Van Someren and Widmer (eds.) ECML-97 poster papers, 29-38, 1997

A Feature-Relevance Heuristic for Indexing and Compressing Large Case Bases

Walter Daelemans[1], Antal van den Bosch[2], and Jakub Zavrel[1]

[1] Computational Linguistics, Tilburg University, The Netherlands
[2] Dept. of Computer Science, Universiteit Maastricht, The Netherlands

Abstract. This paper reports results with IGTree, a formalism for indexing and compressing large case bases in Instance-Based Learning (IBL) and other lazy-learning techniques. The concept of information gain (entropy minimisation) is used as a heuristic feature-relevance function for performing the compression of the case base into a tree. IGTree reduces storage requirements and the time required to compute classifications considerably for problems where current IBL approaches fail for complexity reasons. Moreover, generalisation accuracy is often similar, for the tasks studied, to that obtained with information-gain-weighted variants of lazy learning, and alternative approaches such as C4.5. Although IGTree was designed for a specific class of problems –linguistic disambiguation problems with symbolic (nominal) features, huge case bases, and a complex interaction between (sub)regularities and exceptions– we show in this paper that the approach has a wider applicability when generalising it to TRIBL, a hybrid combination of IGTree and IBL.

1 Introduction

The formalism presented here originated in research on the application of lazy-learning techniques (such as case-based learning, instance-based learning, and memory-based reasoning) to real-world problems in language technology, cf. [Dae95] and [Car93] for overviews of the approach and results. The main difference between learning linguistic problems and learning the typical "benchmark" problems employed in machine-learning research, is that for the linguistic problems huge datasets are available. For example, in the problem of part-of-speech tagging (disambiguating the syntactic category of a word given the words in its context), one case is generated for each word in the training corpus, resulting in millions of cases if the case space is to be filled sufficiently in order to cope with the sparse data problem. Because of their computational complexity, standard inductive machine-learning algorithms can often not be applied to datasets of more than a fraction of this size.

In lazy learning (e.g., the IB1 instance-based learning algorithm described in [Aha91]), similarity of a new case to stored cases is used to find the nearest neighbours of the new case. The class of the new case is predicted on the basis of the classes associated with the nearest-neighbour cases and the frequency of their occurrences. In IB1, all features are assigned the same relevance, which is undesirable for our linguistic problems. We noticed that IB1, when extended with a simple feature-weighting similarity function, outperforms IB1, and sometimes also outperforms both connectionist approaches and knowledge-based "linguistic–engineering" approaches [VD93]. The similarity function we introduced in lazy learning [DV92] consisted simply of multiplying, when comparing two feature vectors, the similarity between the values for each feature with the corresponding *information gain*, or in case of features with different numbers of values the *gain ratio*, for that feature.We call this version of lazy learning IB1-IG.

During experimentation with the linguistic problems, we also found that accuracy (generalisation performance) decreased considerably when the case base is pruned in some way (e.g., using IB2 [Aha91], or by eliminating non-typical cases). Keeping available all potentially relevant case information turns out to be essential for good accuracy on our linguistic problems, because they often exhibit a lot of sub-regularity and pockets of exceptions that have potential relevance in generalisation. Unfortunately, as the prediction function in lazy learning has to compare a test case to all stored cases, and our linguistic datasets typically contain hundreds of thousands of cases or more, processing of new cases is prohibitively slow on single-processor machines. Based on these findings with IB1-IG, we designed a variant of IB1 in which the case base is compressed into a tree-based data structure in such a way that access to relevant cases is faster, and no relevant information about the cases is lost. This simple algorithm, IGTree [VD93, DVW97], uses a feature-relevance metric such as information gain to restructure the case base into a decision tree.

In Section 2, we describe the IGTree model and its properties. Section 3 describes comparative experiments with IGTree, IB1, and IB1-IG on learning one of our linguistic tasks and some of the UCI benchmark problems. In Section 4, we discuss problems for IGTree with other benchmark datasets, introduce TRIBL, and describe comparative experiments. We discuss related research in Section 5, and present our conclusions in Section 6.

2 IGTree

In this Section, we provide both an intuitive and algorithmic description of IGTree, and provide some analyses on complexity issues. A more detailed discussion can be found in [DVW97].

IGTree compresses a case base into a decision tree by recursively partitioning the case base on the basis of the most relevant features. All nodes of the tree contain a test (based on one of the features) and a class label (representing the *most probable* (most frequently occurring) *class* of the case-base partition indexed by that node). Nodes are connected via arcs denoting the outcomes for the test (feature values), so that individual cases are stored as paths of connected nodes. A feature-relevance ordering technique (e.g., information gain) is used to determine a fixed order in which features are used as tests throughout the whole tree. Thus, the maximal depth of the tree is always equal to the number of features. A considerable compression is obtained as similar cases share partial paths, i.e., prefixes of the relevance-ordered features. A considerable speed gain is obtained over IB1, since the classification of a test case can now be found by a fast tree traversal restricted to those stored cases that share the same prefix. Tree traversal involves matching all feature values of the test case with arcs in the order of the overall feature information gain, and either retrieving a classification when a leaf is reached, or using the default classification on the last matching node if a feature-value match fails.

Instead of converting the case base to a tree in which all cases are fully represented as paths, the size of the tree is kept small by only expanding those nodes for which the underlying case base partition is still ambiguous with respect to the class. The underlying idea is that it is not necessary to fully store a case as a path when only a few feature values of the case make its classification unique.

A final compression is obtained by pruning the derived tree. All leaf-node children of a parent node that have the same class as the parent node are removed from the tree, as their class information does not contradict the default class information already present at their parent.

The recursive algorithms for tree construction (except the final pruning) and retrieval are given in Figure 1.

In Figure 2, an example application of IGTree to a simple database of 12 objects with three (nominal) features is visualised. On the basis of the dataset, we computed that the information gain values of the three features are 0.75 for 'size', 0.90 for 'shape', and 1.10 for '# holes'; consequently, '# holes' is the feature expanded at the first level of the tree, followed by 'shape' at the second level, and concluded by 'size'. The tree generated by IGTree (displayed at the bottom right of Figure 2) contains 10 nodes. For comparison, Figure 2 also displays the tree generated by C4.5 on the same data (bottom left). This tree contains 13 nodes.

The asymptotic complexity of IGTree (i.e, in the worst case) is extremely favourable. Complexity of searching a query pattern in the tree is proportional to $F * log(V)$, where F is the number of features (equal to the maximal depth of the tree), and V is the average number of values per feature (i.e., the average branching factor in the tree). In IB1, search complexity is $O(N * F)$ (with N the number of stored cases). Retrieval by search in the tree is independent from the number of training cases, and therefore especially useful for large case bases. Storage requirements are proportional to N (compare $O(N * F)$ for IB1). Finally, the cost of building the tree on the basis of a set of cases is proportional to $N * log(V) * F$ in the worst case (compare $O(N)$ for training in IB1).

In practice, for a typical linguistic dataset (part-of-speech tagging) IGTree retrieval is over 200 times faster than IB1 retrieval, uses over 95% less memory, and building the tree takes about twice the time of simply storing the cases.

3 Experiments

In this Section we describe empirical results achieved with IGTree on a morphosyntactic disambiguation problem for English (one of our linguistic datasets),

```
Procedure BUILD-IG-TREE:
Input:
  – A training set $T$ of cases with their classes (start value: a full case base),
  – an information-gain-ordered list of features (tests) $F_i...F_n$ (start value: $F_1...F_n$).
Output: A (sub)tree.
  1. If $T$ is unambiguous (all cases in $T$ have the same class $c$), create a leaf node with class
     label $c$.
  2. Else if $i = (n + 1)$, create a leaf node with as label the class occurring most frequently in
     $T$.
  3. Otherwise, until $i = n$ (the number of features)
     – Select the first feature (test) $F_i$ in $F_i...F_n$, and construct a new node $N$ for feature
       $F_i$, and as default class $c$ (the class occurring most frequently in $T$).
     – Partition $T$ into subsets $T_1...T_m$ according to the values $v_1...v_m$ which occur for $F_i$
       in $T$ (cases with the same value for this feature in the same subset).
     – For each $j \epsilon \{1, ..., m\}$: BUILD-IG-TREE $(T_j, F_{i+1}...F_n)$,
       connect the root of this subtree to $N$ and label the arc with $v_j$.

Procedure SEARCH-IG-TREE:
Input:
  – The root node $N$ of a subtree (start value: top node of a complete igtree),
  – an unlabelled case $I$ with information-gain-ordered feature values $f_i...f_n$ (start value:
    $f_1...f_n$).
Output: A class label.
  1. If $N$ is a leaf node, output default class $c$ associated with this node.
  2. Otherwise, if test $F_i$ of the current node does not originate an arc labelled with $f_i$, output
     default class $c$ associated with $N$.
  3. Otherwise,
     – new node $M$ is the end node of the arc originating from $N$ with as label $f_i$.
     – SEARCH-IG-TREE $(M, f_{i+1}...f_n)$
```

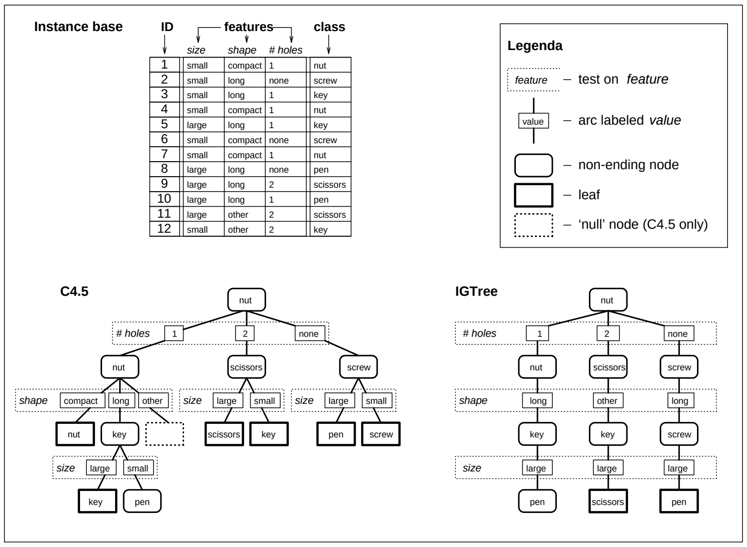
Fig. 1. Procedures for building IGTrees ('BUILD-IG-TREE') and searching IGTrees ('SEARCH-IG-TREE').

and the *mushroom* and *soybean* UCI benchmark datasets. We compare the performance of IGTree in terms of generalisation accuracy and storage to IB1 (implementing 1-nearest-neighbour), and IB1-IG. When available, results for C4.5, and EODG [KL95] are provided as well. EODG is an implementation of Oblivious Decision Graphs, which is an extension to Oblivious Decision Trees (see Section 5 for a discussion on the relation between IGTree and Oblivious Decision Trees).

In all experiments, we used a 10-fold CV setup: i.e., we trained and tested each algorithm on ten different partitions (90% training material, 10% testing material) of the full dataset. All performance results reported below are averaged over these experiments. Continuous features are discretised.

3.1 Morphosyntactic Disambiguation

The problem of morphosyntactic disambiguation (Part-of-speech tagging) is the following: given a text, provide for each word its contextually disambiguated part of speech (morphosyntactic category), i.e., transform a string of words into a string of tags. For example, the sentence `'the old man the boats.'`



	features			class
ID	*size*	*shape*	*# holes*	
1	small	compact	1	nut
2	small	long	none	screw
3	small	long	1	key
4	small	compact	1	nut
5	large	long	1	key
6	small	compact	none	screw
7	small	compact	1	nut
8	large	long	none	pen
9	large	long	2	scissors
10	large	long	1	pen
11	large	other	2	scissors
12	small	other	2	key

Fig. 2. Graphical display of the conversion of an instance base (top, left) to a C4.5 decision tree (bottom left) and a decision tree generated by IGTree (bottom right). The legend provides the meaning of the different symbols.

should be mapped to '`Det Noun Verb Det Noun Punc`'. The target category inventory (tag set) may range from simple (order 10) to extremely complex (order 1000). Regardless of tag-set size, tagging is a complex task because of the massive ambiguity in natural language text. The correct category of a word depends on both its lexical probability $Pr(cat|word)$, and its contextual probability $Pr(cat|context)$. Tagging is a typical instance of the type of disambiguation tasks found in Natural Language Processing.

The architecture of our Instance-Based Learning tagger (see [DZBG96] for a full description) takes the form of a *tagger generator*: given a corpus tagged with the desired tag set, a tagger is generated which maps the words of new text to tags in this tag set according to the same systematicity. The construction of a tagger for a specific corpus is achieved in the following way. Given an annotated corpus, three datastructures are automatically extracted: a *lexicon* (containing words with their associated tags as occurring in the training corpus), a case base for *known words* (words occurring in the lexicon with their contexts), and a similar case base for *unknown words*. Case bases are compressed using IGTree for efficiency. During tagging, each word in the text to be tagged is looked up in the lexicon. If it is found, its lexical representation is retrieved and its context is determined, and the resulting pattern is disambiguated using extrapolation from nearest neighbours in the known-words case base.

For unknown words, a tag can be guessed only on the basis of the *form* and the *context* of the word. We provide word form information to the tagger by encoding the first letter and the three last letters of the word as separate features in the case representation. Context information is added to the case representation in a similar way as with known words. In an evaluation reported in [DZBG96] it is shown that the IGTree tagger, trained on 2 million cases, has a performance on new text that is competitive with alternative hand-crafted and statistical approaches (96.7% on known words, 90.6% on unknown words, 96.4% overall generalisation accuracy). Both training and testing speed are excellent (new text tagging is possible with a speed of up to 1200 words per second, user time).

For a comparison of IGTree tagging with alternative IBL approaches, we tested generalisation accuracy and storage requirements, of IB1, IB1-IG and IGTree in a 10-fold cross-validation experiment on a dataset of 100,000 words. Table 1 summarises the results. The use of IGTree as a heuristic approximation to IB1-IG did not result in a serious loss of generalisation accuracy while at the same time accounting for a spectacular decrease in memory and time consumption: retrieval is 100 to 200 times faster than IB1-IG retrieval, and uses over 97% less memory.

Table 1. Comparison of three instance-based learning techniques.

Algorithm	Accuracy	Time	Memory (Kb)
IB1	93.9 $\pm$0.2	0:43:34	900
IB1-IG	97.0 $\pm$0.2	0:49:45	900
IGTree	96.6 $\pm$0.2	0:00:29	21

3.2 The mushroom and soybean datasets

We performed a 10-fold CV experiment on the *mushroom* and *soybean* datasets, taken from the UCI benchmark dataset repository. The mushroom dataset contains 5,644 cases. All cases have 22 nominal features, and each case (representing a mushroom instance) is labelled 'edible' or 'poisonous', i.e., each case maps to one of two classes. The soybean (large) dataset contains 631 cases with 35 attributes, all of which were discretised. Each case (representing the symptoms of a plant) is labelled with one of 19 diagnoses.

For both IGTree and IB1-IG, the gain ratio criterion [Qui93] rather than the information gain criterion was employed, as the dataset has differing numbers of feature values. Also displayed in Table 2 are the results reported by Kohavi and Li [KL95], who performed 10-fold CV experiments on the same datasets, with C4.5, C4.5rules, and EODG [KL95]. Table 2 displays the average generalisation performance for these two datasets on test cases of IGTree, IB1, and IB1-IG, and lists the average sizes of the decision trees and graphs generated by IGTree as found in our experiments, and by C4.5 and EODG as reported in [KL95].

The results of Table 2 indicate that the generalisation accuracy of IGTree is similar and sometimes better than that of the algorithms tested by Kohavi and

Table 2. Average generalisation performance (with standard deviation, after the ± symbol), of IGTree, IB1, IB1-IG, C4.5, C4.5rules, and EODG, applied to the *mushroom* and *soybean* datasets.

Algorithm	General. accuracy mushroom (%)	Size (# nodes)	General. accuracy soybean (%)	Size (# nodes)
IGTree	100.00 ±0.00	20.0	91.61 ±2.84	207.1
IB1	100.00 ±0.00	–	91.30 ±3.10	–
IB1-IG	100.00 ±0.00	–	91.30 ±2.49	–
Kohavi & Li (1995):				
EODG	100.00 ±0.00	31.6	81.13 ±1.87	80.9
C4.5	100.00 ±0.00	30.5	92.54 ±1.43	66.5
C4.5rules	100.00 ±0.00	–	91.28 ±1.00	–

Li [KL95], IB1, and IB1-IG. An interesting fact displayed in Table 2 is that for the *mushroom* dataset, IGTree is able to generate a tree that is even smaller than those generated with C4.5 and EODG, compressing the amount of memory needed versus that of IB1 with a factor of 99.9%.

4 The TRIBL hybrid

Preliminary results with applications of IGTree on other UCI datasets suggested that IGTree is not applicable to problems where the relevance of the predictive features cannot be ordered in a straightforward way. In those cases, IB1-IG or even IB1 perform significantly better than IGTree. UCI datasets for which this is the case include *Tic-tac-toe, Vehicle, Chess, Glass*, and *Letter*. For all these datasets, IGTree performs significantly worse than IB1-IG and IB1.

In this Section, we discuss TRIBL, a hybrid generalisation of IGTree and IBL. TRIBL searches for an optimal trade-off between (i) minimisation of storage requirements by building an IGTree (and consequently also optimisation of search speed) and (ii) maximal generalisation accuracy. To achieve this, a parameter is set determining the switch from IGTree to IBL in learning as well as in classification. The parameter used currently is based on *average feature information gain*; when the information gain of a feature exceeds the sum of the average information gain of all features + one standard deviation of the average, then the feature is used for constructing an IGTree according to the tree building algorithm in Figure 1, including the computation of defaults on nodes. When the information gain of a feature is below this threshold, and the node is still ambiguous, tree construction halts and the leaf nodes at that point represent case bases containing subsets of the original training set. During search, the normal IGTree search algorithm is used, until the case-base nodes are reached, in which case IBL is used on this sub-case-base. In the next Section, we present results with TRIBL on the UCI datasets on which IGTree performs badly, and compare these results to alternative IBL methods.

Table 3. Average generalisation performance (with standard deviation, after the $\pm$ symbol), of IGTree, IB1, IB1-IG, TRIBL, C4.5, C4.5rules, and EODG, with sizes of the decision trees and graphs, applied to the *Tic-tac-toe, Vehicle, Chess, Glass*, and *Letter* datasets. The switch point for TRIBL is given below the accuracy.

Algorithm	Generalisation accuracy (%)				
	Tictactoe	*Vehicle*	*Chess*	*Glass*	*Letter*
IGTree	85.59 ±3.32	66.52 ±7.74	96.12±0.87	72.40± 10.82	74.49±1.28
IB1	98.75 ±0.66	69.96 ±7.27	96.62±0.79	78.03±11.57	89.58±0.81
IB1-IG	89.56 ±2.21	70.45 ±6.85	96.65±0.71	75.67±12.58	90.93±1.24
TRIBL	98.75 ±0.66	70.57 ±6.75	97.81±0.72	76.13±12.36	86.99±1.37
Switch point	1	2	6	1	2
Kohavi & Li (1995):					
EODG	89.4 ±0.96	54.6 ±2.72	98.5 ±0.25	–	–
C4.5	85.6 ±1.08	69.8 ±1.77	99.5 ±0.13	–	–
C4.5rules	98.9 ±0.57	71.9 ±1.56	99.5 ±0.12	–	–

4.1 Results on the benchmarks

Table 3 displays the average generalisation performance for five datasets on which IGTree performs significantly worse than unweighted IB1. Significance testing in paired t-tests ($p < 0.05$) gives the following results: for the *Tic-tac-toe* dataset all differences are significant, except for the difference between IB1 and TRIBL. The differences for the *Vehicle* and *Glass* datasets are not significant, except for IGTree performing worse than the other IBL-variants. The *Chess* dataset demonstrates a significant advantage of TRIBL over all other algorithms. For the *Letter* dataset, all differences are significant, showing an advantage of IB1 and IB1-IG over the other algorithms.

In sum, TRIBL performs well on these five datasets, attaining roughly the same performance as the better of algorithms, except with the *Letter* dataset. These results are also in the same range as the results of C4.5, C4.5rules, and EODG reported in [KL95] (except for the *Chess* database).

Due to the fact that a partial IGTree is still built in TRIBL, the speed advantage of IGTree largely remains in TRIBL. Although the proportion of features that remain in the IBL part of the algorithm is quite large in some cases, matching the few indexing features already corresponds to a huge decrease in the size of the remaining case base. Nonetheless, TRIBL represents a trade-off between speed and generalisation accuracy, controlled by the switchpoint parameter. For very large case bases, the use of IB1-IG is often computationally out of reach, and the performance loss can be kept small. Notice that the computational cost of inducing C4.5 trees and rules is very high.

5 Related research

In this Section, we discuss two related algorithms for decision tree induction, viz. Top-Down Induction of Decision Trees (TDIDT), and Oblivious Decision Trees or Graphs.

A fundamental difference with TDIDT concerns the purpose of IGTrees. The goal of TDIDT, as in the state-of-the-art program C4.5 [Qui93], is to *abstract* from the training examples. In contrast, we use decision trees for *lossless* compression of the training examples. Pruning of the resulting tree in order to derive understandable decision trees or rule sets is therefore not an issue in our approach. By lossless we mean that the classifications of the training cases can be completely reconstructed, not that all feature-value information in the original training set can be reconstructed. Generalisation is achieved by the defaults at each node, not by pruning.

A simplification of IGTree as opposed to TDIDT approaches such as C4.5, is that IGTree generates oblivious decision trees, i.e., it computes information gain only once to determine a *fixed* feature ordering. C4.5, in contrast, recomputes information gain (or similar feature selection functions) at each arc of the tree to guide selection of the next test. This makes IGTree induction considerably faster than C4.5. The price paid for this advantage, however, is that the influence of feature interaction on the relevance order is ignored.

The IGTree approach differs in one essential aspect from other oblivious-decision-tree [LS94] and oblivious-decision-graph [KL95] approaches: in trees generated by IGTree, leaves are not necessarily stored at the same level. During tree building, expansion of the tree is stopped when all cases in the subset indexed by a node are of the same class. Similarly, IGTree classifies a new case by investigating a variable and often limited number of features, rather than a fixed number of (relevant) features, as in [KL95].

6 Conclusions

We have shown that IGTree, a case-base compression and indexing formalism based on oblivious decision trees and a feature-relevance heuristic, is extremely well suited for a class of problems which can be characterised by the following properties:

- A large case base is available (on the order of 100-1000 K cases).
- Lazy-learning approaches keeping full memory are at an advantage. This is often the case with complex task domains: regularities and subregularities are contradicted by *pockets of exceptions* which account for part of the generalisation accuracy. Removing the exceptions decreases accuracy significantly.
- A feature-relevance ordering technique is available which assigns sufficiently differing relevance to individual features to allow a fixed ordering.

These properties apply to a large class of real-world problems, including almost all disambiguation tasks in language technology [Dae95, Car93]. For this type of task, IGTree attains a generalisation accuracy similar to alternative lazy-learning techniques and other inductive machine-learning techniques, with modest memory space and processing time requirements. Retrieval is especially fast because its complexity is independent from the number of cases.

For those tasks where the feature relevance ordering heuristic results in an IGTree performance inferior to IBL, we have extended IGTree into the hybrid algorithm TRIBL that allows us to experiment with the trade-off between maximal generalisation accuracy and favourable storage and speed results.

Acknowledgements

We thank David Aha and Ton Weijters for their comments.

References

[Aha91] Aha, D. W., Kibler, D., & Albert, M. (1991). Instance-based learning algorithms. *Machine Learning*, 7, 37–66.

[Car93] Cardie, C. (1996). Embedded Machine Learning Systems for Natural Language Processing: A General Framework. In: Wermter, S., Riloff, E., and Scheler, G. (Eds.), *Symbolic, connectionist, and statistical approaches to learning for natural language processing*, Springer LNAI Series, 1996.

[Dae95] Daelemans, W. (1995). Memory-based lexical acquisition and processing. In Steffens, P. (Ed.), *Machine Translation and the Lexicon*, Lecture Notes in Artificial Intelligence 898. Berlin: Springer.

[DV92] Daelemans, W. and Van den Bosch, A. (1992). Generalisation performance of backpropagation learning on a syllabification task. In M. Drossaers and A. Nijholt (Eds.), *TWLT3: Connectionism and Natural Language Processing.* Enschede: Twente University.

[DVW97] Daelemans, W., Van den Bosch, A., and Weijters, A. (1997). IGTree: Using trees for classification in lazy learning algorithms. To appear in *Artificial Intelligence Review*, special issue on lazy learning.

[DZBG96] Daelemans, W., J. Zavrel, P. Berck, S. Gillis. (1996) MBT: A Memory-Based Part of Speech Tagger-Generator. In: E. Ejerhed and I. Dagan (eds.) *Proceedings of the Fourth Workshop on Very Large Corpora*, Copenhagen, Denmark, 14–27.

[KL95] Kohavi, R. and Li, C-H. (1995). Oblivious decision trees, graphs, and top-down pruning. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1071–1077. Montreal: Morgan Kaufmann.

[LS94] Langley, P. and Sage, S. (1994). Oblivious decision trees and abstract cases. In D. W. Aha (Ed.), *Case-Based Reasoning: Papers from the 1994 Workshop* (Technical Report WS-94-01). Menlo Park, CA: AAAI Press.

[Mur95] Murphy, P. (1995). UCI repository of machine learning databases – a machine-readable repository. Maintained at the Department of Information and Computer Science, University of California, Irvine. Anonymous ftp from `ics.uci.edu` in the directory `pub/machine-learning/databases`.

[Qui93] Quinlan, J. (1993). *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

[VD93] Van den Bosch, A. and Daelemans, W. (1993). Data-oriented methods for grapheme-to-phoneme conversion. In *Proceedings of the 6th Conference of the EACL*, 45–53. Utrecht: OTS.