Relational Random Forests Based on Random Relational Rules

Grant Anderson
Department of Computer Science
University of Waikato
gpsa@cs.waikato.ac.nz

Bernhard Pfahringer
Department of Computer Science
University of Waikato
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, $R^4F$, for generating Random Forests over relational data. $R^4F$ 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 $R^4F$ 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 $R^4F$ 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 $R^4F$ are also investigated.

1 Introduction

In propositional learning Random Forests [Breiman, 2001] 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 [Quinlan, 2000] and [Assche et al., 2006] 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 [Landwehr et al., 2006] using random relational rules [Anderson and Pfahringer, 2007] 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 and discusses its complexity. Experimental results are reported and discussed in Section 3. Finally, Section 4 summarises and points out directions for future research.

2 Random Forests using Random Relational Rules

Propositional Random Forests [Breiman, 2001] 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 $\sqrt{\text{num\_attributes}}$ has been found to work well.

In [Assche et al., 2006], 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:

- 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.

- The second and more practical issue is a consequence of the specific form of randomization chosen in FORF. 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 [Assche et al., 2006], 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. This approach is problematic for various reasons. First of all it is computationally expensive, as both the estimation and the subsequent coverage computation for the subset are expensive and have to be repeated for every single node in every tree in the ensemble. In their empirical investigation they have found that they need high percentages (25% and more) to achieve good performance, contrary to propositional Random Forests, which do well with much smaller percentages. Not surprisingly, FORF is therefore much slower than its propositional counterpart, and also does not seem to yield such impressive accuracy improvements as Random Forests do. The latter might be a consequence of the large number of tests and predicates being explored; there is some anecdotal evidence, but unfortunately no good publication, that suggests that even Random Forests struggle to cope with very large numbers of attributes, as tree construction runs out of examples exponentially fast.

To combat these issues, \( R^4 F \) 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 \( R^4 F \) 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 computation compared to FORF. Such coverage computation is the single most expensive step in most relational learning systems.

The following subsections will explain the construction of random relational rules, detail the construction of the random forests based on these random rules, and discuss its computational complexity.

2.1 Rule Construction
The random relational rules employed here are definite clauses, which comprise both predicates containing variables, as well as tests on and comparisons between these variables and so-called theory constants. Neither functors nor recursion are permitted. For example, the Mutagenesis dataset [Srinivasan et al., 1994] comprises the following three predicates:

\[
\text{molecule(MolID, Class)} \\
\text{atom(MolID, AtmID, El, Quanta, Charge)} \\
\text{bond(MolID, AtmID1, AtmID2, BondType)}
\]

A molecule is described by two parameters: a unique identifier and a class label (active or inactive). An atom is described by five parameters: the identifier of the molecule it belongs to, a unique identifier, its element type, its quanta type, and its electrical charge. A bond is described by four parameters: the identifier of the molecule that it belongs to, the unique identifiers for the two atoms it is linking, as well as its own bond type. An example of a rule generated on that dataset is:

\[
\text{active(MolID):=} \\
\text{atom(MolID,_,_,_,Charge),} \\
\text{Charge >= 0.078,} \\
\text{bond(MolID,_,AtomID1,BondType1),} \\
\text{bond(MolID,_,AtomID2,BondType2),} \\
\text{BondType1 != BondType2,} \\
\text{AtomID1 = AtomID2.}
\]

This rule describes all compounds that contain an atom with a charge above 0.078 and two bonds of different types that both include a particular atom. Underscores are used here for clarity, to denote variables not used in this rule.

Such random rules are generated in the following way: every rule has a user-specified length; at each stage a predicate or test is chosen uniformly at random with the following restrictions: for a predicate exactly one variable (or parameter) must already appear in the rule; all other variables are new. This ensures that clauses are linked. Tests on the other hand may not add any new variables. Tests include the usual equal and not-equal comparisons to other variables or theory constants, as well as range comparisons for numeric arguments. All prefixes of a rule can also be used as (shorter) random rules. Coverage computation can produce coverage information for all prefixes of the full rule at no additional cost.

2.2 Forest Construction
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 pseudo code 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 staggering 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 into the ensemble were also tried, and will be discussed and evaluated in the next section. 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.

To clarify the forest building procedure, Figures 1 to 3 show three exemplary stages of forest construction, with the trees designated by letters, and internal nodes marked with an identifier corresponding to the rule that split them. Nodes are described by the tree designation followed by a numeric identifier. The training data consists of 20 instances, 10 each of two classes. The class distribution at each node is given to the node’s left. For simplicity we assume that MRC is set to one, i.e. splitting is done immediately, if possible at all. We also assume that full rules (not prefixes) lead to the highest information-gain in all cases.

Figure 1 shows the state of the forest after the first rule, Rule R1, has been added. Initially, the root node of Tree A, A1, was the only node on the Open Leaf List. Now, A1 has been split, and two leaves A2 and A3 created. A2, A3 and the root node of Tree B, B1 have been added to the Open Leaf list and A1 has been removed from the list.

![Figure 1: Example of R^4F Forest Construction, Stage 1](image)

Figure 2 shows the state of the forest after another rule, R2, has been processed. Both A2 and A3 have been split, B1 has been split, and most of the new leaves thus created (A5 through A7 and B2 through B3), along with the root node of Tree C, C1, have been added to the Open Leaf list. A4 now contains instances of only one class (denoted by the double circle) and so was not added to the Open Leaf list, and will not be split any further.

Figure 3 shows the state of the forest after the third rule R3 has been processed. Nodes A5 and A6 have been split, adding A8 through A11 to the Open Leaf list. Node A7 has not been split by R3, and therefore had its Fail count incremented. We set the maximum fail count to 1 (to keep this example concise), and thus A7 (marked by the crossed circle) will now be removed from the Open Leaf list and will not be split any further. A7 will predict a class distribution of (1/3, 2/3) for test examples. Nodes B2 and B3 have been split, producing B4 through B7, of which B4, B6 and B7 will be added to the Open Leaf list, while B5 will not, as it is class-pure. C1 has been split, producing C2 and C3, and the root of tree D, D1, has also been added to the Open Leaf list.

To summarize, R^4F differs from FORF [Assche et al., 2006], in two main ways. First of all FORF does not use full rules in every node, but in contrast paths from the root to each leaf comprise rules. As logical variables can only be shared across positive pathes, this complicates both generation as well as interpretation of such trees. Like Breiman’s original random forest, FORF randomly restricts the set of possible tests (features) and then picks the best test from that restricted set. R^4F on the other hand uses a fully self-contained randomly generated relational rule as a test. As a consequence, R^4F can easily generate its trees in a staggered parallel fashion, with each new rule being available for all open leaves, while FORF processes both nodes and trees fully sequentially. Even though in theory FORF could also parallelize node and tree generation, it could still not share the expensive cover computation across nodes or trees the way R^4F can.

R^4F can also be seen as an example of dynamic propositionalization [Landwehr et al., 2006], in that the features are generated dynamically on-demand, and do not have to be pre-computed in advance as would be common in static propositionalization [Kramer et al., 2000].
2.3 Complexity of Forest Construction

When $R^4F$ generates and evaluates a rule, it then applies the derived test at every open leaf in each active tree. The cost of rule evaluation is the same as for previous uses of Random Relational Rules [Botta et al., 2003; Anderson and Pfahringer, 2007]. As each test is applied to all open leaves, the number of rules required to be evaluated is substantially less than would be required if a new rule were being evaluated for each open leaf, as in standard random forests. The number of rules required for forest construction is heavily influenced by the number of trees in the forest and the Maximum Rule Count. A rough estimate for this value is the sum of the average number of rules required to construct a single tree and the number of trees in the forest, as when the last tree in the forest is completed, the previous trees are also likely to be complete:

$$\text{Rules required for forest generation} \approx (n + s)$$  \hspace{1cm} (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 [Anderson, 2009].

Because of the staggered fashion in which the trees are generated, each tree has access to at least one more rule than its immediate successor and so its construction has probably already finished at the time the construction of that successor finishes. Thus, when the final tree is complete, it is likely that all previous trees are complete or nearly so.

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:

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

where $t$ is the size of the training set and $\text{MRC}$ 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 (see Figure 4). In practice the number of rules required has always been substantially lower than this worst case upper bound.

3 Empirical Evaluation

$R^4F$ was tested using the following standard ILP datasets: Mutagenesis (with and without regression-unfriendly instances) [Srinivasan et al., 1994], Musk1 [Dietterich et al., 1997], Carcinogenesis [Srinivasan et al., 1997], and Diterpenes [Dzeroski et al., 1997]. Mutagenesis and Carcinogenesis were limited to low-level structural information as represented by atoms and bonds; additional propositional information such as global properties $\text{lumo}$ or $\text{logP}$, 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 $R^4F$ is limited to two classes, so the Diterpenes dataset was transformed
Table 1: Accuracy for $R^4F$ using 500 trees, bagged roots, and MRC equal to 50 compared to static propositionalization.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$R^4F$</th>
<th>Static</th>
<th>Significant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carcinogenesis</td>
<td>61.24</td>
<td>60.91</td>
<td>no</td>
</tr>
<tr>
<td>Diterpenes$_{52,3}$</td>
<td>97.31</td>
<td>96.97</td>
<td>yes</td>
</tr>
<tr>
<td>Diterpenes$_{52,54}$</td>
<td>96.24</td>
<td>94.22</td>
<td>yes</td>
</tr>
<tr>
<td>Diterpenes$_{54,3}$</td>
<td>98.43</td>
<td>97.69</td>
<td>yes</td>
</tr>
<tr>
<td>Musk$_1$</td>
<td>85.32</td>
<td>89.13</td>
<td>no</td>
</tr>
<tr>
<td>Mutagenesis$_{All}$</td>
<td>79.27</td>
<td>76.66</td>
<td>no</td>
</tr>
<tr>
<td>Mutagenesis$_{RF}$</td>
<td>85.99</td>
<td>84.95</td>
<td>no</td>
</tr>
</tbody>
</table>

into three two-class versions by using all pairwise combinations of the three largest classes called 3, 52 and 54 - Diterpenes$_{54,3}$, Diterpenes$_{52,3}$ and 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.

$R^4F$ has several advantages over a static two-stage method that generates a propositional representation of the data first, and then constructs a Random Forest based on the propositionalized data. The latter approach must generate a sufficiently large number of rules in the first stage without knowing which ones will actually be useful. The propositional representation is potentially very large, but might still not be a good enough approximation of the relational problem. Thus the number of rules to generate will be a critical parameter for the user to set. $R^4F$ on the other hand has a simple stopping condition: completion of the forest, so it never generates more rules than needed. No memory is needed for any intermediate representation, and forest generation is fully parallel. Still, as can be seen in this table, static propositionalization works fairly well for the datasets studied here, and even outperforms $R^4F$ on the Musk dataset, even though that win is not statistically significant due to the small size of that dataset. The only significant differences are the wins for $R^4F$ on each of the three Diterpenes datasets, in all other cases the standard deviations over the ten times ten-fold cross-validation runs are too large.

For comparison with published results for FORF[Assche et al., 2006], we also tested $R^4F$ on Mutagenesis$_{All}$ (Table 2) and the Financial dataset (Table 3) using out-of-bag evaluation rather than cross-validation. This was necessary as apparently it is not feasible to evaluate FORF using ten times ten-fold cross-validation. In their paper they used only four datasets, two of which are used here as well, and two were dropped for the following reasons: one is simply too small for any meaningful comparison, as it only comprises 20 examples in total; the other one is a multiple classes problem that the current version of $R^4F$ cannot handle yet. We com-

Figure 3: Example of $R^4F$ Forest Construction, Stage 3
Table 2: Comparison of $R^4F$ and FORF (out-of-bag evaluation) on MutagenesisRF

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>$R^3F$ is</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^4F$</td>
<td>79.1 ± 1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORF</td>
<td>74.7 ± 1.4</td>
<td>better</td>
<td>95%</td>
</tr>
<tr>
<td>FORF-SA</td>
<td>78.9 ± 1.8</td>
<td>equal</td>
<td>&lt; 90%</td>
</tr>
<tr>
<td>FORF-RA</td>
<td>78.1 ± 1.2</td>
<td>equal</td>
<td>&lt; 90%</td>
</tr>
<tr>
<td>FORF-LA</td>
<td>79.0 ± 1.4</td>
<td>equal</td>
<td>&lt; 90%</td>
</tr>
</tbody>
</table>

Table 3: Comparison of $R^4F$ and FORF (out-of-bag evaluation) on Financial

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Accuracy</th>
<th>$R^3F$ is</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^4F$</td>
<td>87.8 ± 0.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FORF</td>
<td>85.7 ± 0.6</td>
<td>better</td>
<td>95%</td>
</tr>
<tr>
<td>FORF-SA</td>
<td>99.8 ± 0.2</td>
<td>worse</td>
<td>99%</td>
</tr>
<tr>
<td>FORF-RA</td>
<td>99.7 ± 0.4</td>
<td>worse</td>
<td>99%</td>
</tr>
<tr>
<td>FORF-LA</td>
<td>99.8 ± 0.4</td>
<td>worse</td>
<td>99%</td>
</tr>
</tbody>
</table>

pared 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.

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 on these aspects please see [Anderson, 2009].

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 $R^4F$: FORF benefits significantly from adding aggregates, so $R^4F$ might also be able to take advantage of the additional expressiveness that aggregates provide. As it currently stands, $R^4F$ outperforms the non-aggregate version of FORF and is already equal in performance to the aggregate-enhanced version of FORF on some datasets. 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


