Relational Random Forests based on Random Relational Rules

Grant Anderson and Bernhard Pfahringer

University of Waikato, New Zealand
gpsa,bernhard@cs.waikato.ac.nz

Abstract. Random Forests have been shown to perform very well in propositional learning. FORF is an upgrade of Random Forests for relational data. In this paper we investigate shortcomings of FORF and propose an alternative algorithm, R^{4}F, for generating Random Forests over relational data. R^{4}F employs randomly generated relational rules as fully self-contained Boolean tests inside each node in a tree and thus can be viewed as an instance of dynamic propositionalization. The implementation of R^{4}F allows for the simultaneous or parallel growth of all the branches of all the trees in the ensemble in an efficient shared, but still single-threaded way. Experiments favorably compare R^{4}F to both FORF and the combination of static propositionalization together with standard Random Forests. Various strategies for tree initialization and splitting of nodes, as well as resulting ensemble size, diversity, and computational complexity of R^{4}F are also investigated.

1 Introduction

In propositional learning Random Forests [4] are one of the best performing off-the-shelf methods, competitive with both support vector machines and boosted decision trees. In relational learning ensemble methods in general and Random Forests specifically have not been investigated widely. Both [9] and [3] specifically report issues with excessive runtimes, and comparatively small increases over non-ensemble approaches. In this paper we try to address these issues by using a form of dynamic propositionalization in the spirit of [8] using random relational rules [1] as self-contained boolean tests inside decision tree nodes. We introduce a new algorithm that grows forests in a parallel fashion which minimizes costly coverage computation at the potential expense of diversity. Several variants of the plain algorithm can achieve satisfactory diversity without compromising the parallel generation process.

The next section describes the new algorithm[1] and discusses its complexity. Experimental results are reported and discussed in Section 3. Finally, Section 4 summarises and points out directions for future research.

[1] An extended version of this paper has been accepted by IJCAI2009 [2]

Presented at the 19th International Conference on Inductive Logic Programming (ILP'09).

2 Random Forests using Random Relational Rules

Propositional Random Forests [4] are a combination of Bagging with randomized decision tree induction: at each bagging iteration one randomized decision tree is generated for the respective bootstrap sample of the training data. Tree induction can be randomized in various different ways; in Random Forests it is done in the following way: instead of choosing the best test to split data at internal tree nodes, first a subset of all possible attributes is drawn at random, and then the best possible test out of that subset is selected. The size of this subset is a parameter specified by the user, but generally a default of $\sqrt{num\_attributes}$ has been found to work well.

In [3], Random Forests were upgraded to relational problem domains by replacing the propositional decision tree learner with a randomized version of a relational decision tree learner (Tilde). This approach exhibits two problems. Firstly, interpretation of relational trees is not straightforward: every node in a tree is either a test or a predicate, which might introducing new variables. The scope of these variables is limited to the successful (or "yes") branches of the tree. Each path from the root to a leaf in the tree can then be interpreted as a logical rule, but care must be taken with negated (or "no") branches, which must be represented by complex negations to ensure proper variable treatment. Secondly, in propositional Random Forest generation the total number of attributes available is a constant known upfront and it is therefore straightforward to specify the size for and compute random subsets thereof. When growing a relational decision tree, the number of possible tests or predicates at a given node is variable and a function of the current path from the root to the respective node. Also, in general this number tends to be both large and to grow quickly with the depth of the tree. In [3], a sampling approach was taken to estimate the number of possible tests and predicates applicable at a node, and then a user-specified percentage of these literals was actually evaluated to determine the single "best" literal out of this subset.

To combat these issues, R^4F takes a very different approach to relational Random Forest construction inspired by propositionalization. Each node in the tree uses one fully self-contained logical rule as a boolean test. If an example is covered by the rule, the test succeeds, and the example is passed down the "yes" branch; otherwise the test fails, and the example is passed down the "no" branch. This immediately alleviates the interpretation problem mentioned above and also simplifies and speeds up the randomized tree construction: every node simply needs to choose the best of a number of random relational rules, which from the tree construction point of view is simply choosing one boolean attribute from a random set of boolean attributes. Thus R^4F can be viewed as a Random Forest, where split attribute selection has been replaced with selecting one out of a small set of boolean features which are generated by some form of oracle on the fly as needed. Every one of these random features or rules is only evaluated for coverage once on the full dataset, and can then be used by any split node in any of all the trees of the ensemble, thus cutting down drastically on coverage

Algorithm 1 Pseudocode for the R^{4}F algorithm

```
Initialize the forest
while #(open leaves) > 0 do
  Generate a Rule and compute total cover
  if #(root nodes) < Maximum #(trees) then
    Initialize the next root node and add to open leaves
  end if
  for all Open Leaves do
    for all possible prefixes do
      Calculate infomation gain for the prefix
    end for
    if prefix with best infogain splits the leaf then
      if said prefix is better than current best then
        update current best prefix
      end if
    end if
    increment Rule Count
    if Rule Count == Maximum Rule Count then
      if bestPrefix == NULL then
        Close the leaf
      else
        Split the leaf using bestPrefix
        add children to open leaves
      end if
    end if
  end for
end while
```

computation compared to FORF. Such coverage computation is the single most expensive step in most relational learning systems.

We apply Random Relational Rules to random forests by using randomly generated rules as the splitting condition or boolean test in internal tree nodes. As the rule generation process is independent of the current tree state it is straightforward to parallelize tree and indeed forest generation. The pseudocode for the simplified Random Relational Forest algorithm (Random Relational Rules - Random Forests, or R^{4}F) is given in Algorithm 1.

Usually cover computation is the most time-consuming operation a relational learner needs to perform. This costly operation is executed exactly once for each random rule on the full dataset, and then every node on the waiting list can efficiently check whether the current rule actually properly splits its subset of the full data. This way all nodes of all trees of the ensemble can be grown in parallel. Clearly this operation would lead to identical trees, if all trees would be started simultaneously on the full dataset. To introduce the diversity necessary for good ensemble performance the algorithm staggers the start of individual trees, therefore different tree nodes see different subsets of random rules before making a split decision. Additional diversity is ensured by the use of bagging, i.e.

each root node is initialized with a randomly drawn bootstrap sample instead of the full training set. Other options for inducing more diversity were evaluated, but usually performed worse. Once nodes are initialized, and are not class-pure, they are put onto a list and will wait for rules that will split their data into two non-empty sets. After a user-defined maximum number of rules have been seen (MRC, or maximum rule count), a node will either be split on the best rule seen, or will be turned into a leaf predicting an appropriate class distribution, if no rule was found to split this node. Root nodes are an exception as they ignore the MRC setting and split on the first non-trivial rule seen. Together with bagging this ensures sufficient diversity of the trees, even though they are grown in parallel from the same stream of boolean features.

R^{4}F can also be seen as an example of dynamic propositionalization [8], in that the features are generated dynamically on-demand, and do not have to be precomputed in advance as would be common in static propositionalization [7]. Its runtime complexity is basically determined by the number of rules that must be generated before the last tree is started plus the number of rules needed to finish the last tree:

$$\text{Rules required for forest generation} \approx (\mathrm{n} + \mathrm{s}) \tag{1}$$

where n is the number of trees in the forest and s is the average number of rules required to construct a single tree. Empirical evidence confirming Equation 1 can be found in [1]. The number of rules required to construct a single tree can vary substantially. It is a function of the particular dataset, the Maximum Rule Count and the particular random rules generated. A worst case upper bound is given by:

$$\text{Max \#rules needed per tree} = (\mathrm{t} - 1) \times \mathrm{MRC} \tag{2}$$

where t is the size of the training set and MRC is the Maximum Rule Count. It is unlikely that this upper bound would be reached under normal circumstances, as it describes the pathological case where each node in the tree is split only after the maximum possible number of rules have been generated, and at every node the split has resulted in one single-instance leaf and one leaf containing all the remaining instances.

3 Empirical Evaluation

R^{4}F was tested using the following standard ILP datasets: Mutagenesis (with and without regression-unfriendly instances) [11], Musk1 [5], Cancer [10], and Diterpenes [6]. Mutagenesis and Cancer were limited to low-level structural information as represented by atoms and bonds; additional propositional information such as global properties *lumo* or *logP*, or predefined functional groups was deliberately excluded: they are known to improve classification accuracy significantly, thereby potentially masking the relational performance of the investigated algorithms. The current implementation of R^{4}F is limited to two classes, so the Diterpenes dataset was transformed into three two-class versions

by using all pairwise combinations of the three largest classes called 3, 52 and 54 – $\text{Diterpenes}_{54,3}$, $\text{Diterpenes}_{52,3}$ and $\text{Diterpenes}_{52,54}$.

All results were obtained as averages over ten times ten-fold cross-validation. Table 1 compares the accuracy obtained for each dataset, using both R^4F using 500 trees, bagged roots, and MRC equal to 50, and a static propositionalization based on 1000 non-trivial random rules turned into boolean features and processed by an equivalent standard Random Forest.

Table 1. Accuracy for R^4F using 500 trees, bagged roots, and MRC equal to 50 compared to static propositionalization.

Dataset	R^4F	Static	Significant
Carcinogenesis	**61.24**	60.91	no
$\text{Diterpenes}_{52,3}$	**97.31**	96.97	yes
$\text{Diterpenes}_{52,54}$	**96.24**	94.22	yes
$\text{Diterpenes}_{54,3}$	**98.43**	97.69	yes
Musk_1	85.32	**89.13**	no
Mutagenesis_{All}	**79.27**	76.66	no
Mutagenesis_{RF}	**85.99**	84.95	no

For comparison with published results for FORF[3], we also tested R^4F on Mutagenesis_{All} and the *Financial* dataset (Table 2)using out-of-bag evaluation rather than cross-validation. We compare to the highest FORF results listed in their paper, and see that R^4F significantly beats the standard aggregate-less FORF both times. Remember that R^4F currently does not include aggregates, still it is as good as the all FORF-plus-aggregates variants (FORF-LA: lookahead aggregates, FORF-SA: simple aggregates, and FORF-RA: refined aggregates) on one of the two datasets. Clearly, for the second dataset aggregates are essential for near-perfect prediction. Adding aggregates to R^4F will be one major direction for future research.

Table 2. Comparison of R^4F and FORF (out-of-bag evaluation) on $Mutagenesis_{RF}$ and *Financial*

Algorithm	Accuracy	Muta			Accuracy	Financial	
R^4F	79.1 ± 1.1	R^4F is	Significance		87.8 ± 0.7	R^4F is	Significance
FORF	74.7 ± 1.4	better	95%		85.7 ± 0.6	better	95%
FORF-SA	78.9 ± 1.8	equal	$< 90\%$		99.8 ± 0.2	worse	99%
FORF-RA	78.1 ± 1.2	equal	$< 90\%$		99.7 ± 0.4	worse	99%
FORF-LA	79.0 ± 1.4	equal	$< 90\%$		99.8 ± 0.4	worse	99%

In addition to measuring accuracy we have also conducted studies to judge the sensitivity of the algorithm to parameter settings and to alternative split rule selection methods. For lack of space we can only summarize here, claiming that, as expected, larger number of trees consistently produce higher accuracies, but that after a couple of hundred trees accuracy usually levels out on a plateau. The MRC or maximum rule count is more of an optimization parameter, usually with optimal values between 50 and 100, where lower values tend to underfit, and higher values cause overfitting. Measuring tree sizes and tree diversity, fully random trees are usually larger and more diverse, and higher MRC values reduce both the size and diversity of trees. For a lot more detail please consult [1].

4 Summary and Future Work

The efficiency and effectiveness of the R^4F algorithm is the result of the application of randomly generated relational rules to the random forests framework. Staggered root initialization allows R^4F to produce diverse trees in parallel, and the experimental results obtained are very competitive with those achieved by another Relational Random Forest algorithm. The main direction for future work will be the inclusion of aggregates into R^4F. Alternative and more targeted random rule generation methods, which work on explicit seed examples that must be covered by a rule, will also be investigated.

References

1. G. Anderson. PhD thesis: Random relational rules, 2009.
2. G. Anderson and B. Pfahringer. Relational random forests based on random relational rules. In *Proc. Int. J. Conf. on Artificial Intelligence (IJCAI09)*, 2009.
3. A. Assche, C. Vens, H. Blockeel, and S. Džeroski. First order random forests: Learning relational classifiers with complex aggregates. *Mach. Learn.*, 64(1-3):149–182, 2006.
4. L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
5. T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.
6. S. Dzeroski, S. Schulze-Kremer, K. R. Heidtke, K. Siems, and D. Wettschereck. Applying ilp to diterpene structure elucidation from 13c nmr spectra. In *ILP '96: Selected Papers 6th Int. Workshop on ILP*. Springer, 1997.
7. S. Kramer, N. Lavrač, and P. Flach. Propositionalization approaches to relational data mining. *Relational Data Mining*, pages 262–286, 2000.
8. N. Landwehr, A. Passerini, L. De Raedt, and P. Frasconi. kfoil: Learning simple relational kernels. In *AAAI*. AAAI Press, 2006.
9. J. R. Quinlan. Relational learning and boosting. *Relational Data Mining*, pages 292–304, 2000.
10. A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg. Carcinogenesis predictions using ILP. In S. Džeroski and N. Lavrač, editors, *Proc. of the 7th International Workshop on Inductive Logic Programming*. Springer, 1997.
11. A. Srinivasan, S. Muggleton, R.D. King, and M.J.E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237. GMD, 1994.