Bounded-Error Preclassification Trees

Henry S. Baird
Colin L. Mallows

AT&T Bell Laboratories
600 Mountain Avenue, Room 2C-322
Murray Hill, NJ U.S.A.

Abstract

We discuss an automatic method for constructing high-performance preclassification trees. The role of preclassifiers is to prune the number of classes to a fraction of the total (by contrast, classifiers pick exactly one class). Decision trees make fast preclassifiers; if they also prune strongly, they can increase speed of the overall system significantly when used with classifiers whose runtime increases with the number of classes. It is an irrecoverable error if the true class is pruned: unfortunately, such errors accumulate rapidly in decision trees. We attack this problem through massive simulation of precisely defined classification problems using a pseudo-random generator of labeled samples. We first build a tree using greedy entropy minimization, using as many training samples (*e.g.* 1.1M) as computing resources permit. This tree is then ‘‘populated’’ with many more samples (*e.g.* 180M), driving the error rate down to an acceptable level. We present statistically rigorous *stopping rules* for decision-tree population to enforce user-specified upper bounds on error, and illustrate their behavior in large-scale OCR experiments.

Keywords: decision tree, classification, character recognition, image defect models, learning, population

1. Introduction

Non-backtracking decision trees promise fast classification. Inferring optimal trees, under various measures, has been shown to be computationally infeasible [Bun87] in the worst case. However, in practice on OCR problems, experiments show that suboptimal heuristics often build roughly balanced, strongly pruning trees [CN84,WS87]. Many such heuristics — including the one we use — have the serious practical drawback that error rapidly accumulates as the tree deepens, when using a training set of fixed size.

In contrast with much prior work on classification trees, we stress their role as preclassifiers in a multi-stage decision strategy without backtracking. Preclassification trees prune the set of classes to a smaller number: that is, each leaf of the tree owns a subset of the classes, not always only one class. A preclassification tree makes an error if it prunes the true class. For simplicity of

Submitted for publication in *Proceedings, IAPR Int’l Workshop on Syntactic and Structural Pattern Recognition (SSPR’94)*, Nahariya, Israel, October 4-6, 1994.

analysis, and somewhat pessimistically, we will assume that these errors cannot be corrected by later stages of classification or contextual analysis. A more discriminating but slower classifier, executed downstream, will thus be presented with fewer classes to be distinguished; often, it can then run faster. In our case, runtime of the classifier is approximately linear in the number of classes. In this context, a good preclassifier is one that rarely prunes the true class but otherwise prunes as many classes as possible.

It is often possible to construct shallow preclassification trees with an acceptably low probability of error. However, trees that are deeper and more strongly pruning — and thus offering greater speed-up — often exhibit unacceptable error. This suggests the possibility that the essential problem is not that the greedy tree-building heuristic is sub-optimal: it is that the training data is too sparse. In this view, tree-building rapidly becomes unreliable as the tree deepens because the data surviving at the leaves rapidly becomes sparse.

This conjecture has motivated a series of experiments in which sample-image generators are used in a brute force manner to drive down the error rate of decision trees. In an earlier stage of this research [Bai93], we built preclassification trees for 100 typefaces of the 94 ASCII symbols, achieving an error rate of less than 1.0% and a pruning factor of $\times 5.2$, on type sizes 10 point and higher at 300 pixels/inch (ppi).

We now describe a method that achieves almost an order of magnitude lower error rate (0.15%) with about half the pruning factor ($\times 2.7$), on an OCR problem which is harder in some respects (the alphabet is over twice the size) and easier in others (20 typefaces are used rather than 100). We used fewer typefaces in this trial because our new method requires much more computation, in its present stage of development. We hope eventually to expand these experiments to at least the scale of the earlier trials.

In both trials, we built trees using a greedy entropy-minimization heuristic, using as many sample images as our computing resources comfortably admit (in this latest trial, 1,066,639 samples). In the earlier trial, we ‘‘populated’’ the resulting tree with another sample set of fixed size, and then measured the error rate. A drawback of this approach was that neither the error rate nor the pruning factor could be confidently predicted in advance.

In the present trial, we populate the tree with a sequence of sample images generated on the fly, as many as are required to drive the error rate below a user-specified target. Thus we can *guarantee an upper bound on the error rate*, assuming of course that the training set is representative. The pruning factor, however, remains difficult to predict. The essential new technology is a statistically motivated *stopping rule* for tree-population.

We describe the engineering context in Section 2, and the derivation of a stopping rule in Section 3. The experimental trial is described in Section 4. Section 5 contains conclusions and discussion.

2. The Engineering Context

We now briefly sketch the engineering context of this work, including the application, the source of image samples, features and classifiers, binary decision trees, and building and testing the trees.

2.1. The Application

The context of this effort was a project to build a classifier subsystem for a family of machine-print page readers for ten European languages [BGI94]. This required distinguishing 209 character symbols (all of ASCII and Latin-1, plus a few Turkish symbols), in 20 typefaces, over the range of type sizes 8-12 point (at 300 ppi).

2.2. Image Samples

We used a quantitative model of imaging defects [Bai92] with parameters for type size, spatial sampling rate (digitizing resolution), blur, binarization threshold, pixel sensitivity variations, jitter, skew, stretching, height above baseline, and kerning. Associated with it is a pseudo-random defect generator that reads one or more sample images of a symbol — in this trial, these are high-resolution noiseless artwork purchased from typeface manufacturers — and writes an arbitrarily large number of distorted versions exhibiting a user-specified distribution over the model parameters. These distributions have been roughly calibrated on image populations occurring in printed books and typewritten documents. During training, including tree-building, we use only synthetic images. We test on both synthetic data (for consistency checking and debugging) and on real data from printed and scanned paper documents.

2.3. Features and Classification

The classifier technology used here is described in [Bai88]. Briefly, it extracts local geometric shapes from the input image of an isolated symbol, maps this diverse collection of shapes into a feature vector with binary components. This binary feature vector is designed to be insensitive to location and type size; the preclassifier will examine only this vector. From a training set labeled with true class, type size, etc, we infer a single-stage Bayesian classifier under an assumption of class-conditional independence among the features. This is the ‘‘main’’ classifier for which we need a preclassifier. Its runtime is $O(\ C\ (F + \log C)\ )$, where C is the number of classes to be distinguished at runtime, and F the number of binary features.

In this trial, $C\ =\ 4032$ classes (one for each <typeface,symbol> pair for which we have artwork), and $F\ =\ 704$ features.

2.4. Decision Trees

At each node of our decision trees, a single feature is tested. Thus these are binary decision trees. Each leaf of the tree contains a subset of the classes. An input image, represented by its feature vector, is said to be correctly preclassified if the leaf it arrives at contains its true class.

2.5. Building a Tree

The tree-growing heuristic is to execute a sequence of *splits* (<leaf,feature> pairs) until a stated pruning factor is reached. At each step, examine all possible next splits and choose the one which most decreases the expected entropy of the tree. The tree’s expected entropy and pruning factor are estimated on the assumption that the distribution of the training data is representative: this depends on the validity of the image defect model. The method is greedy, with no look-ahead: multiple splits are not examined at each step.

Even with the short-cuts of this sub-optimal heuristic, the large scale of our problem strained our computing resources. We use a Silicon Graphics Computer Systems Challenge XL, with 150MHz R4.4k processors, running time-shared UNIX. In order for the program to terminate in reasonable time, it was necessary throughout the tree-building phase to hold in main memory all training data and the growing tree itself. As a result, the number of training samples was effectively limited to about a million: precisely, 1,066,639, or 3/4 of the number used to build the Bayesian classifier. After 32.5 CPU hours of tree-growing, the pruning factor reached ×9. The ×9 tree contained 932 decision nodes and 933 leaves. Before the pruning factor could reach ×10, the size of the running process exceeded 512M bytes, a hard system limit, and the run terminated abnormally.

2.6. Testing the Tree

The trees are perfect on the training set by construction. Testing on a distinct set reveals, in general, non-zero preclassification error and a pruning factor different from (but often very close to) the one estimated during training.

In fact, a 15% error rate was revealed by a test on 1,030,000 synthetic test samples: 1000 for each <typeface,size,symbol> triple for the single typeface Avant Garde-Book Oblique and the five integer type sizes 8-12 point. In the same test, the pruning factor was measured as ×9.3, slightly higher than on the training data.

2.7. Populating the Trees

In ''population'' of the tree, we use a sample set distinct from the training set to correct the contents of the leaves without changing the structure of the tree otherwise. Each sample is tested: if the leaf does not contain the true class (the tree fails), then that class is added to the leaf. Every time a leaf is corrected, the error rate of the tree drops — but its pruning factor also drops.

In this trial, we populate using a sequence of samples generated on the fly until the error rate is driven down to a user-specified target.

3. A Stopping Rule for Population

We wish to populate the tree until the target accuracy has been reached or exceeded, with high statistical confidence, and at the same time minimizing the number of samples generated. Choosing such a *stopping rule* is complicated by our ignorance of the rate at which the accuracy is increasing. We proceed to formalize the problem as follows.

Let us model the execution of the population algorithm as a stochastic source S_0 that writes a sequence {0,1}*, e.g.:

11110111111101111...

where '1's occur randomly independently with initial probability p_0. The randomness is provided by the pseudo-random image generator; '1's represent the event that the true class is found at the leaf, and '0's represent tree failures. This is a sequence of Bernoulli trials as long as p_0 is constant.

However, in our case the probability of success is not constant. Whenever a failure occurs, the source is modified so that the success rate increases. Thus, after the first '0' is seen, S_0 is modified, giving a new stochastic source S_1 whose probability of success is $p_1 > p_0$. This occurs at every failure, so that $\{S_0, S_1, ..., S_i, ...\}$ runs a sequence of trials in which the success rate increases monotonically whenever i increments.

In our case the probability of success rises asymptotically to 1, and will eventually attain 1, since the number of errors that can occur is bounded above by a constant (the number of leaves in the tree times the number of classes). Other than this, however, we know nothing about the rate of growth of p_i.

We want to stop the trials as soon as possible after the success rate exceeds a specified target $t < 1$ (say 0.99). Since we can never be sure that the target has been reached, we ask for a stopping rule that makes an error only 5% of the time, so that with 95% confidence we can assert that the target has been reached.

A naive rule is to choose some number k and to stop as soon as a success-run of length k is seen, asserting that the current value of p is at least t. The worst case for this rule is when there are a very large number of p's that are increasing very slowly, and all just less than the target t. In this case, no matter how k is chosen, at each stage there is a probability of just less than t^k that a success-run as long as k will occur, and eventually one of these events will indeed occur; thus with very high probability we will stop with p still less than t, and the assertion will be false every time. Thus the naive rule does not have the desired 95% confidence.

One way to achieve a proper confidence procedure, *i.e.* one where the probability of stopping with $p < t$ is at most α, is to ensure that in the limit of the situation described above, *i.e.* where there are infinitely many p's all just less than t, we have probability $1 - \alpha$ of never stopping at all.† Can we construct a stopping rule with the property that when there are infinitely many p's just less than t, we stop with probability at most α? We can do this by letting the required lengths of the success-runs increase. Suppose we stop as soon as one of the following events happens:

$$n_1 = k_1$$

$$n_1 < k_1 \text{ and } n_2 = k_2$$

$$n_1 < k_1 \text{ and } n_2 < k_2 \text{ and } n_3 = k_3$$

$$\ldots$$

where n_j is the length of the jth success-run. Then if we take

$$k_j = (2 * log(j) + c)/log(1/t) \tag{1}$$

we will have

$$P(\text{ never stop } \mid p_1 = p_2 = \cdots = t\,) \;=\; \prod_{j=1}^{\infty} P(n_j < k_j) \;=\; \prod_{j=1}^{\infty} (1 - t^{k_j})$$

and this product is convergent by the standard test:

$$\sum_{j=1}^{\infty} t^{k_j} \;=\; \sum_{j=1}^{\infty} \frac{e^{-c}}{j^2}$$

which is convergent. By changing c we can make the product anything we like between 0 and 1; *e.g.* taking $c = 3$, for $t = .99$ the k_j are (rounding up):

$$298,\ 436,\ 517,\ 574,\ 619,\ 655\ \ldots$$

and

$$\prod_{j=1}^{\ldots} (1 - t^{k_j}) \;=\; 0.950,\ 0.938,\ 0.933,\ 0.930,\ 0.928,\ 0.927\ \ldots$$

These values converge to 0.92. So this gives a procedure with the property that if we stop, and assert that $p \geq t$, we will make a mistake with probability at most 0.08. If the p's never got above t, we would keep on sampling indefinitely (but, as we have noted above, this will not happen in practice since in reality there are only a finite number of p's).

We implement this rule by allowing the user to specify at runtime the target probability of success t and the confidence $1 - \alpha$. From these, the program determines c by solving numerically for

$$\prod_{j=1}^{\infty} (1 - t^{k_j}) \;=\; 1 - \alpha \tag{2}$$

Then, at each failure, j is incremented, k_j is computed using Eqn. (1), and population is stopped if the jth unbroken run of successes exceeds k_j in length.

Note that we have not made use of any assumption about the initial success rate p_0 or the rate of increase of the p's. If we knew something more, the stopping rule could perhaps be made more efficient. For example, if we knew somehow that at most k_0 p's were $< t$, we could simply go on until we see k_0 0's, but this procedure would be invalid if the assumption were wrong.

† There are some papers by Robbins *et al.* relating to ''tests of power one,'' that have a similar flavor, but they are concerned with i.i.d. observations and a single parameter, and do not seem to have anything to do with our problem.

4. Experimental Trials

In our trials, $t = 0.99$, $(1-\alpha) = 0.95$, $c = 3.472$, and the k_j's are:

$$346,\ 484,\ 565,\ 622,\ 666,\ 702,\ 733,\ 760,\ 783,\ 804\ \ldots$$

We apply the stopping rule independently to each <typeface,symbol> pair letting type size vary randomly uniformly in the interval [7.5,12.5]. Thus we expect that the error rate of the tree will be < 1% for at least 95% of the <typeface,symbol> cases. Of course the error rate may be lower, perhaps by a large margin, and perhaps more often. The pruning factor is unpredictable, at our current level of understanding.

We populated trees for each symbol of nine of the 20 typefaces: each of these trees is specific to a <typeface,symbol> pair. Then for each typeface, we merged their symbols' trees into one by computing the set union of their leaf contents, giving a tree populated for the entire typeface. Each of these trees were tested using both synthetic and real data.

The results for a typical typeface, Avant Garde-Book Oblique, are as follows. Populating the tree required 8,161,999 samples. On 47,511 (0.58%) of these, the tree failed when they were first seen. When tested using a distinct set of 1,005,000 synthetic samples (1000 for each <typeface,size,symbol> triple for the five sizes 8-12), the error rate was 0.15% overall. The effective pruning factor on the same test data is ×2.58 (the measured pruning factor was ×8.2, which is artificially high since the other typefaces had not yet been populated). Similar results were achieved for each of the other eight typefaces, when tested separately on synthetic data.

In another test, we used ''real'' data: 600 pages printed and then scanned (at 400ppi) with text in each of ten European languages, twenty typefaces, and three type sizes (8, 10, and 12 point). For this test, the nine populated trees were combined into a single tree in the fashion described above. The test exercised not only the classifier, but also other stages of a complete page reader including geometric layout analysis, shape-directed resegmentation, and contextual analysis (both typographic and linguistic). On the nine typefaces for which the tree had been populated, the error rate *decreased* by 5.3%, surprisingly. Unsurprisingly, on the eleven typefaces which had *not* been populated, the error rate increased, by 25.6%. The speed-up of the entire page reader was ×2.5, on average; this can be expected to drop somewhat when all of the typefaces are populated.

It is gratifying but unexpected that use of the decision tree resulted in a *lower* error rate of the page reader overall, on those typefaces which were populated. We don't fully understand why: it may be an artifact resulting from mixing populated and unpopulated typefaces in the same tree. In any case, it suggests that our estimates of the extra errors due to the decision tree, which we measured using synthetic data, may be biased high: perhaps the synthetic data is harder than the real data used in the tests.

We project that populating all 20 typefaces in this way will require about 180 million samples.

5. Discussion

We have described a method for building fast preclassification decision trees which guarantee an upper bound on the extra error that they contribute to a multi-stage decision procedure. The essential technical device is a statistically motivated stopping rule for tree-population. Our experimental trials show that this rule can effectively enforce a 1% upper bound on error. In fact it went further, achieving an average error rate of 0.15%, 1/7th of the target. The tree sped up the execution of the classifier by a factor of ×2.5.

This stopping rule is conservative, requiring the population program to generate more than the minimum number of samples required to achieve the error bound. In future investigations along these lines, we will look for stopping rules that guarantee error bounds more tightly, if possible while maximizing pruning factor and minimizing the number of image samples required.

6. Acknowledgement

Stimulating conversations with Tin Ho and David Ittner are much appreciated.

7. References

[Bai88] H. S. Baird, ‘‘Feature Identification for Hybrid Structural/Statistical Pattern Classification,’’ *Computer Vision, Graphics, and Image Processing*, Vol. 42, No. 3, June 1988, pp. 318-333.

[Bai92] H. S. Baird, ‘‘Document Image Defect Models,’’ in H. S. Baird, H. Bunke, and K. Yamamoto (Eds.), *Structured Document Image Analysis*, Springer-Verlag: New York, 1992, pp. 546-556.

[Bai93] H. S. Baird, ‘‘Document Image Defect Models and Their Uses,’’ *Proc., IAPR 2nd ICDAR*, Tsukuba, Japan, October 20-22, 1993.

[BGI94] H. S. Baird, D. Gilbert, D. J. Ittner, ‘‘A Family of Eurpoean Page Readers,’’ [submitted for publication in] *Proc., IAPR 12th ICPR*, Jerusalem, Israel, October 9-13, 1994.

[Bun87] W. Buntine, ‘‘Learning Classification Trees,’’ *Statistics and Computing*, vol. 2, 1992, pp. 63-73.

[CN84] R. G. Casey and G. Nagy, ‘‘Decision Tree Design Using a Probabilistic Model,’’ *IEEE Trans. Information Theory*, Vol. IT-30, No. 1, Jan. 1984, pp. 94-99.

[WS87] Q. R. Wang and C. Y. Suen, ‘‘Large Tree Classifier with Heuristic Search and Global Training,’’ *IEEE Trans. PAMI*, **PAMI-9**, No. 1, Jan. 1987, pp. 91-102.