Competitively Evolving Decision Trees Against Fixed Training Cases for Natural Language Processing

to appear in
Advances in Genetic Programming
editted by Kim Kinnear

Eric V. Siegel

Competitive fitness functions can generate performance superior to absolute fitness functions [Angeline and Pollack 1993], [Hillis 1992]. This chapter describes a method by which competition can be implemented when training over a fixed (static) set of examples. Since new training cases cannot be generated by mutation or crossover, the probabilistic frequencies by which individual training cases are selected competitively adapt. We evolve decision trees for the problem of word sense disambiguation. The decision trees contain embedded bit strings; bit string crossover is intermingled with subtree-swapping. To approach the problem of overlearning, we have implemented a fitness penalty function specialized for decision trees which is dependent on the partition of the set of training cases implied by a decision tree.

19.1 Introduction

Competitive fitness functions can generate performance superior to absolute fitness functions [Angeline and Pollack 1993], [Hillis 1992]. A competitive fitness function is computed through some type of interaction between co-adapting individuals. For example, [Angeline and Pollack 1993] evolves Tic Tac Toe players whose fitness measurements are computed by having population members participate in a Tic Tac Toe tournament. Hillis [1992] evolves sorting networks in competition with a separate, evolving population of lists of numbers to be sorted. A sorting network's fitness depends on how well it sorts lists of numbers from the competing population, and a list of number's fitness is dependent on how poorly it is sorted by sorting networks. Competitive fitness functions guide the adaptive process since weaknesses of adapting individuals are discovered and therefore accented by competing adapting individuals.

Hillis' sorting networks are evaluated against a dynamic population of training cases whose adaptation involves crossover and mutation. However, many induction tasks involve a fixed, static "population" of training examples which cannot participate in these creative evolutionary operations. Such situations arise when the training data have been empirically collected, as in symbolic regression. In this chapter, we present a method by which the training set can competitively adapt without the generation of new training examples.

Decision trees [Quinlan 1986] are appropriate for many pattern classification problems. Koza [1991] has demonstrated that the genetic programming paradigm is capable of inducing decision tree structures. In this chapter, we evolve decision trees for a real world problem with noisy, empirical data.

In [Tackett 1993], genetic programming is used for a two class classification problem. It is compared to a binary tree classifier, which statistically induces trees similar to the decision trees presented in this chapter.

Sections 19.2 and 19.3 describe the problem domain and the set of training examples. Sections 19.4 and 19.5 explain how decision trees work and how crossover is implemented for the decision tree representation. Section 19.6 describes how fixed training data can participate in competitive adaptation. Section 19.7 describes a method to avert overlearning when inducing decision trees over a limited training set. Finally, section 19.8 draws conclusions.

19.2 The Domain: Word Sense Disambiguation

In natural language, many words have multiple *senses* (meanings). For example, *anyway* can mean "in any case", as in, "I did it *anyway*." (This is considered the *sentential* meaning of *anyway*.) It can also mean, "Let's return to a previous topic", as in "*Anyway*, what were you saying before?" (This is considered the *discourse* meaning of *anyway*.) The sense of an ambiguous word such as *anyway* is dependent on the context in which it is used. A major thrust of the natural language understanding field is deriving mechanisms to resolve such ambiguities, a process called *disambiguation*.

The approach to word sense disambiguation taken here is to evolve decision trees which attempt to establish word sense by looking only at *immediate* context, that is, the *tokens*, (words and punctuations marks) residing within a small distance of the word to be disambiguated. Specifically, a decision tree can examine the tokens residing immediately to the left of the word, and those up to four positions to the right (positions **-1** through **4**).

The class of words being disambiguated are *discourse cue words* (e.g. *anyway*). Table 19.2 contains example discourse cue words. A discourse cue word is used by a speaker to convey intentions with respect to the "flow" of a discourse. Cue words often indicate how a sentence or clause relates to the current topic of conversation, e.g. digression, conclusion, etc. Each discourse cue word also has at least one alternative sense as a verb, adverb or connective, its *sentential* sense. Therefore, any instance of such a word must be disambiguated as to whether it is being used in a *discourse* sense or a *sentential* sense.

Hirschberg and Litman [1993] explore several methods for cue word sense disambiguation, including the examination of intonational features. They also measure the ability to perform this task by looking only at punctuation marks immediately before and after the

Table 19.1
Tableau for the competitive evolution of decision trees for word sense disambiguation

Objective:	Evolve a decision tree which classifies occurrences of *discourse cue words* as to their usage.
Terminal set:	The two classes of this classification problem, specifically, *discourse* and *sentential*.
Function set:	Each internal node is like a `switch` statement in C. A series of comparisons is made, and one downward arc is selected. See section 19.4 for detail.
Fitness cases:	513 examples of *discourse cue words*, their immediate context as used in spoken English, and their meaning as used in that context.
Raw fitness:	A decision tree is evaluated over a competitively selected distribution of training cases (513 cases), and raw fitness is the number of training cases correctly classified. See section 19.6 for details on competition against fixed training cases. See section 19.7 for the description of a fitness penalty which averts overlearning.
Standardized fitness:	513 minus raw fitness.
Parameters:	Number of generations = 500, population size = 900.
Termination predicate:	Reach final generation of run.
Identification of best:	Every 20 generation, the population of decision trees is tested for absolute fitness (as evaluated uniformly across the 513 training cases).

Table 19.2
Example discourse cue words – those which occur most frequently in this study. A total of 34 cue words occur. The number of times each discourse cue word occurs as a training case is listed.

Cue word	**Instances**	**Cue word**	**Instances**	**Cue word**	**Instances**	**Cue word**	**Intances**
and	348	*like*	71	*say*	36	*actually*	29
now	75	*but*	56	*well*	35	*see*	29
so	74	*or*	55	*look*	35	*first*	25

cue word, suggesting the strategy embodied by the decision tree in Figure 19.1.[1] This decision tree correctly classifies 79.16% of the training cases used in these experiments. If the word is the first in a sentence, i.e. following a period, it is classified as *discourse*.

The next section introduces the set of training examples we have for the problem of word sense disambiguation, and formalizes word sense disambiguation as a classification problem.

[1]This decision tree is extrapolated from Table 11 in [Hirschberg and Litman 1993]. Their study used the same transcript as this one, and primarily used the same training cases.

Table 19.3
Example training cases. Each training case has a value for each of 6 attributes, and a class. Attribute **0** is the cue word to be disambiguated.

-1	**0**	**1**	**2**	**3**	**4**	**Class:**
work	*and*	*we*	*are*	*really*	*pleased*	`discourse`
.	*But*	*we*	*stop*	*there*	*because*	`discourse`
.	*Now*	*that*	*doesn't*	*mean*	*we*	`discourse`
very	*well*	*founded*	*principle*	*principled*	*in*	`sentential`
to	*look*	*more*	*like*	*sentences*	.	`sentential`
,	*and*	*that's*	*on*	*the*	*second*	`sentential`
description	*ok*	.	*Is*	*a*	*surgeon*	`discourse`

19.3 The Training Cases

For these experiments, we have access to the transcript of spoken English used by Hirschberg and Litman [1993]. In this transcript, each discourse cue word has been manually marked by a linguist as to whether its sense is *discourse* or *sentential*. The training examples are therefore empirical data – measurements of human perception. The transcript provides 1,027 examples.

Table 19.3 contains sample training data. Each training example has 6 *attributes*: positions -1 through 4. This includes position zero, the discourse cue word to be disambiguated. For any given training example, each attribute has a corresponding *value*, that is, a token. Each training example also has a *class*, that is, the word sense. Note that we have formalized word sense disambiguation as a two class classification problem.

Decision trees operate on one training example at a time, attempting to derive the correct classification. We evolutionarily induce decision trees which correctly classify a high percentage of the training cases. In order to ascertain the generalization performance of these trees, induction takes place over one half of the training cases, and evolved trees are tested over the remaining cases, the *test cases*. The division into training and test sets is made randomly upon each run.

The next section describes the decision tree mechanism, and how it is used for word sense disambiguation.

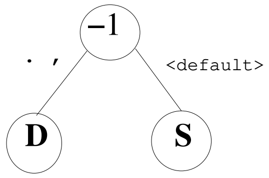

Figure 19.1
This small, manually created decision tree correctly classifies 79.16% of the training examples. At leaves, "D" stands for *discourse* and "S" for *sentential*.

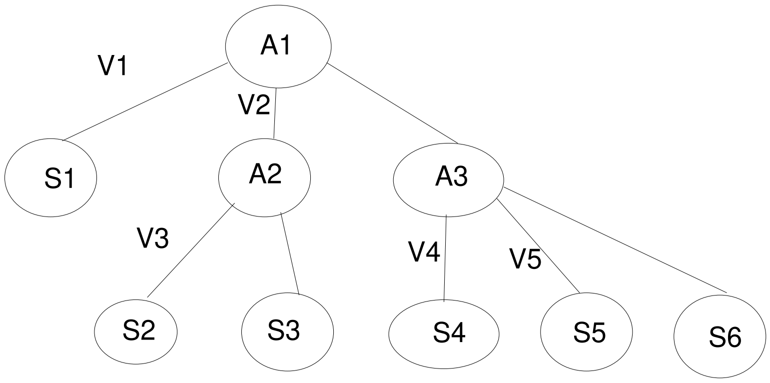

Figure 19.2
Formal representation of a decision tree. Internal nodes are labeled with attribute sets, arcs are labeled with value sets, and leaves are labeled with a class. Rightmost arcs are "default paths".

19.4 How Decision Trees Work

Figure 19.2 shows the formal representation of a decision tree, and Figures 19.3 and 19.4 show example decision trees generated by evolution.[2] Each internal node of a tree is labeled with a set of attributes (token positions), each arc is labeled with a set of values (tokens), and each leaf is labeled with a class name (word sense).

The internal nodes are treated like a `switch` statement in C. A series of comparisons is made, and one downward arc is selected. For a given training example, the tree is traversed deterministically from root to leaf, thus classifying the example, by the following recursive process:

[2]These trees have been automatically edited to remove most redundant and useless data for the purpose of inspection. The editing process preserves semantics and is *not* part of the evolutionary process.

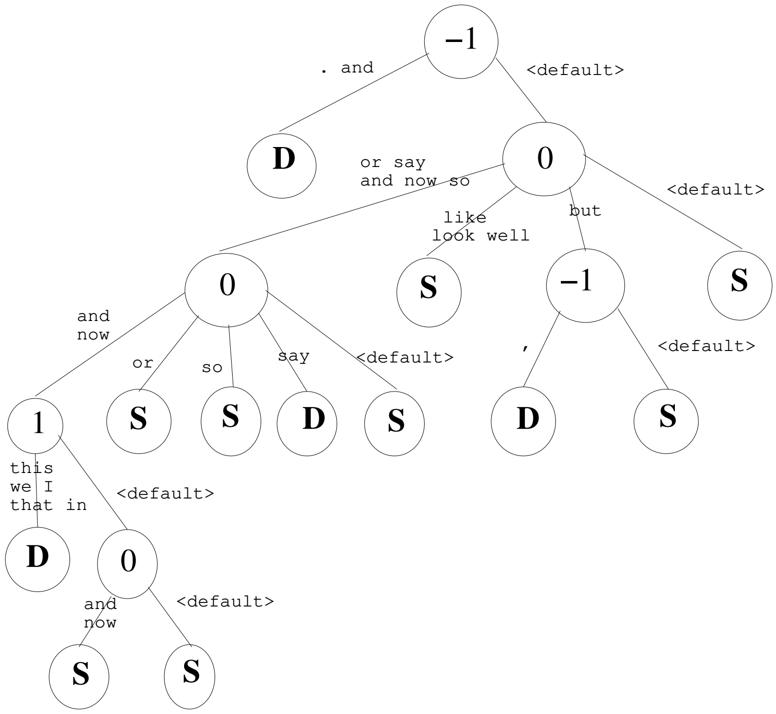

Figure 19.3
Example decision tree, induced by evolution. At leaves, "D" stands for *discourse* and "S" for *sentential*. This tree scored 81.09% over the training set and 83.27% over the test set. The original (unedited) tree has 37 nodes.

At the current internal node, the set of values from the training example which correspond to the node's attributes is identified, and the first arc with an intersecting value set, going from left to right, is selected.

The rightmost arc under each internal node is a "default" arc which has no explicit value set. This arc is traversed if none of its sister arcs has an intersecting value set.

For example, to classify the first training example from Table 19.3 with the decision tree in Figure 19.3, the tree traversal starts at the root node. The right arc is traversed, since position -1 has neither a period nor *and*. Then the leftmost arc is traversed, since position **0** has value *and*. Then two more leftmost arcs are traversed, leading to a leaf which classifies the training example as *discourse*, the correct classification.

The superset of values which can be members of a decision tree's value sets is the compilation of all values which occur in the training examples. A value frequency threshold,

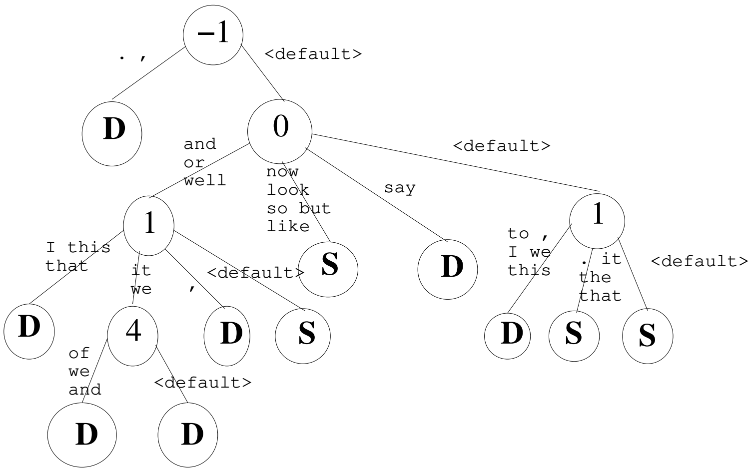

Figure 19.4
Example decision tree, induced by evolution. At leaves, "D" stands for *discourse* and "S" for *sentential*. This tree scored 83.24% over the training set and 81.52% over the test set. The original (unedited) tree has 42 nodes.

15, has been selected manually; only values occurring frequently enough are considered for explicit inclusion in decision trees. The resulting superset of values is of size approximately 26.[3] Since some values cannot appear explicitly in value sets, the default arcs are necessary.

If a node contains attribute set $\{\mathbf{0}\}$, its arcs may only contain discourse cue words. Therefore, a separate superset of values is used for the downward arcs leading from a node with attribute set $\{\mathbf{0}\}$ – The set of all discourse cue words. A frequency threshold of 4 has been selected for the superset of discourse cue words, resulting in a superset of size approximately 20. The attribute sets are therefore restricted to either being $\{\mathbf{0}\}$ or a subset of $\{\mathbf{-1, 1, 2, 3, 4}\}$.

Note that although the attribute sets of the decision trees in Figures 19.3 and 19.4 are all one-member sets, the representation allows for more than one attribute to appear at each internal node. Also note that a decision tree does not necessarily have to look at the word it is disambiguating; there are some generalizations that hold for all discourse cue words.

[3]Since the partition of train/test sets is randomly selected at the beginning of a run, the frequency count of the tokens varies, so the number of tokens with frequency above the frequency threshold varies.

19.5 Crossover Operations on Decision Trees

The crossover mechanism is designed to allow for any decision tree architecture to emerge. There are two representational issues which simple subtree swapping does not address. First, there are a variable number of daughters per node. Second, there are attribute and value sets at the internal nodes and arcs, respectively.

The attribute sets at internal nodes and value sets on tree arcs embedded in decision trees are represented as bit strings. In order to represent a set as a bit string, each member of the superset is assigned a location on the bit string. A bit string contains member x of the superset if and only if x's bit location has a 1. Therefore, attribute set bit strings are of length 6, and value set bit strings are of length approximately 20 or 26.

One-point bit string crossover is intermingled with the subtree-swapping crossover of genetic programming.[4] Once two trees have been selected for crossover, a random node from each is selected. If neither node is a leaf, and their attribute sets are *compatible*, i.e. are both $\{\mathbf{0}\}$ or neither contain $\mathbf{0}$, then with 66% probability bit string crossover will take place. In all other cases, the subtrees which have the chosen nodes as roots are swapped.

When bit string crossover is selected, crossover takes place between the nodes' attribute sets. Then, a random downward arc is selected from each node, *arc1* and *arc2*. The value sets corresponding to these arcs are crossed over. Then, the set of sister arcs to the right of *arc1*, as well as the subtrees they lead to, are exchanged with the set of arcs to the right of *arc2*, as well as the subtrees they lead to. Note that this operation alters the number of daughters per node.

19.6 How Fixed Training Data Participate in Competitive Adaptation

In order to measure the fitness of a decision tree, a subset of training cases is selected, and raw fitness is the number of these training cases correctly classified by the decision tree. The canonical method for induction over a training set is to select training cases with uniform distribution. This section describes a method by which training cases are selected competitively.

Fixed training cases cannot participate in creative evolutionary operations such as crossover and mutation. Therefore, it does not make sense to use fitness-based selection for selecting reproduction participants. Instead, fitness-based selection of individuals is used to select training cases when computing the fitness of a decision tree.

[4]David Andre's chapter in this book presents work in which subtree-swapping is intermingled with bitmap crossover.

Table 19.4
Performance improvement generated by a competitive fitness function in terms of performance over the training set. The last row shows the average performance of the individual scoring highest over the training cases during 500 generations of evolution. For the other rows, the best individual of the given generation is selected. Runs without competition compute decision tree fitness across 513 random training cases.

	Without competition			With competition		
Generation	Number of runs	Average score over training cases	Standard deviation	Number of runs	Average score over training cases	Standard deviation
100	58	80.85%	1.24	46	81.96%	1.35
200	52	81.22%	1.34	44	82.81%	1.35
300	44	81.49%	1.53	44	83.34%	1.31
400	42	81.65%	1.60	43	83.83%	1.26
500	42	81.79%	1.59	42	84.07%	1.29
Overall best	42	81.95%	1.55	42	84.09%	1.31

In a competitive environment, weaknesses are sought out by competitors. In our implementation of competition, the training cases which tend to be incorrectly classified by decision trees become more fit, and therefore selected more frequently during fitness measurements.

Competition is implemented as follows:

- Each training case has a fitness measure which is initialized to zero before the first generation of decision trees are evaluated. This fitness measure continuously adjusts. It is never again initialized.

- Each time a decision tree is tested on a training case, the training case's fitness is incremented if the tree makes the incorrect prediction, and is decremented if the tree makes the correct prediction.

- When calculating the fitness of a decision tree, 2-member tournament selection[5] over the set of training cases is repeatedly used to select 513 (non-unique) cases, so the decision tree's raw score is between 0 and 513. Note that the same training case may be used more than one time during a fitness measure. Also note that the fitnesses of training cases change *during* the fitness calculation of one decision tree.

Competitive fitness measurements are *relative*, that is, they are computed across a non-uniform distribution of training cases, and measure fitness with respect to the current fitnesses of the training cases. Therefore, in order to ascertain how much is being learned on

[5] *Tournament selection* is accomplished by selecting two or more individuals at random, and keeping only the one with highest fitness.

Table 19.5
Performance improvement generated by a competitive fitness function in terms of score over the test set. See caption of Figure 19.4 for more information.

	Without competition			With competition		
Generation	Number of runs	Average score over test cases	Standard deviation	Number of runs	Average score over test cases	Standard deviation
100	58	78.51%	1.33	46	78.49%	1.18
200	52	78.59%	1.34	44	78.80%	1.62
300	44	78.72%	1.35	44	79.02%	1.49
400	42	78.54%	1.45	43	79.11%	1.68
500	42	78.64%	1.52	42	78.92%	1.71
Overall best	42	78.45%	1.44	42	79.12%	1.72

an *absolute* scale, it is necessary to periodically compute the absolute fitness measurements (that is, uniformly across the entire training set) of the population of decision trees. The measurements of absolute fitness are used only to keep track of the best decision tree created so far; it is *never* used by the evolutionary process. Absolute fitness is computed every 20 generations.

Since 2-member tournament selection is used to select training cases, the distribution of selected cases is not as skewed as it could be for fitness proportional selection, and decision trees are given less of an opportunity to "forget" what they've already learned. Further experiments would be necessary to determine the effect of other selection procedures. It is possible that cyclic behavior could emerge, during which absolute fitness stops increasing.

Table 19.4 shows the improvements gained from a competitive fitness function in terms of the score attained over the training cases, and Table 19.5 shows the scores over the test cases for the same batch of runs. Section 19.8.2 discusses these results.

The decision trees in Figures 19.3 and 19.4 were evolved with competition.

19.7 Averting Overlearning with Decision Trees: Fitness Penalty

When evolving decision trees, there is an intrinsic tendency towards generalized learning. This is because subtrees with smaller depth have the survival advantage that they have a greater probability of remaining intact, and since shorter subtrees have fewer choice points they have less of an opportunity to over-tune to the training data than subtrees with greater depth. Also, the frequency threshold imposed on members of the value sets will tend to avert overlearning. In spite of these factors, average performance of an evolved decision tree over the test cases does not out-perform the decision tree in Figure 19.1. We therefore have implemented a fitness penalty to avert overlearning.

A decision tree can be viewed as the compilation of many *rules*. Any traversal of the tree from root to leaf in which one attribute and one value is selected at each choice point (i.e. node) is a rule of the form:

`if` (*attribute1* = *value1*) `and` (*attribute2* = *value2*) `and` (*attribute3* = *value3*) `...`
`then` *classification* = *class1*

When a decision tree is used to classify a set of training cases, each training case will be classified by one and only one of these rules. Therefore, the rules indicate a partition of the set; each rule corresponds to one partition. When training over a small set of examples, the same examples must be used repeatedly for fitness measure. Therefore, it is possible for decision tree fitness to improve by discovering many rules with small partitions. However, the smaller a rule's partition, the less likely it is that the rule embodies a valid generalization. Therefore, we have implemented the following fitness penalty:

Rules with a corresponding partition with size less than a preselected threshold are considered "illegal", and their contribution to raw fitness is subtracted.[6]

With this fitness penalty in effect, a rule must apply to some minimal number of training cases in order to add to the fitness of the decision tree it is a part of. Therefore, the penalty adds pressure for decision trees to find generalizations by prohibiting decision trees to gather data which is idiosyncratic to the training set.

This fitness adjustment is active in two contexts, both with the same threshold. First, it is used when computing a decision tree's absolute fitness (that is, over the entire training set, uniformly). Second, when computing the competitive (i.e. relative) fitness of a decision tree, it is applied by looking at the partition of the set of training cases selected for that particular fitness measure, which is also a 513 member set, but is not uniform, i.e. it can contain zero copies of some training cases, and more than one copy of other training cases. Therefore, when computing competitive fitness, the penalization is based on an *approximation* of the partition sizes implied by the decision tree. However, since tournament selection is used to select training cases, the distribution of training cases is less skewed than it could be for fitness proportional selection. The penalty is not used when evaluating a decision tree over the test cases.

Note that this strategy for increasing generalization performance is not a change to the evolutionary process, but simply a change to the fitness measure. Also, although this

[6]Since the rule defining a partition is not necessarily correct for every training case it applies to, the amount it contributes to raw fitness can be different (less) than the size of its partition.

Table 19.6
Results from 4 batches of runs, all with competition. See table 19.1 for the parameters used for these runs. The fourth row shows the performance of the tree in figure 19.1, illustrating the standard deviation which results from the random partitioning of the example data into training and test sets.

Threshold on size of *rule partitions*	Number of runs	Average score over test cases	Standard deviation	Average score over training cases	Standard deviation
0	42	79.12%	1.72	84.09%	1.31
3	58	79.20%	1.71	81.84%	1.22
4	35	78.88%	1.73	80.78%	1.18
N/A *(Tree in fig 19.1)*	100	78.99%	1.23		

penalty bears similarity to a parsimony factor, it does not directly penalize a tree based on its size.

Table 19.6 shows the results of experiments with particular thresholds. Section 19.8.3 discusses these results, and section 19.9 suggests more sophisticated methods of penalizing raw fitness.

19.8 Conclusions

19.8.1 Non-trivial Learning and Generalization Performance

The existence of the decision tree in Figure 19.1, which is small yet achieves a high success rate, adds to the difficulty of this problem domain. It is easy to induce a strategy similar to the one embodied by the small decision tree, even by random search, so every run accomplishes at least that. This weakens the comparisons made between different fitness measures, since the range of possible performance is small. Additionally, the loss in performance over the test cases as compared to the performance over the training cases is just enough that the average test score is comparable to the performance of the small tree in Figure 19.1. This is illustrated in Tables 19.6 and 19.5.

It is important to recognize that a non-trivial task is taking place when evolving a decision tree with a higher success rate than that in Figure 19.1. High scoring decision trees implicitly partition the training and test sets into portions which are mostly non-trivial in size. (See section 19.7 for a description of how decision trees partition the training examples.) For example, the decision tree in Figure 19.3 partitioned the test set into partitions of sizes 12, 130, 4, 5, 1, 2, 11, 17, 32, 97, 21, 6, 14, 5, 1, 31, 15, 94, 1, and 15. Therefore, it would be a mistake to assume that the rules embodied in an evolved decision tree other than the simple rules of the tree in Figure 19.1 are exactly the ones which fail when evaluating the decision tree over the test cases; each rule of an evolved decision tree tends to perform more poorly over the test cases. It is *coincidental* that the average score over the test set is approximate to the score attained by the decision tree in Figure 19.1. In domains without a small, high-scoring tree, evolved decision trees will outperform simple trees over the training set to a greater degree. When this is the case, the relative loss of performance over the test cases will likely not bring performance below that of any simple decision tree.

19.8.2 Competition

The mean training score for the best decision tree found over 500 generations of competitive evolution was significantly different from the mean training score for trials without competition (t=6.760, P<.001). The improvement over test cases with competition is less obvious, however the mean test score for competitive evolution with a threshold of 3 (Table 19.6, second row) was significantly different from the mean test score for trials without competition and with a threshold of 0 (Table 19.5, last row) (P<.0265). The usefulness of competition will prove to be dependent on the domain to which it is applied.

19.8.3 Fitness Penalty

Table 19.6 compares the average train and test performances attained when the threshold on rule partition size is set to 0 (i.e. no fitness penalty), 3 and 4. The usefulness of the fitness penalty for this domain is inconclusive, however the results are informative. With a threshold of 3 or 4, learning is inhibited and the average training score is less than that with a threshold of 0. A higher average training score is expected to correspond to a higher average test score. This can be verified by cross-referencing Tables 19.4 and 19.5. However, the average test score attained with a threshold of 3 is not lower than the average test score attained with a threshold of 0. That is, the difference between average training score and average test score is smaller when the penalty is in use. One way to view this is that the penalty decreases learning potential, but also decreases overlearning. It is possible that with the fitness penalty, many extraneous rules which help a tree's training score, but do not help its test score, are "trimmed".

Since the classification problem in this chapter has only two classes, a rule which has overtuned to the training set (i.e. only applies to a small number of training examples) has as least a 50% chance of correctly classifying a test case it applies to. Therefore, in a classification problem with more than two classes, the negative effects of overlearning will probably prove to be more detrimental. See section 19.9 for variations which could increase the usefulness of a fitness penalty.

19.8.4 Linguistic Data

Evolved decision trees often include rules which provide insightful hints for linguists. For example, the decision tree in Figure 19.3 contains a rule that *and* followed by *in* is of class *discourse*. In looking at the training cases we note that the *in* always prefaces the prepositional phrases *in particular*, *in fact*, and *in a certain respect* when following *and*. These are cases in which *and* is being used to introduce an elaboration. As another example, some decision trees contain the rule that *say* preceded by *to* is of class *sentential*. This is linguistically viable, since, when preceded by *to*, *say* is most likely a verb, as in, *"That is what I wanted to say."*

19.9 Further Work

There are ways to vary the method by which competition has been implemented for induction over a fixed set of training cases. For one, higher selection pressure for selecting training cases by way of greater than two-member tournament selection or fitness-proportional selection should be evaluated for various domains. Also, it may be beneficial to have the fitness scores of training cases change only at generation boundaries, so that their adaptation is synchronous with the adaptation of the decision trees.

Various fitness penalties should be contrasted for evolving decision trees. In particular, instead of an absolute threshold, a weighted penalty could be implemented by which the bigger the partitions of a decision tree are, the smaller the fitness penalty. The weight would have to be tuned in a domain-specific manner.

The method by which competition is implemented could influence generalization performance. Other parameters which have potential to influence performance over the test set include the fitness penalty weight and the token frequency thresholds mentioned in section 19.4. Schaffer et al. [1990] have used a GA to tune parameters to increase the performance of a neural network over test cases after back-propagation. A similar method could be employed to tune the parameters listed above, i.e. meta-GA.

Ryan's chapter in this book discusses a method by which diversity can be maintained when a parsimony factor is in use. This method could also apply to a penalty based on the partition sizes implied by a decision tree; this penalty bears similarity to a parsimony factor.

Automatically attained statistical data concerning how often words co-occur (e.g. Hatzivassiloglou and McKeown [1993] and Schuetze [1992]) can aid predictive tasks such as word sense disambiguation. [Brown et al. 1991] We intend to evolve disambiguation mechanisms which have access to such data.

Acknowledgments

Thanks to Andy Singleton, Alex Chaffee, David Schaffer and Kathy McKeown for their supportive exchange of ideas. Thanks to Diane Litman for providing the transcript of spoken English used in this work.

Bibliography

Angeline, P. J. and Pollack, J. B., (1993) Competitive Environments Evolve Better Solutions for Complex Tasks, In *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.

Axelrod, R. (1989) Evolution of strategies in the iterated prisoner's dilemma. *Genetic Algorithms and Simulated Annealing*, L. Davis editor, Morgan Kaufmann.

Brown, P. F., DellaPietra, S. A., DellaPietra, V. J., and Mercer, R. L., (1991) Word sense disambiguation using statistical methods, in *Proceedings 29th Annual Meeting of the Association for Computational Linguistics*, (Berkeley, CA), pp. 265-270, June 1991.

Hatzivassiloglou, V. and McKeown, K., (1993) Towards the Automatic Identification of Adjectival Scales: Clustering Adjectives According to Meaning. *Proceedings of the 31st Annual Meeting of the ACL*, Association for Computational Linguistics, Columbus, Ohio, June 1993.

Hillis, D. (1992) Co-evolving Parasites Improves Simulated Evolution as an Optimization Procedure, In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Hirschberg, J. and Litman, D., (1993) Empirical Studies on the Disambiguation of Cue Phrases, in *Computational Linguistics*, Vol. 19, No. 3, in press.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: The University of Michigan Press.

Christie, A. M. (1993) Induction of decision trees from noisy examples, in *AI Expert*, 5(8).

Koza, J. R. (1991) Concept formation and decision tree induction using the genetic programming paradigm. In Schwefel, Hans-Paul, and Maenner, Reinhard (editors), *Parallel Problem Solving from Nature*. Berlin, Germany: Springer-Verlag.

Koza, J. R. (1992a) *Genetic programming:On the programming of computers by mean of natural selection*. Cambridge, MA: MIT press.

Koza, J. R. (1992b) Genetic Evolution and Co-Evolution of Computer Programs. In *Artificial Life II*, edited by C. Langton, C. Taylor, J. Farmer and S. Rasmussen. Reading, MA: Addison-Wesley Publishing Company, Inc.

Quinlan, J.R. (1986) Induction of decision trees. *Machine Learning* 1(1), New York: Kluwer Academic Publishers, 1986, pp. 81-106.

Schaffer, D., Caruana, R. A. and Eshelman, L. J., (1990) Using Genetic Search to Exploit the Emergent Behavior of Neural Networks. Physica D 42, p. 244-248.

Schuetze, H. (1992) Dimensions of meaning. In *Proceedings of Supercomputing '92*.

Tackett, W. A. (1993) Genetic Programming for Feature Discovery and Image Discrimination. In *Proceedings of the Fifth International Conference on Genetic Algorithms*. San Mateo, CA: Morgan Kaufmann.