INDUCTIVE LEARNING WITH BCT

Philip K. Chan

CUCS-451-89

August, 1989

Department of Computer Science
Columbia University
New York, NY 10027
pkc@cs.columbia.edu

Abstract

BCT (Binary Classification Tree) is a system that learns from examples and represents learned concepts as a binary polythetic decision tree. Polythetic trees differ from monothetic decision trees in that a logical combination of multiple (versus a single) attribute values may label each tree arc. Statistical evaluations are used to recursively partition the concept space in two and expand the tree. As with standard decision trees, leaves denote classifications. Classes are predicted for unseen instances by traversing appropriate branches in the tree to the leaves. Empirical results demonstrated that BCT is generally more accurate or comparable to two earlier systems.

The workshop version of this paper is in *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 104-108), June 30 - July 2, 1989, Ithaca, NY: Morgan Kaufmann.

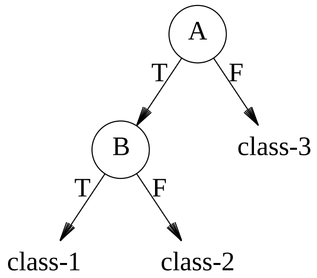
INTRODUCTION

BCT (Binary Classification Tree) is a system that learns from examples. Given a set of preclassified instances, it searches for logical descriptions that accurately represent these classes. BCT is an attempt to unify valuable features exhibited by similar systems. As in AQ (Michalski, 1973) BCT extensively searches for concept descriptions that are consistent with the training examples. Like ID3 (Quinlan, 1986) the system partitions the concept space recursively to identify the most effective class descriptions. Similar to CN2 (Clark & Niblett, 1989) it uses a polythetic approach (based on multiple attribute values instead of single ones) in evaluating the quality of concepts. Moreover, noise-free examples can rarely be obtained and are not assumed during the inductive process. Most importantly, statistically-sound heuristics are incorporated in BCT to guide the search in an accurate and precise manner. In many ways BCT can be viewed as a combination of ID3 and CN2: the concept space is hierarchically partitioned using statistical measures to guide CN2-like searches at each tree level.

In this paper we describe BCT in terms of its knowledge representation, learning operators, and heuristic measures. Empirical comparisons reveal that BCT compares favorably with ID3 and CN2. Finally, related systems and directions for future work are discussed.

KNOWLEDGE REPRESENTATION

BCT represents concept knowledge as a binary polythetic decision tree. Each tree node is a *complex*, which is a combination of attribute values, called *selectors*, in conjunctive normal form (CNF). For example, [(surface = hard) and (color = red or blue)] and [(color = red)] are complexes and (surface = hard), (color = red or blue), and (color = red) are selectors. The two branches coming out from a node represent matching and non-matching of the node's complex (without loss of generality, assume that the left branch matches the complex and the right one does not). Essentially, a complex divides the concept space into two, branching on the complex. The leaves of a tree are classes present in the preclassified examples. Thus, all the leaves (classes) are described by conjunctions of complexes or their negations on the corresponding traversed path. For example, the tree



has three classes: A ˆ B describes class-1, A ˆ ˜B describes class-2, and ˜A describes class-3, where A and B are complexes. If more than one leaf has the same class, the class is denoted by a disjunction of the leaves' descriptions. For instance, if class-3 were replaced by class-2 in the above tree, ((A ˆ ˜B) v (˜A)) would describe class-2.

The classification of an instance can be predicted by comparing the instance's description with the tree nodes and traversing the appropriate branches. When a leaf is reached, the class prediction is the classification at the leaf.

LEARNING OPERATORS

Mitchell (1982) has characterized concept learning as a search task. In his *version space* approach search proceeds in two directions: general to specific and specific to general. Similarly, BCT searches in both directions. It commences with all valid single attribute-value complexes (for example, [(color = red)] and [(surface = soft)]) and then iteratively specializes and generalizes an appropriate subset of these complexes.

BCT's learning operators were inspired by CN2. Our system uses two operators to specialize a complex: *add-selector* and *drop-value.* Add-selector inserts a selector whose attribute is not already present in the complex and drop-value deletes one of the values in a selector. For example, let the complex [(color = red or blue) and (surface = hard or soft)] be *Cmp. Cmp* could be specialized to [(color = red or blue) and (surface = hard or soft) and (legs = 1, 2, or 4)] by add-selector and [(color = red) and (surface = hard or soft)] by drop-value. Similarly, BCT uses two operators to generalize a complex: *drop-selector* and *add-value.* Drop-selector deletes a selector from the complex, and add-value inserts an absent valid value into a selector. That is, *Cmp* could be generalized to [(color = red or blue)] by drop-selector and [(color = red, blue, or green) and (surface = hard or soft)] by add-value.

HEURISTIC MEASURES

An evaluation function, *quality*, is used to guide the search through the space of complexes. It measures how effective a complex is in partitioning the concept space in two. When a complex is evaluated, two frequency distributions of classes in the training examples are calculated—one satisfies the complex (*t-dist*) and the other does not (*f-dist*). Kullback's *estimated discrimination information statistic*, $2\hat{I}$, (Kullback, 1959) is then adapted to measure how dependent *t-dist* and *f-dist* are on the binary values of the complex (in other words, how different the two distributions are). The more dependent the distributions are, the more effective the complex is. In terms of a two-row frequency matrix, *f*, the first row in *f* is *t-dist* (f_{0j}'s) and the second row is *f-dist* (f_{1j}'s). The quality function, $2\hat{I}$, is then defined as:

$$2\sum_{i=0}^{r-1}\sum_{j=0}^{c-1} f_{ij}\log\frac{Nf_{ij}}{f_{i.}f_{.j}}$$

where the number of rows (2: *t-dist* and *f-dist*) and columns (classes) are *r* and *c*, the row and column sums are denoted $f_{i.} = \sum_{j=0}^{c-1} f_{ij}$ and $f_{.j} = \sum_{i=0}^{r-1} f_{ij}$, and the total number of training examples, *N*, is $\sum_{i=0}^{r-1}\sum_{j=0}^{c-1} f_{ij}$.

$2\hat{I}$ is approximately χ^2 distributed with $(r-1)(c-1)$ degrees of freedom. With a user-specified *confidence threshold* (a percentage indicating the certainty that the two distributions are dependent) and the calculated degree of freedom, the corresponding *quality threshold* is determined. A complex's distributions are considered dependent and subject to further scrutiny if its quality is above the quality threshold.

BCT also uses the quality function to predict whether a complex can lead to more effective complexes after the learning operators are applied. The prediction is based on whether the quality of the best conceivable specialization or generalization, *potential quality,* exceeds the computed quality threshold. For the distributions [*t-dist*: (3 1 1 1), *f-dist*: (1 4 1 2)], the best specialization is [(4 0 0 0), (0 5 2 3)] and the best generalization is [(4 0 2 3), (0 5 0 0)]. In effect, BCT provides a one-step look-ahead and tries to avoid further examination of non-fruitful complexes.

ALGORITHM

BCT iteratively searches for a complex which is most effective in partitioning the given training examples. The examples are then split into two sets according to the complex and BCT is recursively applied on those sets.

Initially, Find-best-complex() starts off the search with a *pool* of all valid single attribute-value complexes (Figure 1.). For each of these complexes, its *quality* and *potential quality* are calculated. If the quality of a complex is better than the most effective one found so far, the complex becomes *best-complex*. If a complex's quality and potential quality are below the *quality-threshold*, it is removed from the complex pool. According to the user specified *beam-size* (*breadth*), BCT retains a fixed number of top quality complexes in the pool and discards the rest. The retained complexes are then specialized and generalized producing a new pool of complexes. Again, the system evaluates the new complexes and repeats the same procedure until the user specified iteration limit, *depth*, is reached or the pool is empty. Limited breadth and depth are introduced to avoid extensive computation and exhaustive searches.

Once Find-best-complex() identifies the complex with the best partition, Build-BCT() uses the complex to divide the training set into two disjoint subsets: one satisfies the complex (*t-set*) and the other one does not (*f-set*). The algorithm is then recursively applied on the subsets creating two subtrees. If Find-best-complex() cannot find a satisfactory partition, according to the quality threshold, Build-BCT() returns the class with the highest frequency in the training set, which forms a leaf of the final binary decision tree.

```
Find-best-complex(training-examples)

    best-complex = nil
    pool = all valid single attribute-value complexes
    while (depth ≠ 0 and pool ≠ empty)
        for each complex
            calculate its quality and potential quality
            if the qualities are lower than quality threshold
                remove the complex from pool
            endif
            if quality is the the best found so far
                best-complex = the current complex
            endif
        endfor
        select the top beam-size complexes
        pool = specialization and generalization of the selected complexes
        depth = depth − 1
    endwhile
    return best-complex


Build-BCT(training-examples)

    complex = Find-best-complex(training-examples)
    if complex is not nil
        use complex to split training-examples into t-set and f-set
        return complex, Build-BCT(t-set), and Build-BCT(f-set)
    else
        return the class with the highest frequency in the training set
    endif
```

Figure 1. Algorithm.

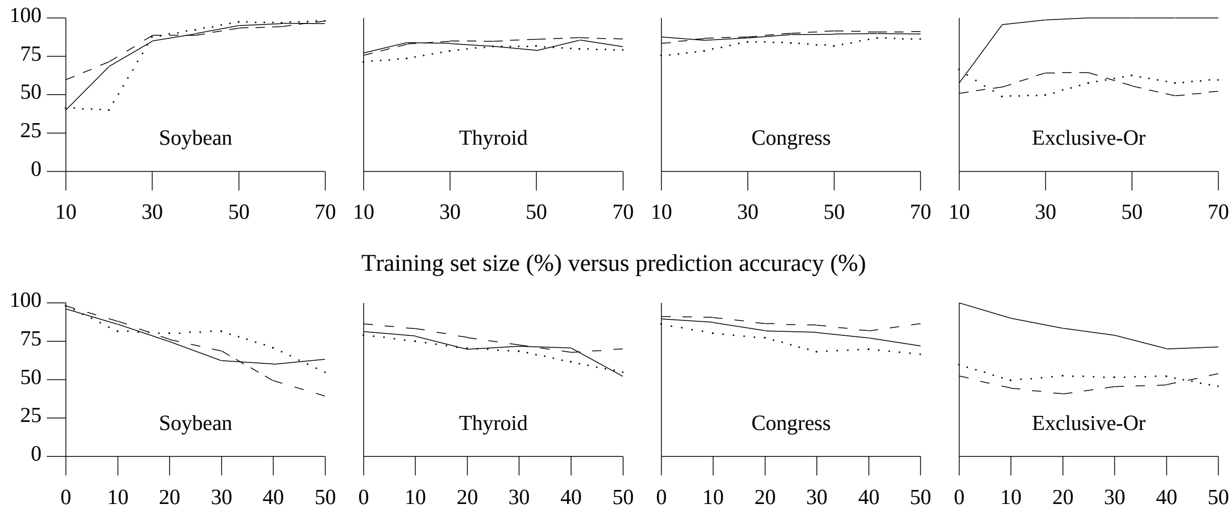
The cost of the whole algorithm is about $O(abde(b+e))$ where a is the number of attributes, b is the beam size, d is the depth, e is the number of training examples, and constant number of values per attribute is assumed. In Find-best-complex(), $O(ab)$ complexes are generated as $O(ab)$ learning operators are applied. It takes $O(e \times ab)$ time to evaluate each complex with the training examples and $O(b \times ab)$ time to compare the top *beam-size* complexes. One loop in Find-best-complex(), therefore, takes $O(abe+ab^2)$ time. Since Find-best-complex() has d loops, it takes $O(abd(e+b))$ time. The generated tree has $O(e)$ nodes (complexes) so BCT takes $O(abde(b+e))$ time. If e is much larger than b, BCT's cost is reduced to $O(abde^2)$.

EMPIRICAL RESULTS

BCT was tested on data sets from three natural domains and one artificial domain: *soybean disease case histories* (courtesy of Robert Stepp), *thyroid disease case histories* (courtesy of Ross Quinlan), *congressional voting records*, and the *exclusive-or function*. For comparative purposes, reimplementations of CN2 (reimplemented by Chan, 1988) and ID3 (courtesy of Doug Fisher) were also tested on the same data sets. For each domain, we divided the data into two disjoint subsets, *training* and *testing*. The experiments used data in the training set to generate a binary classification tree and those in the testing set to test prediction accuracy (percentage of correct classifications) of the tree discovered in training. Random selection from the initial data set determined the training and testing sets. In addition, noise was introduced to the data by randomly changing the classifications of a predetermined percentage of the training instances. Experimental results presented were averages of ten learning trials each.

A set of experiments was conducted in which the training set size varied from 10% to 70%. Confidence threshold was set at 90%, beam size at 5, and depth at 10 (similar settings for ID3 and CN2). As expected, prediction accuracy generally increased with the training set size. In the natural domains the prediction accuracy of BCT was comparable to those of ID3 and CN2 (Figure 2). In the exclusive-or domain, however, BCT clearly surpassed the other two systems. BCT began and continued to achieve perfect accuracy with a training set size of 40%, while CN2 and ID3 could only achieve 65% accuracy in their best trials.

Another set of experiments was performed in which the amount of noise varied from 10% to 50% (it becomes less meaningful if more than 50% of the training data are noisy). Training set size was set at 70%, confidence threshold at 90%, beam size at 5, and depth at 10 (similar settings for ID3 and CN2). As expected, the presence of noise did not cause detrimental decrease in BCT's performance. In the natural domains BCT's performance was generally comparable to ID3 and CN2 (Figure 2). In the exclusive-or domain, however, BCT undoubtedly surpassed ID3 and CN2.



Amount of noise (%) versus prediction accuracy (%)
[—— : BCT, - - - : ID3, and .... : CN2]

Figure 2. Empirical comparisons.

DISCUSSION

The poor performance of ID3 in the exclusive-or domain was mainly due to its monothetic evaluation approach. In the artificial domain classifications are based on the exclusive-or function of two binary attributes (the first two in our experiments). Since classifications depend on the combination of the two attributes, values of either one of them cannot partition the data adequately. Thus, selecting either attribute will not result in a sizable *information gain* (Quinlan, 1986) in ID3. On the other hand, the polythetic evaluation approach allows BCT and CN2 to discover the correlations among attributes. Unfortunately, CN2 has a less-than-optimal heuristic (Chan, 1988), which prevents CN2 from fully exploring its advantage in polythetic evaluation. Moreover, BCT did not excel ID3 or CN2 in the natural domains. The comparable performance of BCT was due to limited search of the space of complexes, which might result in skipping informative complexes. Another reason might be the inherent monothetic characteristics of those data.

As in ID3, BCT represents knowledge in a decision tree, which contrasts to *decision list* (Rivest, 1987) representation used in CN2. However, BCT's decision tree logically subsumes the tree representation in ID3 and is equivalent to the list representation in CN2. It is easy to see that a BCT tree can simulate a CN2 list by always making the left branch a leaf (class) and a CN2 list can mimic a BCT tree as overviewed in the knowledge representation section. In ID3's case, BCT can build an equivalent tree by only using single attribute-value nodes. ID3 trees, however, cannot simulate BCT trees.

BCT, as well as ID3, uses a *divide-and-conquer* approach, where the concept space is recursively decomposed into two smaller subspaces in a top-down fashion. In contrast, CN2 follows a *separate-and-conquer* approach (Pagallo & Haussler, 1988), where subspaces of the concept space are individually singled out in a sequential fashion. In CN2's approach, the choice of the next best partitioning complex is independent of the possibilities for partitioning the rest of the subspace.

CONCLUDING REMARKS

In sum, BCT was generally more accurate than ID3 and CN2. However, it was comparable to these two systems in some domains. To further improve BCT, we could fine tune the search techniques and heuristic measures. Currently, the author is investigating a variant of the beam search and other statistical/information techniques or combinations of them. Lastly, future explorations might involve strategically converting a BCT tree to a production system and eliminating insignificant nodes (Quinlan, 1987).

Acknowledgments

I would like to thank Andrea Danyluk, Tom Ellman, Doug Fisher, and Sal Stolfo for their invaluable comments on an earlier draft of this paper.

Erratum

In the discussion section of Chan (1989), Kullback's statistic was claimed to be more effective than Quinlan's information gain in guiding search. Mingers (1989), however, shows that the two measures are equivalent. The erroneous claim was based on some implausible sample distributions. The author hereby apologizes for any inconvenience that might have been caused.

References

Chan, P. K. (1988). A critical review of CN2: A polythetic classifier system. Technical report CS-88-09, Department of Computer Science, Vanderbilt University, Nashville, TN.

Chan, P. K. (1989). Inductive learning with BCT. *Proceedings of the Sixth International Workshop on Machine Learning* (pp. 104-108). Ithaca, NY: Morgan Kaufmann.

Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*, 261-283.

Kullback, S. (1959). *Information theory and statistics*. New York, NY: Wiley.

Michalski, R. S. (1973). Discovering classification rules using variable-valued logic system VL1. *Advanced Papers of the Third IJCAI* (pp. 162-172). Stanford, CA: Morgan Kaufmann.

Mingers, J. (1989). An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, *3*, 319-342.

Mitchell, T. M. (1982). Generalization as search. *Artificial Intelligence*, *18*, 203-226.

Pagallo, G. & Haussler, D. (1988). Feature discovery in empirical learning. Technical report UCSC-CRL-88-08, Department of Information and Computer Science, University of California, Santa Cruz, CA.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81-106.

Quinlan, J. R. (1987). Generating production rules from decision trees. *Proceedings of the Tenth IJCAI* (pp. 304-307). Milan, Italy: Morgan Kaufmann.

Rivest, R. L. (1987). Learning decision lists. *Machine Learning*, *2*, 229-246.