H-0255 (H0602-016) February 11, 2006
Computer Science

IBM Research Report

Learning Rules of Thumb

Dan Pelleg
IBM Research Division
Haifa Research Laboratory
Mt. Carmel 31905
Haifa, Israel

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Learning Rules of Thumb

Dan Pelleg[1]

IBM Haifa Research Labs

Abstract. When machines classify data, the rules can be arbitrarily complex. Humans, on the other hand, have trouble manipulating large structures and algebraic formulae. Therefore, when the rules are learned for the purpose of their execution by humans, the classifier's accuracy needs to be balanced against the hardness of evaluating the rules given an unseen instance. This work examines several known learners in this light, and proposes a new algorithm for building human-friendly rules from datasets where easy classifications exist. We also propose a new accuracy-complexity measure. In contrast to measures like BIC where the complexity penalty is rigid, ours allows flexibility in the weight given to complexity. We evaluate our "rule of thumb" learner against the known learners, and show that it produces better results over a wide range of the complexity-accuracy trade-off parameter.

1 Introduction

Often in real life, people are required to make "fast and frugal" [1] decisions. The canonical example is triage in an emergency room, where a classical decision tree may contain many nodes and require many physiological tests to obtain attribute values. In contrast, a much shallower tree with many fewer nodes will be faster and more practical to follow. The improvement in decision time balances the (presumably small) loss of prediction accuracy.

The same approach also holds in less extreme scenarios. Administrators of large data storage systems can use machine learning tools to help them identify system performance degradation and anticipate or mitigate downtime. But it would be much more effective if they could also identify corrective actions, such as throttling certain applications, or ordering hardware upgrades. Ideally, such actions will be triggered by system-level events that can be easily measured and found to cross some threshold value [2]. Once learned, the events and their corresponding actions could be taught at the training classes for the administration of the corresponding storage system.

Our goal is to suggest a learner for a model that only contains very few if-then elements, where each conditional is a simple threshold (or equality) test of a single input attribute. It is flexible enough to lengthen the list of rules to improve accuracy, but does this in addition to optimization for brevity. The idea is not to produce a general-purpose classifier, but rather one that works if indeed there is such an easy separation of the data. In more complex cases, its simplicity will necessarily mean high error rate. In those cases it would be preferable to use a richer classifier, and possibly abandon the idea of manual classification.

In the statistical context, penalization of model complexity has been addressed by several model-selection criteria (e.g., AIC and BIC [3]). Below we propose an alternative measure, which differs in that it can be directly elicited from a human user depending on his or her requirements.

The rest of the paper is as follows. We first look at several well-known classifiers which produce "simple" models. We proceed to describe our algorithm for greedily learning short decision lists. We examine the performance of all algorithms on UCI [4] data, and show our algorithm's superiority over a wide range of the complexity-accuracy trade-off parameter and a variety of inputs.

2 Related Work and Definitions

Some of the very first methods used for machine learning lend themselves easily to human evaluation. These include decision trees [5] and decision lists [6]. But they were not optimized for brevity, and in general may comprise of hundreds or thousands of nodes. Even the typical evaluation of a single unseen instance will require going down a long evaluation sequences from root to leaf. Sokolova et. al [7] address this issue, and even propose parameterization of the accuracy cost like we do. But the basis features they use are inclusion in hyperspheres. This is not only hard for humans to compute, but also presumes the existence of a good distance metric. In addition, only binary classification is discussed, and it seems the obvious multi-class extension would make their models much more complex.

In [8], a learner is given for probabilistic decision lists (where the predicted value is a list of probabilities). It is reported to produce much shorter lists than previous algorithms. But this is a side-effect rather than a goal. Even so, the sizes reported are significantly larger than what is conveniently grasped by a human.

Holte [9] suggested a "one rule" classifier, where a single attribute is used for classification. The rule is a simple lookup table (numeric values are discretized first), and the attribute is chosen to minimize the error. Even though it was meant as a strawman, it was found to perform reasonably well on many UCI datasets.

Typically in the context of boosting, a "decision stump" is learned. This is a single if-else rule, chosen based on the squared error (for numeric classes) or entropy (for nominal classes). Later, an ensemble of many stumps is composed to increase performance.

Arguably, it is easy for humans to perform nearest-neighbor classification, on some graphical plot of the data. The prior assumption is that the data is sparse and also lies in a very low dimensional space. The same can be said on linear separators. Association rules are also very intuitive, but have very low coverage ("support"), and so are produced in very large numbers.

In what follows, we suggest our own quantifier of evaluation complexity. Its main aim is to capture some subjective measure of human difficulty. It counts the number of decision *branches* required in the worst case. Informally, a branch is a single comparison of one data attribute, and results (if matched) in a prediction of a single class. More precisely, a branch must meet the following constraints:

1. A branch contains a single predicate on a single input attribute.

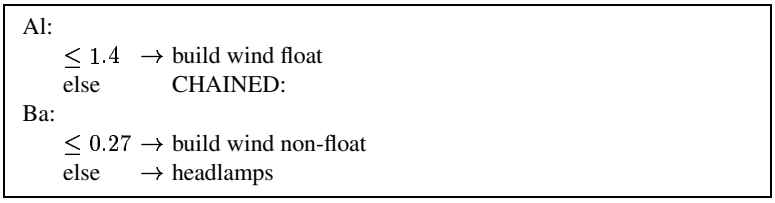

Fig. 1. Example rule of thumb learned.

2. If the predicate is true, a single class may be predicted. Alternatively, another branch could be chained, and its predicated would be AND-ed.
3. The "else" arm may include either a single class prediction, or a branch.

It is evident that decision lists, decision trees and stumps, and other simple learners can all be easily defined in terms of their branch count. We will examine this more deeply in Section 4.

Given the (test set) accuracy a (in percent) and branch count c, our measure is $a - c \cdot w$, where w is a parameter describing the weight we assign to complexity over accuracy. For example, $w = 10$ means we would rather lose 10 percentage points of accuracy than add another branch to the rule. And $w = 1/10$ would mean the opposite. We believe this parametrization makes it simpler for an end-user to understand how the penalty affects actual performance. The parametrization by w stands in contrast to the "one size fits all" nature of BIC. Arguably, it is also easier to elicit than the unit-less penalties proposed by [7]. A similar goal has also been attempted by Atzmueller et. al [10]. Their measure requires three user-supplied parameters (threshold_c, γ, and α), and only broad guidelines are given to determine their value. In addition, their method is computationally inefficient, and the reported results show rulesets with hundreds or thousands of rules. In Section 4 below we show our method typically produces just a handful of rules.

3 Rule of Thumb Learner

Our learner (abbreviated as `TR` below) produces a decision list. See Figure 1 for an example. It works by iteratively learning an single additional branch in each step. The branch has a single equality test for the nominal case, or a threshold test for the numeric case. If the condition is met, a single class is predicted. If not, a default class is predicted. In each step we refine the input by considering only the data covered by the "else" arm of the previous rule learned. We learn a rule for this data, and substitute it for the default branch of the previous rule. Thus, we grow a decision list, one additional rule per step. In each step, each attribute is inspected in turn, and an optimal if-else rule derived for it, as described below. To complete the step, the attribute with the least classification error, and its associated rule, is chosen and committed to.

The stopping criterion is either reaching a user-defined branch count, or when the accuracy of the current list is above a threshold. During list buildup, the BIC score of

each intermediate decision list is measured. The list finally output is the one with the highest BIC score.

We now describe how to find an optimal rule, given an attribute. For nominal attributes, we first populate a count matrix C_{ij} for all attribute values i and target classes j. A single "if" branch corresponds to selecting an element c_{ij} in this matrix, with i being the value to compare against and j being the predicted class. Disregarding the "else" branch, the best choice will produce the greatest number of correct predictions (against the training set). To account for the "else" branch, we also look at the remainder of the data, and its partition into classes. This information is obtained by subtracting the i-th row from the total class counts. We seek to increase the number of correct predictions, and therefore choose the class with the maximal count. The precise algorithm is shown in Figure 2. Note that the choice of class for the "else" branch is independent of the value of the element representing the "if" branch.

For the numeric case, things are slightly more complicated. Again we fix an attribute. Our first goal is to produce a counts matrix like in the nominal case. Here, the rows will correspond to threshold values. We sort the possible values for the attribute in question, and assign a matrix row to each unique value, in increasing order. As before, matrix columns stand for prediction classes. After populating the matrix, the i-th row stores the class counts for the data which have their attribute value less than or equal to the i-th smallest value observed in the training data. Similarly to the nominal case, we can choose the maximal element in the row for the "if" branch. And again, we subtract the row from the total class distribution to get the complement distribution. We choose the best (maximal) element in the complement distribution as before. See Figure 3.

For a single step in the nominal case, we scan the data set to compute the counts. Then, for each attribute, we examine each of its possible values to find a maximum among the target classes. Therefore the complexity due to a single attribute scan in one step is $O(Nk + tk)$, where N is the number of input records, k the maximal number of attribute values, and t the number of target classes. In the numeric case, the operation is preceded by sorting the values. This sums up to $O(N \log N + Nk + tk)$, where k here is the largest number of unique values of any numeric attribute. For the overall time complexity, the numbers above need to be multiplied by the number of attributes, and the number of input records.

4 Experiments

In our experiments below we compared performance against the Weka [11] implementations of the following learners:

- `ZeroR`: A "zero-rule" classifier. Predicts the mean for a numeric target class, and the mode for a nominal target class.
- `OneR`: Holte's one-rule learner.
- `PART`: A greedy decision-list learner [12]. Iteratively, C4.5 trees are grown, and one of their leaves is converted into a rule.
- `DecisionStump`: A decision stump as described above.
- `REPTree`: Weka's fast decision-tree learner.

Input:
D Data instances
A The attribute's values $\in \{1..n\}$
for elements of D
L Instance labels where $L(x) \in \{1..m\}$
for $x \in D$
Output:
v Equality value for the "if" branch
c Predicted class for the "if" branch
d Predicted class for the "else" branch

1. Scan D, populate an $n \times m$ count matrix C where
   $C_{ij} = |\{x \in D | A(x) = i \wedge L(x) = j\}|$.
2. For $i = 1...n$:
   (a) Populate a $1 \times m$ complement vector: $\overline{C}_j = \sum_{k=1}^{n} C_{kj} - C_{ij}$.
   (b) Let $c' = \arg\max\{C_{ij} | j = 1...m\}$
   (c) Let $d' = \arg\max\{\overline{C}_j | j = 1...m\}$
   (d) If $C_{i,c'} + \overline{C}_{d'}$ is greater than any value seen so far:
   Let $v = i$; $d = d'$; $c = c'$

Fig. 2. Branch construction for a nominal attribute.

Given our definition, the branch complexity of the algorithms above is:

- `ZeroR` One.
- `OneR` The number of distinct values in the prediction attribute (nominal), or number of bins (numeric).
- `PART` The number of rules.
- `DecisionStump` Two. Optionally, a third branch is added for handling missing values.
- `REPTree` The number of tree nodes.

A brief description of the datasets in our experiments is shown in Table 1. Based only on such simple statistics on the data, we can already determine that several datasets are particularly hard to classify with a concise rule. These include: LETTER, with 26 equally-sized target classes, OPTDIGITS and PENDIGITS, each with 10 equally-sized target classes, VOWEL, with 11 equally-sized target classes, and SOYBEAN, where the top 5 classes make up 65% of the data, and the remaining 14 classes being roughly equal in size. The difficulty is due to a combination of two factors. First, by definition, the number of predicted labels cannot exceed the number of branches. Second, the ratios of class sizes is such that no small set of classes covers a significant fraction of the data. When these two hold, no algorithm, including ours, can achieve both a short description and high accuracy. Therefore, any reasonable usage scenario will preclude the "simple" algorithms from consideration for these kinds of datasets. But we include them in the experiments for completeness. Removing them from the input improves the results in our favor (data not shown).

Input:

D Data instances

A The attribute's values
for elements of D where
$A(x) \in \{v_1 ... v_n\}$ for $x \in D$

L Instance labels where
$L(x) \in \{1..m\}$ for $x \in D$

Output:

v Upper bound for the
"if" branch comparison

c Predicted class for the
"if" branch comparison

d Predicted class for the
"else" branch

1. Sort D by increasing value of A. Populate an $n \times m$ matrix B where $B_{ij} = |\{x \in D | A(x) = v_i \wedge L(x) = j\}|$.
2. For $i = 1...n$ and $j = 1...m$, let $C_{ij} = \sum_{k=1}^{i} B_{kj}$.
   /* Now
   $C_{ij} = |\{x \in D | A(x) \leq v_i \wedge L(x) = j\}|$ */
3. For $i = 1...n$:
   (a) Populate a $1 \times m$ complement vector: $\overline{C}_j = \sum_{k=1}^{n} C_{kj} - C_{ij}$.
   (b) Let $c' = \arg\max\{C_{ij} | j = 1...m\}$
   (c) Let $d' = \arg\max\{\overline{C}_j | j = 1...m\}$
   (d) If $C_{i,c'} + \overline{C}_{d'}$ is greater than any value seen so far:
   Let $v = v_i$; $d = d'$; $c = c'$

Fig. 3. Branch construction for a numeric attribute.

Table 1. Summary of test datasets.

Datasets	Instances	Attributes	Classes
anneal	898	39	6
arrhythmia	452	280	13
audiology	226	70	24
autos	205	26	6
balance-scale	625	5	3
breast-cancer	286	10	2
colic	368	23	2
credit-a	690	16	2
credit-g	1000	21	21
diabetes	768	9	2
ecoli	336	8	8
glass	214	10	6
heart-c	303	14	2
heart-h	294	14	2
heart-statlog	270	14	2
hepatitis	155	20	2
hypothyroid	3772	30	4
ionosphere	351	25	2
iris	150	5	3
kr-vs-kp	3196	37	2
labor	57	17	2
letter	20000	17	26
lymph	148	19	4
mushroom	8124	23	2
optdigits	5620	65	10
pendigits	10992	17	10
primary-tumor	339	18	21
segment	2310	20	7
sick	3772	30	2
sonar	208	61	2
soybean	683	36	19
vehicle	846	19	4
vote	435	17	2
vowel	990	14	11
waveform-5000	5000	41	3
zoo	101	18	7

We first discuss the raw performance of all algorithms along two "base" axes. For each of the 36 datasets, and for each algorithm, we measured the average accuracy (percentage of correct predictions) and complexity (number of branches) for 10 cross-validated runs. In the results below, we refer to the negative of the complexity, which we call *simplicity*. This way, the direction its optimization (up) matches that of accuracy. For our learner, the upper limit on number of rules was set to 10, and the accuracy stopping criterion was set to 99%. Note that the values we measured are not comparable across datasets (as is generally the case with model selection measures). Therefore, we sorted the values obtained for each dataset, and considered just the ranks of each algorithm. Thus, the best possible score is 1, and the worst is 6. The score we report for a given algorithm is its average rank across all datasets. See Figure 4. There are no surprises here. Algorithms which consistently produce concise models have low accuracy, and the algorithms with the best accuracy also have the most elaborate descriptions.

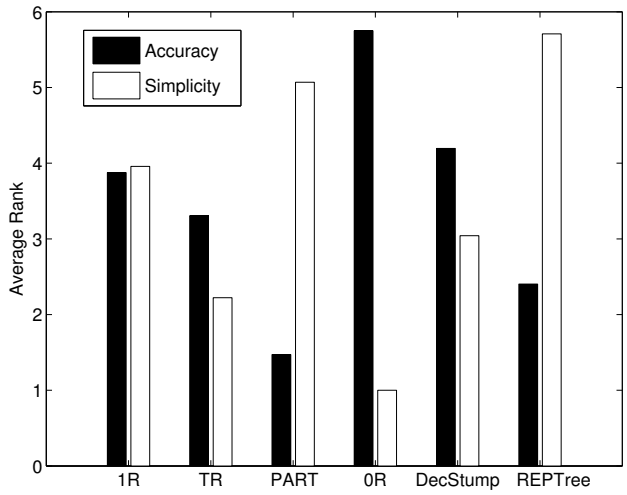

Fig. 4. Raw performance.

Table 2 summarizes the statistics on the number of rules produced by various algorithms on the test data. In particular, observe how the number of rules produced varies highly for `OneR`, `PART`, and `REPTree`, and it is also generally too high for manual evaluation.

It is well-known (and also evident from the data) that accuracy and complexity need to be traded off. In the statistical context, log-likelihood is usually penalized by an increasing measure of model complexity [3]. And the main objective criterion for our algorithm is indeed BIC. However, we chose not to judge its relative performance based on BIC. The first reason is that no other algorithm optimizes directly for BIC, and such a comparison would be unfair. But more importantly, we suggest a new eval-

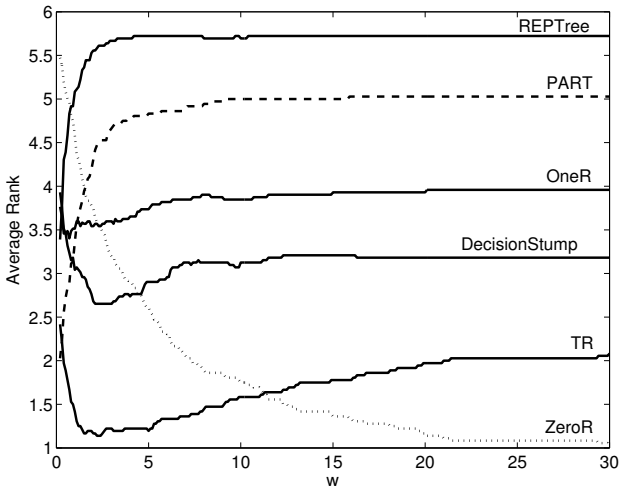

Fig. 5. Combined performance as a function of w.

Table 2. Aggregate number of rules for each algorithm.

Algorithm	Mean	Min	Max	Median
OneR	14.5	2.0	101.5	6.3
TR	2.05	1.0	6.0	2.0
PART	45.5	3.4	733.2	18.8
REPTree	80.9	1.0	1212.0	30.3

uation measure. Our measure has one degree of freedom, which can be interpreted as the importance one places on the simplicity of the learned model. Depending on the case, a user could prefer accuracy or simplicity by setting the value of the parameter appropriately.

Our main result (Figure 5) is based on our penalized accuracy measure. For each algorithm and dataset, we computed our performance measure, and considered the average ranks as above. We did this for values of w in the range 0.2–30, and plotted the results as a function of w[1]. As can be seen, our algorithm is superior throughout the range 0.3–11.2, with `ZeroR` and `PART` dominating it in the obvious places.

To establish statistical significance for the data in Figure 5, we performed Wilcoxon signed-ranks tests [13], comparing our algorithm and each of the five others, for each value of w. At the $p < 0.01$ significance level (after applying Bonferroni correction for five tests), our algorithm is superior throughout the range 0.8–7.0 for w. If we instead

[1] The lines balance off for $w > 15$; this region has been cut out for clarity.

use the Friedman test and apply the Bonferroni-Dunn correction [13], we establish significance throughout the range 0.7–5.5. The (conservative) interpretation is that in those scenarios where loss of up to 5 percentage points of accuracy is preferred to the addition of a single decision rule, our method should be used.

5 Conclusion

This work examines the model selection problem from a human-cognitive aspect. We suggest a new complexity-accuracy trade-off designed to support decision making. It is parameterized in a way that makes it very clear what is the price (in terms of complexity) one is willing to pay for increased accuracy. This, too, fits our philosophy of simplicity, as can be judged by non-technical users. Additionally, we propose a new and very computationally-efficient algorithm for the learning of "rules of thumb", and evaluate its performance. We show it reliably produces good classifications with very few rules, while being flexible enough to allow for more complex modeling where needed.

Our weighted penalty measure is a function of an algorithm's black-box performance, and so does not require calculation of log-likelihoods. This property makes it a preferred alternative where log-likelihoods are not easily computed (`ZeroR` and `OneR` come to mind). But in general, future work should more deeply inspect the relationship between BIC and our measure. Another interesting aspect would be an algorithm that takes the complexity weight w as input, and produces a learner optimized for the induced cost function, similarly to the way [7] incorporates the loss function.

Admittedly, rules of thumb can only work in easy cases. We do not aim to address scenarios where the number of significant classes is large, where the important features are complex functions of the input attributes, or even with XOR-type difficult inputs. The scope of this work is the assumption that a (presumably human) consumer of the classification rules is after a rule that is easy and cheap to execute. Where this does not hold, classifiers with greater power, or ensembles of simple classifiers, should be used. To the best of our knowledge, no similar work has attempted to cover this space.

We present performance results for several well-known algorithms. These can be used to guide the selection of a particular classification algorithm where rule brevity is sought. Given the desired value of w, one can select the best-performing algorithm. This, too, acts as a rule of thumb — this time, for selecting the classification algorithm.

References

1. Gigerenzer, G., Selten, R., eds.: Bounded Rationality: The Adaptive Toolbox. The MIT Press (2001)
2. Breitgand, D., Henis, E., Shehory, O.: Automated and adaptive threshold setting: Enabling technology for autonomy and self-management. In: The 2nd IEEE International Conference on Autonomic Computing. (2005)
3. Wasserman, L.: Bayesian model selection and model averaging. Technical Report TR666, Carnegie Mellon University, Pittsburgh, PA (1997) Also available from `http://www.stat.cmu.edu/www/cmu-stats/tr/tr666/tr666.html`.
4. Blake, C., Merz, C.: UCI repository of machine learning databases (1998) `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

5. Mitchell, T.: Machine Learning. McGraw Hill (1997)
6. Rivest, R.L.: Learning decision lists. Machine Learning **2**(3) (1997) 229–246
7. Sokolova, M., Marchand, M., Japkowicz, N., Shawe-Taylor, J.: The decision list machine. In: Advances in Neural Information Processing Systems 15, Cambridge, MA, USA, MIT-Press (2003) 921–928
8. Goodman, J.: An incremental decision list learner. In: Conference on Empirical Methods in Natural Language Processing, University of Pennsylvania, Philadelphia, PA, USA (2002)
9. Holte, R.C.: Very simple classification rules perform well on most commonly used datasets. Machine Learning **3** (1993) 63–91
10. Atzmueller, M., Baumeister, J., Puppe, F.: Quality measures for semi-automatic learning of simple diagnostic rule bases. In: Proceedings of the 15th International Conference on Applications of Declarative Programming and Knowledge Management (INAP 2004), Potsdam, Germany, 2004 (2004) 203–213
11. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2nd edn. Morgan Kaufmann, San Francisco (2005)
12. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. In Jude Shavlik, ed.: Proceedings of the Fifteenth International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA (1998)
13. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7** (2006) 1–30
14. Dhagat, A., Hellerstein, L.: PAC learning with irrelevant attributes. In: Proceedings of the 35rd Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA (1994) 64–74
15. Bohanec, M., Bratko, I.: Trading accuracy for simplicity in decision trees. Mach. Learn. **15**(3) (1994) 223–250
16. Giordana, A., Neri, F.: Search-intensive concept induction. Evolutionary Computation **3**(4) (1995) 375–419
17. Furnkranz, J., Flach, P.: An analysis of rule evaluation metrics. In: Proc. 20th International Conference on Machine Learning (ICML'03), AAAI press (2003) 202–209