examples, because it will generate a more general starting theory and thus

- *Speed up the growing phase*, because the most expensive theories will not be generated
- *Speed up the pruning phase*, because pruning starts from a simpler theory and thus the number of possible pruning operations is much smaller.[4]

The next section reports experiments that confirm the above hypotheses.

4 Experiments

We tested the algorithm developed in the last section on the chess king-rook-king endgame domain that has been extensively used in ILP. The setup for the experimental evaluation of *Top-Down Pruning* was the same as described in [Fürnkranz, 1993]. The only difference was that the most recent version of FOSSIL allows to specify modes, types and symmetries of background predicates, and this facility was used in all experiments. Experiments were performed with 10% of the examples having their classification reversed. Testing was done on sets of 5000 noise-free examples. *Reduced Error Pruning* (REP)

[3] This is based on on idea in CART [Breiman *et al.*, 1984], where the most *general* pruned decision tree within one SE of the best will be returned. The standard classification error is defined as $SE = \sqrt{\frac{p \times (1-p)}{N}}$ where p is the probability of mis-classification (estimated on the pruning set) and N is the number of examples in the pruning set.

[4] These considerations, of course, only apply to noisy domains. In non-noisy domains the most specific theory will in general be the most precise and thus our algorithm will be a little slower than REP.

and *Top-Down Pruning* (TDP) were both given the same 10 training sets for each of the 4 different training set sizes. Both algorithms split the sets into the same growing (ca. 2/3) and pruning sets (ca. 1/3). REP generated the most specific theory first ($Cutoff = 0$), and then used the method described in [Brunk and Pazzani, 1991] for pruning. TDP is an implementation of the algorithm decribed in the last section, which made use of all optimizations described there.

Our first concern, of course, is whether the new method does not loose predictive accuracy compared to REP (see table 1).

Average Accuracy		50	100	250	500
REP	Before Pruning	79.28	84.84	86.88	87.11
	After Pruning	**78.60**	**94.67**	**96.72**	**97.80**
TDP	Before Pruning	79.90	89.15	92.30	97.84
	After Pruning	**78.60**	**95.14**	**96.81**	**98.77**

Table 1: Accuracy in the KRK domain with 10% classification noise.

Surprisingly, it not only does not lose, but gains predictive accuracy. The reason for this is that TDP already starts with a much better theory than REP (see below) and thus is less likely to get caught in a local optimum. In particular at training set size 500, REP sometimes returned suboptimal theories. However, REP may profit from this in some rare cases, as it happened in one of the ten sets with 250 training examples: Here TDP started off with a theory that was 98.18% correct, but unfortunately one of the literals had no support in the pruning set and consequently was pruned, thus yielding a theory with a mere 81.34%. This did not happen to REP because it got caught in a 91.36% correct theory, and did not even get to the 98.18% theory.

As we have already mentioned, it becomes obvious from table 1 that TDP's top-down search for a good starting point for pruning helps a lot. TDP's difference between the accuracy of the theories before and after pruning is steadily decreasing with training set size. At size 500 the starting theories for TDP were already slightly better than the pruned theories obtained from REP. The top-down search for a good theory (without pruning) could even yield better results if we did not use the most specific theory

within one standard error of the best theory, but the best itself. However, this could lead to over-generalization. Starting with a more specific theory is better for the subsequent pruning process, which is quite inexpensive, as can be seen from table 2.

Average Run-time (sec.)		50	100	250	500
REP	Growing	1.56	6.66	75.22	397.17
	Pruning	0.35	2.93	91.46	1248.48
	Total	**1.91**	**9.59**	**166.68**	**1645.65**
TDP	Growing	2.11	7.87	48.55	47.18
	Pruning	0.29	1.71	23.50	3.33
	Total	**2.40**	**9.58**	**72.05**	**51.01**

Table 2: Run-time in the KRK domain with 10% classification noise.

Looking at the run-times, it can be seen that with increasing training set sizes, the costs of REP are dominated by the pruning process. This result is consistent with the findings of [Cohen, 1993] (see section 3.1). TDP on the other hand, even manages to decrease run-time with growing training set sizes. The explanation for this surprising result can be found in two reasons:

- With increasing training set sizes the size of the allowed Standard Error decreases (see footnote 3), so that it is less and less likely that the most specific theories are within the error margin.
- The starting theories learned by FOSSIL become increasingly accurate as the training set grows, so that less and less pruning has to be done. This can be seen from the decreasing differences between the accuracies of the theories before and after pruning (table 1).

This supports the two hypotheses stated at the end of the last section.

5 Conclusion

We have shown that the Inductive Logic Programming algorithm FOSSIL allows a top-down generation of a series of theories that can be used to find

a good starting point for a subsequent pruning process. This method — *Top-Down Pruning* — results in a significant speed-up compared to *Reduced Error Pruning* along with a small gain in accuracy. Similar results have been obtained in [Cohen, 1993], but concentrated mostly on lowering the pruning cost only. In our approach, the entire process of TDP may be significantly faster than REP's growing phase alone. However, TDP has not yet been tested as extensively as the methods proposed in [Cohen, 1993].

Another reason for the differences in predictive accuracy observed in table 1 might be that the rules generated by a FOIL-like learning algorithm are order-dependent in the sense that learning a wrong first rule affects the population of training instances for the learning of subsequent rules, because of FOIL's *separate-and-conquer* strategy. This is currently under investigation.

Of course pruning methods in general are subject to the problem that learning general theories in order to avoid overfitting the noise might be inappropriate in some domains [Schaffer, 1993]. In fact it has been observed in [Cohen, 1993] that pruning sometimes leads to a decrease in predictive accuracy. However, there is some evidence that simple rules perform well in many real world domains (or at least in those domains that are commonly used as a test bed for machine learning algorithms) [Holte, 1993], and TDP's general-to-specific search might be a good method in those domains.

Acknowledgements

This research is sponsored by the Austrian *Fonds zur Förderung der Wissenschaftlichen Forschung (FWF)* under grant number P8756-TEC. Financial support for the Austrian Research Institute for Artificial Intelligence is provided by the Austrian Federal Ministry of Science and Research. I would like to thank my colleagues B. Pfahringer and Ch. Holzbaur for help on many improvements of the PROLOG implementation of FOSSIL and G. Widmer for encouraging this research.

References

[Breiman *et al.*, 1984] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth & Brooks, Pacific Grove, CA, 1984.

[Brunk and Pazzani, 1991] Clifford A. Brunk and Michael J. Pazzani. An investigation of noise-tolerant relational concept learning algorithms. In *Proceedings of the 8th International Workshop on Machine Learning*, pages 389–393, Evanston, Illinois, 1991.

[Cohen, 1993] William W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 988–994, Chambery, France, 1993.

[Džeroski and Bratko, 1992] Sašo Džeroski and Ivan Bratko. Handling noise in Inductive Logic Programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, Tokyo, Japan, 1992.

[Esposito *et al.*, 1993] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. Decision tree pruning as a search in the state space. In *Proceedings of the European Conference on Machine Learning*, pages 165–184, Vienna, Austria, 1993. Springer-Verlag.

[Flach, 1992] Peter A. Flach. Generality revisited. In *Logical Approaches to Machine Learning, Workshop Notes of the 10th European Conference on AI*, Vienna, Austria, 1992.

[Fürnkranz, 1993] Johannes Fürnkranz. FOSSIL: A robust relational learner. Technical Report TR-93-28, Austrian Research Institute for Artificial Intelligence, 1993. Extended version.

[Fürnkranz, 1994] Johannes Fürnkranz. FOSSIL: A robust relational learner. In *Proceedings of the European Conference on Machine Learning*, Catania, Italy, 1994. Springer-Verlag.

[Holte, 1993] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, 11:63–91, 1993.

[Mingers, 1989] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243, 1989.

[Quinlan, 1987] John Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:221–234, 1987.

[Quinlan, 1990] John Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Schaffer, 1993] Cullen Schaffer. Overfitting avoidance as bias. *Machine Learning*, 10:153–178, 1993.