Millions of Random Rules

Bernhard Pfahringer, Geoffrey Holmes, and Cheng Weng

Department of Computer Science
University of Waikato
Hamilton, New Zealand
{bernhard, geoff, cw41}@cs.waikato.ac.nz

Abstract. In this paper we report on work in progress based on the induction of vast numbers of almost random rules. This work tries to combine and explore ideas from both Random Forests as well as Stochastic Discrimination. We describe a fast algorithm for generating almost random rules and study its performance. Rules are generated in such a way that all training examples are covered roughly by the same number of rules each. Rules themselves usually have a clear majority class among the examples they cover, but they are not limited in terms of either minimal coverage, nor minimal purity. A preliminary experimental evaluation indicates really promising results for both predictive accuracy as well as speed of induction, but at the expense of both large memory consumption as well as slow prediction. Finally, we discuss various directions for our future research.

1 Introduction

In this paper we describe current work on exploring the potential of random rules. This work is based both on the theory of Stochastic Discrimination ([1]) as well as the ideas behind Random Forests [2] (and consequently also Bagging).

The algorithm for generating random rules does not employ the standard separate-and-conquer approach that is ubiquitous in inductive rule learning. It is more closely related to alternative rule learners like Slipper [3] and the work on stochastic search for rule learning described in [4]. Somehow stochastic approaches generating lots of potentially complex rules are a kind of anti-thesis to the work on extremely simple classifiers like decision stumps [5].

In the following section Stochastic Discrimination will be explained. Section 3 describes the actual algorithm used for generating random rules. In Section 4 a preliminary experimental evaluation is conducted, where we mainly explore the effect of the size of the random rule sets. In the final section we describe and discuss future directions.

2 Stochastic Discrimination: a one page explanation

Stochastic Discrimination [1] was developed by Eugene Kleinberg, a mathematician, in the context of pattern recognition, which explains why it is almost unknown in the Machine Learning community. The basic idea is very simple:

- For each class, generate almost randomly thousands of features (or even more).
- For prediction use equal-weight voting over all these features and simply predict the class with the highest vote.

Obviously, the interesting step is the "almost random" generation of features. Given some random generator for features, a feature is accepted by the process when it is both what Kleinberg calls "enriched" and improves coverage towards "uniform coverage". A feature is called "enriched" for a class when the *percentage* of examples of this class covered by the feature is higher than the default percentage of this class in the training set. Note that for skewed class distributions enrichment can already be achieved by rules whose majority class *differs* from the class in question. For example, if in a two-class problem `class A` is represented by 10 out 100 training examples, then a feature covering 5 examples of `class A`, but also 10 examples of `class B` is "enriched" for `class A` ($\frac{1}{3} > \frac{1}{10}$), even though the majority class of the cover of that feature is still `class B`.

Enrichment on its own is not enough to guarantee good generalization and therefore good predictive accuracy on new examples. For the voting and averaging process described above the best generalization behavior is achieved when each training example of one class is covered by exactly the same number of features, i.e. when the average cover is equal to each individual cover, or to put it another way, when the variance of the coverage is zero. In practice it is not always possible to generate feature sets exhibiting such "uniform coverage", but the feature selection process is usually targeting examples that currently show below-average coverage.

Even though stochastic discrimination has been applied successfully in pattern recognition, there are not a lot of approaches applying it in a Machine Learning framework. One exception is the work by Ho on so-called "decision forests" as summarized in [6], where a lot of random and correct (and therefore obviously over-fitting) decision trees are generated and simply voted. Coverage is not a problem in this case as each tree covers each example, and enrichment is given by default, as the leaves of the overfitted tree are pure. Only conflicting instances (instances with identical attribute-values, but differing class-values) pose a problem, like they do for every learning algorithm.

This "decision forest" algorithm does have a close relation in Machine Learning called "Random Forests" [2]. These two algorithms differ in one important detail in the tree generation process: random forest trees are generated from bootstrap samples instead of the whole training set, which might be an advantage in the case of noise in the training data. Random forests have been successfully applied to a few real-world problems recently, and some experiments show them to be competitive even with boosting, the current number one off-the-shelf ensemble learning technique.

When generating random rules instead of random trees, we found it hard to actually encode Kleinberg's enrichment criterion in a way that would lead to reasonable performance, i.e. we failed to find an operational definition for enrichment. Therefore we have combined Kleinberg's uniform-coverage idea with

Random Forest's (and Bagging's) idea of sampling with replacement into the algorithm for generating random rules that will be described in the next section.

3 Generating random rules

We have designed an algorithm that generates rules randomly while still trying to cover all training examples evenly. In initial experiments rules were generated completely randomly, and then checked for suitability, but that approach turned out to be much too inefficient. Too many rules would simply be generated and discarded immediately because they were found to be unsuitable. Consequently we have devised the following, still random, strategy that will only generate suitable rules by construction:

- For all training examples initialise cover-count to 0
- Rules = null
- Repeat N times (where N is a user defined parameter):
  1. Rule = true
  2. Seed = Randomly choose a low-coverage example
  3. Neg = Sample with replacement from all other classes
  4. While (Rule covers some x in Neg) do
     (a) NegEx = Randomly choose an example in Neg
     (b) Test = Randomly choose a test covering Seed, but not NegEx
     (c) add Test to Rule
     (d) Neg = $\{x \in Neg | Test.covers(x)\}$
  5. For all training examples x, if Rule.covers(x) then
     if (x.class == Seed.class) increment x.cover
     else decrement x.cover
  6. add Rule to Rules

As we always choose a low-coverage example, we will on average achieve equal coverage for all training examples. The way a single rule is generated, it is guaranteed that the rule will at least cover the seed example and therefore increase its cover count. Additionally, as the rule is correct for the bootstrap sample over all the other classes, on average a reasonably strong rule for the specific class of the seed example will have been generated. As the process does not involve any other examples from the class of the seed example, there is of course no guarantee whatsoever in terms of reasonable coverage for that class. But the experiments reported below indicate that the rules seem to generalize reasonably well most of the time.

Currently the language used for constructing random tests is simple, but straightforward. Assuming that the seed-example and the chosen negative example have different values for a randomly chosen attribute (and assuming no missing values), then for a nominal attribute a test $attr == value_{seedExample}$ is generated; for a numeric attribute a threshold is computed halfway between the two examples' values. During prediction tests executed on missing values always fail.

There are two special cases that have to be considered when running this algorithm. First, examples with different class labels but identical values for all other attributes cause problems, as it will be impossible to generate a rule distinguishing between the two examples. Second, examples with all missing values for their attributes cause problems, as they will lead to inducing the trivial empty rule, which covers all examples. Consequently such a rule could not improve the coverage situation. Both cases occur in practise. Presumably they could be dealt with by some kind of preprocessing, but the current implementation of the algorithm is fitted with special tests to recognise both cases, and deal with them by basically removing the offending examples from the training set, i.e. effectively ignoring them. This is a working solution, but it is not obvious that it is the optimal solution.

Prediction using random rules is straightforward: all rules are given equal weight for voting, and ties are broken in favour of the majority class.

As formulated above, the algorithm involves a single parameter, the number of rules to generate. The experiments reported below seem to indicate that no careful tuning of this parameter is necessary. Usually larger values seem to give better results, but with diminishing returns after a certain level is reached. Also, a larger number of rules leads to longer induction times, higher memory consumption as well as slower prediction of new instances.

Additionally, there is a second potential parameter hiding in the current implementation: the size of the bootstrap sample. We have not performed any experiments yet varying this size, but obviously for smaller samples one would expect more general rules to be generated, but these rules might also exhibit a higher false-positive rate. Therefore the size of the bootstrap sample might actually be a true tuning parameter, allowing for adaptation to a given training set.

4 Preliminary experimental evaluation

For this preliminary experimental evaluation we have only employed 16 binary UCI datasets. We compare sets of random rules of different sizes with two different baseline algorithms[1]:

- Ripper: which usually generates a small, but accurate set of rules.
- Linear SVM: a fast and robust separating hyperplane learner.

Table 1 shows a few learning curves, plotting the average number of incorrectly classified examples over the number of rules induced in a log-log plot. The means are computed from ten runs of ten-fold cross-validation, the red line depicts the mean, the green lines indicate the variability (mean plus or minus one standard error). All the curves basically exhibit the same shape, starting

[1] The obvious candidate algorithm for comparison, Random Forests, is missing here because of technical issues with the current Weka version of it. Such a comparison will be important future work.

Table 1. Learning curves for various datasets

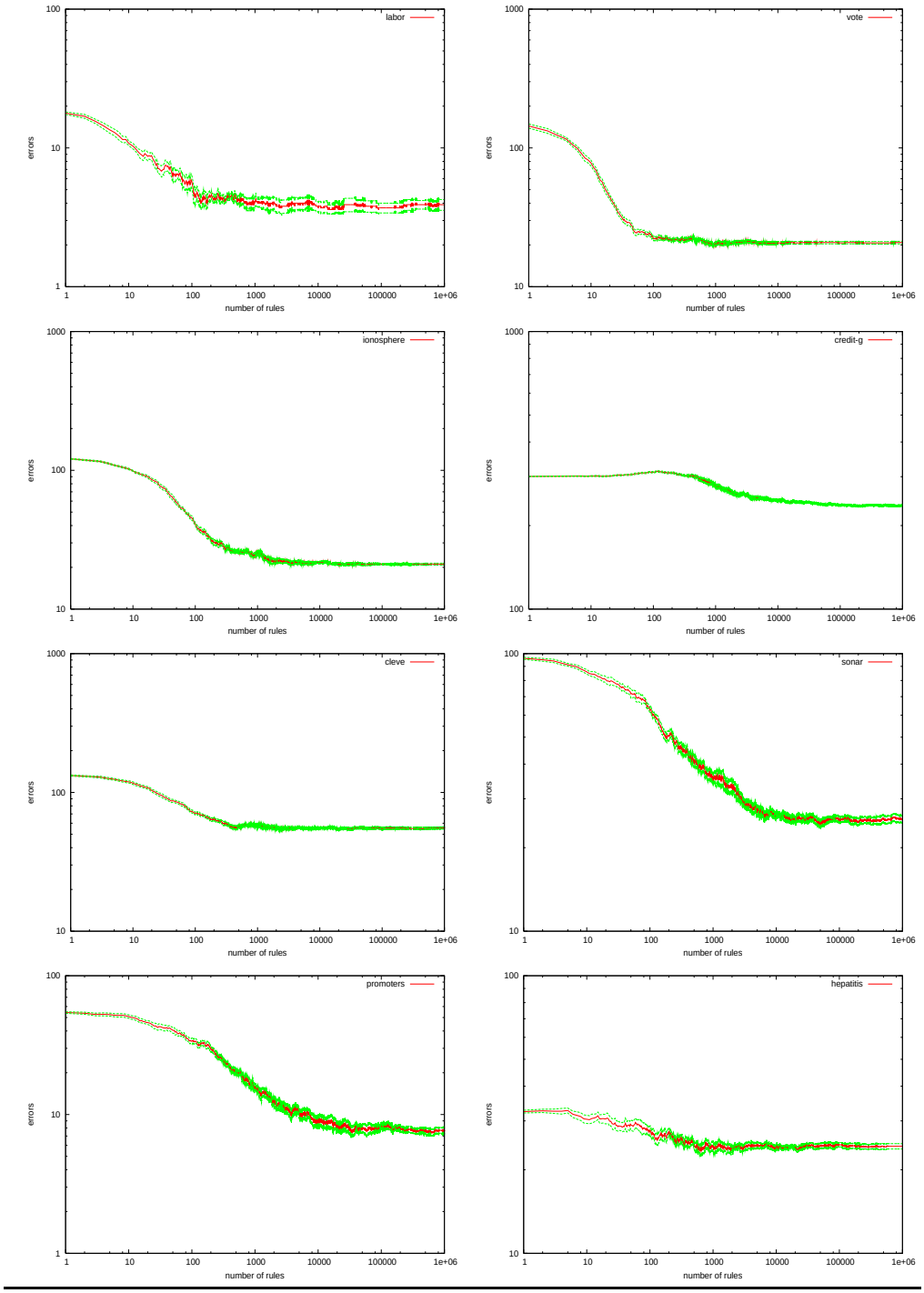

Table 2. Predictive accuracies, v and * indicate significant differences to the first column

Dataset	JRip	SMO	RR10	RR100	RR1000	RR10000	RR100000
kr-vs-kp	99.21	95.79 *	59.21 *	86.11 *	99.12	99.51	99.53
credit-a	85.16	84.88	59.33 *	75.1 *	85.25	87.3	87.55 v
sick	98.29	93.87 *	93.97 *	95.31 *	98	98.49	98.58
breast-w	95.61	96.75	82.69 *	95.78	97 v	96.9	96.9
credit-g	72.21	75.09	70.02	69.31	71.63	75.5 v	76.21 v
ionosphere	89.16	88.07	71.62 *	88	93.25 v	93.94 v	94.02 v
breast-cancer	71.45	69.52	70.32	69.2	70.22	70.47	69.84
hepatitis	78.13	85.77 v	79.28	82.99	84.29	84.98 v	85.05 v
cleve	79.95	83.86	62.34 *	76.13	80.53	81.39	81.88
heart-statlog	78.7	83.89 v	62.04 *	77.15	82.26	81.67	81.89
diabetes	75.18	76.8	65.33 *	67.37 *	72.45	74.87	75.36
sonar	73.4	76.6	56.39 *	69.9	82.8 v	86.69 v	87.7 v
labor	83.7	92.97	78	90.7	92.73	92.77	92.43
vote	95.75	95.77	83.47 *	94.83	95.38	95.26	95.31
promoters	81.27	91.01 v	51.31 *	66.4 *	84.06	90.94 v	92.52 v
vote1	89.47	91.63	73.78 *	88.57	89.98	89.91	90.07
Summary		(3/11/2)	(0/4/12)	(0/11/5)	(3/13/0)	(5/11/0)	(6/10/0)

Table 3. Algorithm rankings for accuracies, training and test times

Rank	Accuracy	Train time	test time
1	RR100000	RR10	JRip
2	RR10000	RR100	RR10
3	RR1000	RR1000	SMO
4	SMO	JRip	RR100
5	JRip	SMO	RR1000
6	RR100	RR10000	RR10000
7	RR10	RR100000	RR100000

from default error dipping down to the limit-performance more or less quickly and more or less smoothly. While most curves are pretty flat and smooth at the one million rule point, some still look a bit rough. An example is the learning curve for the "labor" dataset (top left in Table 1), but that roughness could be attributed to the fact that we plot the number of incorrect predictions, which happens to be very low is this case. The curve for the "credit-g" dataset is the only real exception to the standard behavior, as the error rate initially actually increases to worse than default performance, and only later between a thousand and ten thousand rules improves onto the final better than default performance.

Table 2 compares performance after a fixed number of iterations to both the Weka versions of Ripper (JRip) and linear support vector machines (SMO). Again, all results are averages over ten runs of ten-fold cross-validation, marks indicate significant differences to JRip according to a modified t-test at a 95% significance level. As can be seen from this table, at least a thousand rules are needed to match JRip, but from there on differences slowly gain significance with more and more rules.

Table 3 summaries the accuracies by ranking the algorithms according to significant differences. Additionally we have also ranked the algorithms according to their average induction and prediction times (called train time and test time respectively). The random rule sets with a thousand rules or more outperform both JRip and SMO, and generating up to a thousand rules is still faster than both JRip's or SMO's model generation. But when predicting new instances, even a rule-set with only one hundred rules needs more time than either JRip or SMO.

In summary, given enough rules (1000 or more for most datasets) the performance seems to be quite good in comparison to the two baseline learners. But obviously these results are of limited generality, as they have been derived from a subset of small, two-class UCI datasets only. Future work will investigate larger and especially also multi-class datasets.

5 Conclusion

In this paper we have described a new method for generating almost random rule sets using ideas from stochastic discrimination and Random Forests. A very preliminary experimental evaluation indicates the potential of the method. As this is work in progress, there are quite a number of directions for future work:

- Speed-up prediction: if one were to actually employ such a rule learner and millions of rules would be needed for good performance, prediction has to be sped up. Compiling and thereby compressing the rule set (which may even contain duplicates) into an alternating decision tree [7] might be one option. Alternatively compilation ideas from the field of production rule systems [8] might be applicable instead. And of course, all kinds of pruning methods (see e.g. [9]) could be investigated.
- Lazy induction: if a lot of rules are needed, and especially if the rule-set is actually larger than the dataset it was generated from, and possibly even

too large to fit into main memory, a lazy approach might be more feasible (cf. [10]), especially if rule generation is fast enough.
- Stopping criterion: currently the number of rules to generate is a user-parameter. Alternatively a method could be devised that would determine an appropriate number of rules automatically. The eager approach might use some modified form of out-of-bag estimation, whereas the lazy approach could use the current margin of prediction as input for some statistical test (e.g. using Hoeffding bounds [11]). Such a test would hopefully recognize that a steady state has been reached with high reliability.
- Alternative uses: like Random Forests, random rules could be utilized in at least two additional ways. First, for ranking attributes for attribute selection; simply counting the number of times an attribute is used in a rule seems to produce quite reasonable attribute rankings. Secondly, rules could be used to define a notion of similarity between examples, whereby such similarity is defined by the number of rules that cover both examples.
- Explanation: it would be useful to have a good explanation of why such sets of random rules work well. One approach would definitely be along the lines of Kleinberg and Breiman utilizing the law of large numbers in statistics. Alternatively, an interpretation of it being a kind of nearest neighbour with a parameter-free or model-free distance function (cf. the similarity function defined in the previous item) might provide useful insights as well.
- Bias-Variance analysis: it would be interesting to evaluate the presented rule learning algorithm in a bias-variance decomposition framework. This analysis would also compare sets of single rules to sets of rule sets. From a theoretical point of view one might expect the latter approach to outperform the former, but practically at least sets of single rules seem to perform very well. A bias-variance decomposition could support explaining this practical finding. Furthermore, a comparison of random rules to boosting of rules might be of interest, especially as some rules are duplicated, sometimes multiple times, thus effectively weighting the voting. The best candidate for comparison to boosted rules is probably the Slipper algorithm [3], which uses a reweight-and-conquer approach to rule learning instead of the standard separate-and-conquer or covering approach.

In summary we claim that the algorithm described represents an interesting alternative to the standard separate-and-conquer approach usually employed in inductive rule learning. It clearly deserves further investigation. Future work will especially have to assess whether it is also of practical usefulness.

Acknowledgements

We would like to thank the anonymous reviewers for valuable suggestions, especially on the relationship to boosting as well as the potential of bias-variance decomposition for the analysis of the presented algorithm.

References

1. Kleinberg, E.M.: Stochastic discrimination. Annals of Mathematics and Artificial Intelligence **1** (1990) 207–239
2. Breiman, L.: Random forests. Machine Learning **45** (2001) 5–32
3. Cohen, W.W., Singer, Y.: A simple, fast, and effective rule learner. In: Proceedings of the 16th National Conference on Artificial Intelligence (AAAI-99), AAAI/MIT Press (1999) 335–342
4. Rückert, U., Kramer, S.: Stochastic local search in k-term dnf learning. In Fawcett, T., Mishra, N., eds.: Proceedings of the 20th International Conference on Machine Learning (ICML-03). (2003)
5. Holte, R.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **11** (1993) 63–90
6. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence **20** (1998) 832–844
7. Freund, Y., Mason, L.: The alternating decision tree learning algorithm. In: Proceedings of the 16th International Conference on Machine Learning (ICML-99). (1999) 124–133
8. Forgy, C.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19** (1982) 17–37
9. Fürnkranz, J.: Pruning algorithms for rule learning. Machine Learning **27** (1997) 139–171
10. Zheng, Z., Webb, G.I.: Lazy learning of bayesian rules. Machine Learning **41** (2000) 53–84
11. Hoeffding, W.: Probability for sums of bounded random variables. Journal of the American Statistical Association **58** (1963) 13–30