To appear in AAAI-96

Lazy Decision Trees

Jerome H. Friedman
Statistics Department and
Stanford Linear Accelerator Center
Stanford University
Stanford, CA 94305
jhf@playfair.stanford.edu

Ron Kohavi
Data Mining and Visualization
Silicon Graphics, Inc.
2011 N. Shoreline Blvd
Mountain View, CA 94043-1389
ronnyk@sgi.com

Yeogirl Yun
Electrical Engineering Department
Stanford University
Stanford, CA 94305
yygirl@cs.stanford.edu

Abstract

Lazy learning algorithms, exemplified by nearest-neighbor algorithms, do not induce a concise hypothesis from a given training set; the inductive process is delayed until a test instance is given. Algorithms for constructing decision trees, such as C4.5, ID3, and CART create a single "best" decision tree during the training phase, and this tree is then used to classify test instances. The tests at the nodes of the constructed tree are good on average, but there may be better tests for classifying a specific instance. We propose a lazy decision tree algorithm—LazyDT—that conceptually constructs the "best" decision tree for each test instance. In practice, only a path needs to be constructed, and a caching scheme makes the algorithm fast. The algorithm is robust with respect to missing values without resorting to the complicated methods usually seen in induction of decision trees. Experiments on real and artificial problems are presented.

Introduction

Delay is preferable to error.
—Thomas Jefferson (1743-1826)

The task of a supervised learning algorithm is to build a classifier that can be used to classify unlabelled instances accurately. Eager (non-lazy) algorithms construct classifiers that contain an explicit hypothesis mapping unlabelled instances to their predicted labels. A decision tree classifier, for example, uses a stored decision tree to classify instances by tracing the instance through the tests at the interior nodes until a leaf containing the label is reached. In eager algorithms, the inductive process is attributed to the phase that builds the classifier. Lazy algorithms (Aha to appear), however, do not construct an explicit hypothesis, and the inductive process can be attributed to the classifier, which is given access to the training set, possibly preprocessed (*e.g.*, data may be normalized). No explicit mapping is generated and the classifier must use the training set to map each given instance to its label.

Building a single classifier that is good for all predictions may not take advantage of special characteristics of the given test instance that may give rise to an extremely short explanation tailored to the specific instance at hand (see Example 1).

In this paper, we introduce a new lazy algorithm—LazyDT—that conceptually constructs the "best" decision tree for each test instance. In practice, only a path needs to be constructed, and a caching scheme makes the algorithm fast. Practical algorithms need to deal with missing values, and LazyDT naturally handles them without resorting to the complicated methods usually seen in induction of decision trees (*e.g.*, sending portions of instances down different branches or using surrogate features).

Decision Trees and Their Limitations

Top down algorithms for inducing decision trees usually follow the *divide and conquer* strategy (Quinlan 1993; Breiman *et al.* 1984). The heart of these algorithms is the test selection, *i.e.*, which test to conduct at a given node. Numerous selection measures exist in the literature, with entropy measures and the Gini index being the most common.

We now detail the entropy-based selection measure commonly used in ID3 and its descendants (*e.g.*, C4.5) because the LazyDT algorithm uses a related measure. We will then discuss some of the limitations of eager decision tree algorithms and motivate our lazy approach.

Test Selection in Decision Trees

To describe the entropy-based selection measure, we follow the notation of Cover & Thomas (1991). Let Y be a discrete random variable with range $\mathcal{Y}$; the *entropy* of Y, sometimes called the *information* of Y,

A longer version of this paper is available at
http://robotics.stanford.edu/~ronnyk

is defined as

$$H(Y) = -\sum_{y\in\mathcal{Y}} p(y)\log p(y)\ , \tag{1}$$

where $0\log 0 = 0$ and the base of the log is usually two so that entropy is expressed in bits. The entropy is always non-negative and measures the amount of uncertainty of the random variable Y. It is bounded by $\log|\mathcal{Y}|$ with equality only if Y is uniformly distributed over $\mathcal{Y}$.

The *conditional entropy* of a variable Y given another variable X is the expected value of the entropies of the conditional distributions averaged over the conditioning random variable:

$$\begin{aligned} H(Y \mid X) &= -\sum_{x\in\mathcal{X}} p(x) H(Y \mid X = x) & (2)\\ &= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y \mid x)\log p(y \mid x) & (3)\\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y \mid x)\ . & (4)\end{aligned}$$

Note that $H(Y \mid X) \neq H(X \mid Y)$.

The *mutual information* of two random variables Y and X, sometimes called the *information gain* of Y given X, measures the relative entropy between the joint distribution and the product distribution:

$$\begin{aligned} I(Y;X) &= \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} & (5)\\ &= H(Y) - H(Y \mid X)\ . & (6)\end{aligned}$$

The mutual information is symmetric, *i.e.*, $I(Y;X) = I(X;Y)$, and non-negative (Cover & Thomas 1991). As can be seen from Equation 6, the mutual information measures the reduction in uncertainty in Y after observing X. Given a set of instances, the above quantities can be computed by using the empirical probabilities, with the variable Y representing the class labels and X a given feature variable.

The test selection step of common decision tree algorithms is implemented by testing the mutual information (or a similar measure) for each feature X with the class label Y and picking the one with the highest value (highest information gain).

Many eager decision tree algorithms, such as C4.5 and CART, have a post-processing step that prunes the tree to avoid *overfitting*. The reader is referred to Quinlan (1993) and Breiman *et al.* (1984) for the two most common pruning mechanisms. The current implementation of our LazyDT algorithm does no pruning because there is no simple analog between pruning in lazy decision trees and pruning in ordinary decision trees.

Problems with Decision Trees

The problems with decision trees can be divided into two categories: algorithmic problems that complicate the algorithm's goal of finding a small tree and inherent problems with the representation.

Top-down decision-tree induction algorithms implement a greedy approach that attempts to find a small tree. All the common selection measures are based on one level of lookahead.

Two related problems inherent to the representation structure are *replication* and *fragmentation* (Pagallo & Haussler 1990). The replication problem forces duplication of subtrees in disjunctive concepts, such as $(A \wedge B) \vee (C \wedge D)$ (one subtree, either $A \wedge B$ or $C \wedge D$ must be duplicated in the smallest possible decision tree); the fragmentation problem causes partitioning of the data into smaller fragments. Replication always implies fragmentation, but fragmentation may happen without any replication if many features need to be tested. For example, if the data splits approximately equally on every split, then a univariate decision tree cannot test more than $O(\log n)$ features. This puts decision trees at a disadvantage for tasks with many relevant features.

A third problem inherent to the representation is the ability to deal with missing values (unknown values). The correct branch to take is unknown if a feature tested is missing, and algorithms must employ special mechanisms to handle missing values. In order to reduce the occurrences of tests on missing values, C4.5 penalizes the information gain by the proportion of unknown instances and then splits these instances to both subtrees. CART uses a much more complex scheme of surrogate features. Friedman estimated that about half the code in CART and about 80% of the programming effort went into missing values!

Lazy Decision Trees

We now introduce LazyDT, a lazy algorithm for inducing decision trees. We begin with general motivation and compare the advantages and disadvantages of the lazy construction of decision trees to that of the eager approach. We then describe the specific algorithmic details and the caching scheme that is used to speed up classification.

Motivation

A single decision tree built from the training set is making a compromise: the test at the root of each subtree is chosen to be the best split *on average*. Common feature selection criteria, such as mutual information and the Gini index, average the purity of the children by the proportions of instances in those children. Entropy measures used in C4.5 and ID3 are guaranteed to decrease on average (*i.e.*, the information gain is non-negative) but the entropy of a specific child may not change or may increase. A single tree built in advance can lead to many irrelevant splits for a given test instance, thus fragmenting the data unnecessarily. Such fragmentation reduces the significance of tests at lower levels since they are based on fewer instances. A decision tree built for the given instance can avoid splits on features that are irrelevant for the specific instance.

Example 1 Suppose a domain requires one to classify patients as *sick* or *healthy*. A Boolean feature denoting whether a person is HIV positive is extremely relevant. (For this example we will assume that such persons should be classified as sick.)

Even though all instances having HIV positive set to *true* have the same class, a decision tree is unlikely to make the root test based on this feature because the proportion of these instances is so small; the conditional (or average) entropy of the two children of a test on the HIV-positive feature will not be much different from the parent and hence the information gain will be small. It is therefore likely that the HIV-positive instances will be fragmented throughout the nodes in the tree. Moreover, many branches that contain such instances will need to branch on the HIV-positive feature lower down the tree, resulting in the replication of tests. ▮

The example leads to an interesting observation: trees, or rather classification paths, built for a specific test instance may be much shorter and hence may provide a short explanation for the classification. A person that is healthy might be explained by a path testing fever, blood-cell counts, and a few other features that fall within the normal ranges. A person might be classified as sick with the simple explanation that he or she is HIV positive.

Another advantage to lazy decision trees is the natural way in which missing values are handled. Missing feature values require special handling by decision tree classifiers, but a decision tree built for the given instance simply need never branch on a value missing in that instance, thus avoiding unnecessary fragmentation of the data.

The Framework for Lazy Decision Trees

We now describe the general framework for a lazy decision tree classifier and some pitfalls associated with using common selection measures, such as mutual information or the Gini index. We assume the data has been discretized and that all features are nominal.

Input: A training set T of labelled instances and an unlabelled instance I to classify.
Output: A label for instance I.

1. If T is pure, *i.e.*, all instances in T have label ℓ, return ℓ.
2. Otherwise, if all instances in T have the same feature values, return the majority class in T.
3. Otherwise, select a test X and let x be the value of the test on the instance I. Assign the set of instances with $X = x$ to T and apply the algorithm to T.

Figure 1: The generic lazy decision trees algorithm.

As with many lazy algorithms, the first part of the induction process (*i.e.*, building a classifier) is non-existent; all the work is done during the classification of a given instance.

The lazy decision tree algorithm, which gets the test instance as part of the input, follows a *separate and classify* methodology: a test is selected and the sub-problem containing the instances with the same test outcome as the given instance is then solved recursively. The overall effect is that of tracing a path in an imaginary tree made specifically for the test instance. Figure 1 shows a generic lazy decision tree algorithm.

The heart of the algorithm is the selection of a test to conduct at each recursive call. Common measures used in decision tree algorithms usually indicate the average gain in information after the outcome of the chosen test is taken into account. Because the lazy decision tree algorithm is **given extra information,** namely the (unlabelled) test instance, one would like to use that information to choose the appropriate test.

The simplest approach that comes to mind is to find the test that maximally decreases the entropy for the node our test instance would branch to and define the *information gain* to be the difference between the two entropies. There are two problems with this approach: the first is that the information gain can be negative, in which case it is not clear what to do with negative gains. If class A is dominant but class B is the correct class, then it may be necessary for the created path to go through a node with equal frequencies before class B becomes the majority class. This means that avoiding splits on features that have negative gain is a mistake. A second, related problem, is that only the frequencies are taken into account, not the actual classes. If the parent node has 80% class A and 20% class B and the child node has 80% class B and 20%

class A, then there will be no information gain (the entropy will be the same), but the feature tested at the parent is clearly relevant.

In light of these problems, we *normalize* the class probabilities at every node by re-weighting the instances such that each class has equal weight. The normalization scheme solves both problems. The entropy of the re-weighted parent node will be $\log k$, where k is the number of classes (see Equation 1 and the text following it). The normalization implies that the information gain will always be positive and that the 80%/20% split described above will have large information gain.

The LazyDT Algorithm

We now describe the exact details of the LazyDT algorithm including the way it handles continuous features and the caching scheme used to speed the classification.

Since the LazyDT algorithm described is only capable of processing nominal features, the training set is first discretized (Dougherty, Kohavi, & Sahami 1995). We chose to discretize the instances using recursive minimization of entropy as proposed by Fayyad & Irani (1993) and as implemented in $\mathcal{MLC}$++ (Kohavi *et al.* 1994), which is publicly available and thus allows replication of this discretization step. The exact details are unimportant for this paper.

We considered two univariate test criteria. The first is similar to that of C4.5 (*i.e.*, a multi-way split). The second is a binary split on a single value. To avoid fragmentation as much as possible, we chose the second method and have allowed splitting on any feature value that is *not* equal to the instance's value. For example, if the instance has feature A with value a and the domain of A is $\{a, b, c\}$, then we allow a split on $A = b$ (two branches, one for equality, one for non-equality) and a split on $A = c$. For non-binary features, this splitting method makes more splits, but the number of instances that are split off each time is smaller.

Missing feature values are naturally handled by considering only splits on feature values that are known in the test instance. Training instances with unknowns filter down and are excluded only when their value is unknown for a given test in a path. Avoiding any tests on unknown values is the correct thing to do probabilistically, assuming the values are truly unknown (as opposed to unknown because there was a reason for not measuring them).

The LazyDT algorithm proceeds by splitting the instances on tests at nodes as described in the previous section. Because we found that there are many ties between features with very similar information gains, we call the algorithm recursively for all features with information gains higher than 90% of the highest gain achievable. The recursive call that returns with the highest number of instances in the majority class of a leaf node that was reached makes the final prediction (ties from the recursive calls are broken arbitrarily).

As defined, the algorithm is rather slow. For each instance, all splits must be considered (a reasonably fast process if the appropriate counters are kept), but each split then takes time proportional to the number of training instances that filtered to the given node. This implies that the time complexity of classifying a given instance is $O(m \cdot n \cdot d)$ for m instances, n features, and a path of depth d. If we make the reasonable assumption that at least some fixed fraction of the instances are removed at each split (say 10%), then the time complexity is $O(m \cdot n)$. In order to speed up the process in practice, we cache the information gains and create lists of pointers to instances, representing the sets of instances that filter to each node. After a few instances have been classified, commonly used paths already exist, and the calculations need not be repeated, especially at higher levels. The caching scheme was found to be very efficient time-wise, but it consumes a lot of memory.

Experiments

We now describe experiments that compare LazyDT with other algorithms for inducing decision trees.

The Algorithms and Datasets

We compare LazyDT to three algorithms: simple ID3, C4.5, and C4.5-NP. *Simple ID3* is a basic basic top-down induction of decision trees algorithm. It selects the features based on information gain and considers unknowns to be a separate value. *C4.5* (Quinlan 1993) is a state-of-the-art algorithm that penalizes multi-way splits using the gain-ratio, prunes the tree, and splits every instance into multiple branches when hitting unknown values. We used the default parameter settings. *C4.5-NP* is C4.5 without pruning and it is compared in order to estimate the effect of pruning. Because LazyDT does not prune, the difference between C4.5 and C4.5-NP might indicate that there is similar room for improvement to LazyDT if a pruning algorithm were added.

The datasets we use are common ones used in the literature and stored in the U.C. Irvine repository (Murphy & Aha 1996). The estimated prediction accuracy was computed by doing five-fold cross-validation for all domains except the artificial domains where a standard training set was used and the test set was the complete instance space.

Results and Discussion

Characteristics of the datasets and accuracy results are shown in Table 1, and a graph presenting the difference in accuracies and standard deviations is shown in Figure 2.

The LazyDT algorithm is a reasonably fast algorithm. The largest running time by far was for mushroom with 8.4 Sparc-10 cpu minutes per cross-validation fold (equivalent to a run), followed by chess with 1.59 cpu minutes. These datasets have 8124 instances and 3196 instances, respectively.

From the table we can see that simple ID3 is generally inferior, as is C4.5 without pruning. Pruning improves C4.5-NP's performance, except for a few cases. The LazyDT algorithm and C4.5 (with pruning) behave somewhat similarly but there are some datasets that have large differences. The LazyDT's average error rate is 1.9% lower, which is a relative improvement in error of 10.6% over C4.5's 17.9% average error rate. Three datasets deserve special discussion: anneal, audiology, and the monk2 problem.

Anneal is interesting because ID3 manages so well. An investigation of the problem shows that the main difference stems from the dissimilar ways in which unknown values are handled. Simple ID3 considers unknowns as a separate value whereas C4.5 has a special mechanism for handling unknowns. In this dataset, changing the unknown values into a separate feature value improves the performance of C4.5 to 98.7%. Schaffer (1993) showed that neural nets considerably outperformed C4.5 on the anneal dataset, but we can now attribute this difference to the fact that for backpropagation Schaffer has converted the unknown values to an additional discrete value.

The second file we discuss is audiology. The performance of LazyDT on this dataset is significantly lower than that of C4.5. This dataset has 69 features, 24 classes, and only 226 instances. LazyDT is likely to find a pure class on one of the features because of the small number of instances. Thus the extra flexibility to branch differently depending on the test instance hurts LazyDT in cases such as audiology. This is a bias-variance tradeoff (Kohavi & Wolpert 1996; Geman, Bienenstock, & Doursat 1992) and to overcome such cases we would have to bias the algorithm to avoid early splits that leave only a few instances to classify the test instance.

The final file to discuss is monk2, where the performance of LazyDT is superior to that of C4.5. The monk2 problem is artificial and the concept is that any two features (and only two) have to have their first value. Quinlan (1993) writes that "[The monk2 problem] is just plain difficult to express either as trees or as rules... all classifiers generated by the programs are very poor." While the problem is hard to represent in a univariate decision tree, the flexibility of LazyDT (which is still restricted to univariate splits), is helpful here. The root test in the examined runs indeed tends to pick a feature whose value is not equal to the first value and thus separate those instances from the rest.

Missing Values

To test the robustness of LazyDT to missing values, we added noise to the datasets. The "noise process" changes each feature value to unknown with the 20% probability. The average accuracy over all the datasets changed as follows: ID3's accuracy decreased to 68.22%; C4.5's accuracy decreased to 77.10%, and LazyDT's accuracy decreased to 77.81%.

Some of the biggest differences between the accuracy on the original datasets and the corrupted datasets occur on the artificial datasets: monk1, monk2, and monk3; and pseudo-artificial datasets: tic-tac-toe, and chess. Hayes-roth and glass2 also have large differences probably because they have many strongly relevant features and few weakly relevant features (John, Kohavi, & Pfleger 1994). If we ignore the artificial problems, the average accuracy for LazyDT on the datasets without missing values is 82.15% and the accuracy on the datasets with 20% missing values is 78.40%. Thus there is less than 4% reduction in performance when 20% of the feature values are missing.

With many missing values pruning may be important, but our current implementation of LazyDT does no pruning. For example, the worse difference in accuracy on the corrupted datasets is on the breast (L) dataset. LazyDT overfits the data and has accuracy of 61.54% while majority is 70.28%.

Related Work

Most work on lazy learning was motivated by nearest-neighbor algorithms (Dasarathy 1990; Wettschereck 1994; Aha to appear). LazyDT was motivated by Friedman (1994), who defined a (separate) distance metric for a nearest-neighbor classifier based on the respective relevance of each feature for classifying each particular test instance. Subsequently, Hastie & Tibshirani (1995) proposed using local linear discriminant methods to define nearest-neighbor metrics. Both these methods are intended for continuous features and thus they can control the number of instances removed at each step. In contrast, the caching scheme used by LazyDT cannot be applied with these methods, and hence they are much slower.

Smyth & Goodman (1992) described the use of the J-measure, which is the inner sum of Equation 4. The

Table 1: Comparison of the accuracy of simple ID3, C4.5 with no pruning, C4.5 with pruning, and LazyDT. The number after the ± indicates one standard error of the cross-validation folds. The table is sorted by difference between LazyDT and C4.5.

No.	Dataset	Feat-ures	Train sizes	Test	Simple ID3 accuracy	C4.5-NP accuracy	C4.5 accuracy	LazyDT accuracy
1	monk-2	6	169	432	69.91±2.21	65.30±2.29	65.00±2.30	82.18±1.84
2	monk-1	6	124	432	81.25±1.89	76.60±2.04	75.70±2.07	91.90±1.31
3	tic-tac-toe	9	958	CV-5	84.38±2.62	85.59±1.35	84.02±1.56	93.63±0.83
4	cleve	13	303	CV-5	63.93±6.20	72.97±2.50	73.62±2.25	81.21±3.55
5	glass	9	214	CV-5	62.79±7.46	64.47±2.73	65.89±2.38	72.92±1.81
6	hayes-roth	4	160	CV-5	68.75±8.33	71.25±3.03	74.38±4.24	78.75±1.53
7	glass2	9	163	CV-5	81.82±6.82	72.97±4.05	73.60±4.06	77.92±1.11
8	anneal	24	898	CV-5	100.00±0.00	94.10±0.45	91.65±1.60	95.77±0.87
9	heart	13	270	CV-5	77.78±5.71	75.93±3.75	77.04±2.84	81.11±2.89
10	diabetes	8	768	CV-5	66.23±3.82	67.72±2.76	70.84±1.67	74.48±1.27
11	soybean-small	35	47	CV-5	100.00±0.00	95.56±2.72	95.56±2.72	97.78±2.22
12	labor-neg	16	57	CV-t	83.33±11.2	79.09±5.25	77.42±6.48	79.09±4.24
13	lymphography	18	148	CV-5	73.33±8.21	74.97±2.98	77.01±0.77	78.41±2.19
14	hepatitis	19	155	CV-5	67.74±8.54	76.77±4.83	78.06±2.77	79.35±2.41
15	german	24	1000	CV-5	63.00±3.43	70.20±1.70	72.30±1.37	73.50±1.57
16	pima	8	768	CV-5	67.53±3.79	69.79±1.68	72.65±1.78	73.70±1.58
17	mushroom	22	8124	CV-5	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
18	iris	4	150	CV-5	96.67±3.33	95.33±0.82	94.67±1.33	94.67±0.82
19	vote	16	435	CV-5	93.10±2.73	94.71±0.59	95.63±0.43	95.17±0.76
20	monk-3	6	122	432	90.28±1.43	92.60±1.26	97.20±0.80	96.53±0.88
21	chess	36	3196	CV-5	99.69±0.22	99.31±0.15	99.34±0.12	98.22±0.21
22	breast-(W)	10	699	CV-5	95.71±1.72	93.99±1.05	94.71±0.37	92.99±0.69
23	breast-(L)	9	286	CV-5	62.07±6.43	64.34±1.67	71.00±2.25	68.55±2.86
24	horse-colic	22	368	CV-5	75.68±5.02	82.88±2.15	84.78±1.31	82.07±1.79
25	australian	14	690	CV-5	78.26±3.52	82.90±1.14	85.36±0.74	81.74±1.56
26	crx	15	690	CV-5	79.71±3.44	83.62±1.35	85.80±0.99	82.03±0.87
27	vote1	15	435	CV-5	85.06±3.84	86.44±2.00	86.67±1.13	81.84±1.56
28	audiology	69	226	CV-5	80.43±5.91	76.52±3.30	78.74±3.05	66.36±1.69
	Average				80.30	80.93	82.09	84.00

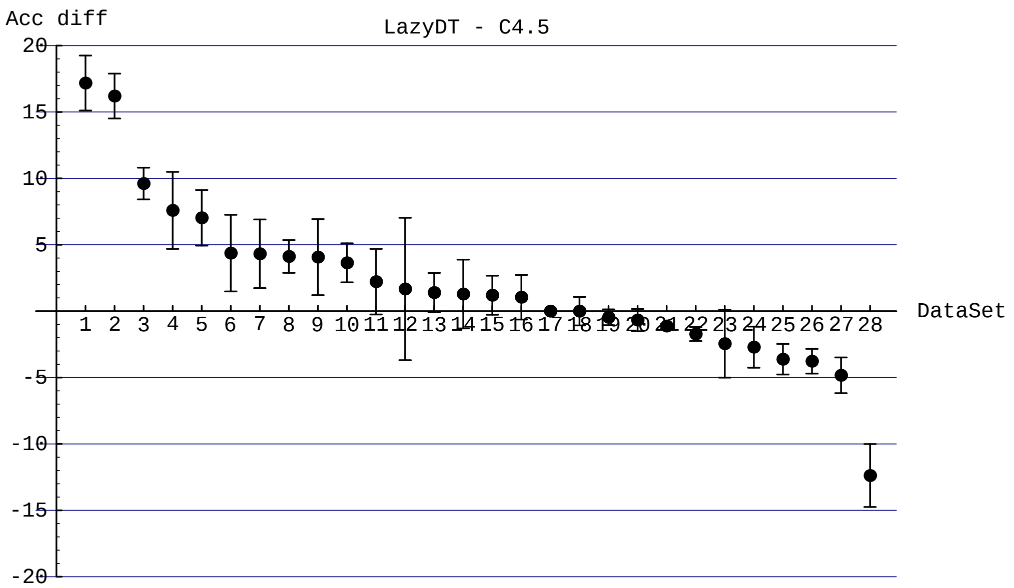

Figure 2: The difference between the accuracy of LazyDT and C4.5. Positive values indicate LazyDT outperforms C4.5. Error bars indicate one standard deviation.

J-measure can be used in LazyDT because it was shown to be non-negative. However, initial experiments showed it was slightly inferior on the tested datasets. Perhaps our measure would be useful in systems where the J-measure is currently used (*e.g.*, ITrule).

Holte, Acker, & Porter (1989) noted that existing inductive systems create definitions that are good for large disjuncts but are far from ideal for small disjuncts, where a disjunct is a conjunction that correctly classifies few training examples. It is hard to assess the accuracy of small disjuncts because they cover few examples, yet removing all of them without significance tests is unjustified since many of them are significant and the overall accuracy would degrade. The authors propose a *selective specificity* bias and present mixed results; Quinlan (1991) suggests an improved estimate that also takes into account the proportion of the classes in the *context* of the small disjunct. We believe that LazyDT suffers less from the problem of small disjuncts because the training set is being "fitted" to the specific instance and hence is likely to be less fragmented. The normalization of class probabilities in LazyDT is in line with Quinlan's suggestions (Quinlan 1991) of taking the context (the parent node in our case) into account.

Quinlan (1994) characterizes classification problems as sequential or parallel. In parallel tasks, all input features are relevant to the classification; in sequential type tasks, the relevance of features depends on the values of other features. Quinlan conjectures that parallel type tasks are unsuitable for current univariate decision-tree methods because it is rare that there are enough instances for doing splits on all the n relevant features; similarly, he claims that sequential type tasks require inordinate amounts of learning time for backpropagation based methods because if a feature i is irrelevant, inopportune adjustment to a weight w_{ij} will tend to obscure the sensible adjustments made when the feature is relevant. LazyDT might be inferior to backpropagation and nearest-neighbor methods on some parallel tasks with many relevant features, but it should fare better than decision trees. Good examples are the monk2 and tic-tac-toe domains: all features are relevant, but if a split is to be made on all features, there will not be enough instances. LazyDT makes the relevant splits based on the feature values in the test-instance and thus fragments the data less.

Future Work

LazyDT is a new algorithm in the arena of machine learning. The weakest point of our algorithm is the fact that it does no regularization (pruning). The australian dataset has 14 features, but the background knowledge file describes five features as the important ones. If we allow LazyDT to use only those five features, its accuracy increases from 81.7% to 85.6%. An even more extreme case is breast-cancer (L), where removal of all features improves performance (*i.e.*, majority is a better classifier).

Data is currently discretized in advance. The discretization algorithm seems to be doing a good job, but since it is a pre-processing algorithm, it is not taking advantage of the test instance. It is possible to extend LazyDT to decide on the threshold during classification, as in common decision tree algorithms, but the caching scheme would need to be modified.

The caching algorithm currently remembers all tree paths created, thus consuming a lot of memory for files with many features and many instances. An enhancement might be made to allow for some space-time tradeoff. In practice, of course, the caching scheme might be avoided altogether; a doctor, for example, can wait a few seconds for classification. Our experiments required hundreds of test instances to be classified for twenty-eight datasets, so caching was a necessity.

The dynamic complexity of an algorithm (Holte 1993) is the number of features used on average. An interesting experiment would be to compare the dynamic complexity of C4.5 with that of LazyDT.

Summary

We introduced a novel lazy algorithm, LazyDT, that can be used in supervised classification. This algorithm differs from common lazy algorithms that are usually based on a global nearest-neighbor metric. LazyDT creates a path in a tree that would be "best" for a given test instance, thus mitigating the fragmentation problem.

Empirical comparisons with C4.5, the state-of-the-art decision tree algorithm, show that the performance is slightly higher on the tested datasets from the U.C. Irvine repository. However, since no algorithm can outperform others in all settings (Wolpert 1994; Schaffer 1994), the fact that they exhibit different behavior on many datasets is even more important. For some datasets LazyDT significantly outperforms C4.5 and vice-versa.

Missing feature values are naturally handled by LazyDT with no special handling mechanisms required. Performance on corrupted data is comparable to that of C4.5, which has an extremely good algorithm for dealing with unknown values.

The algorithm is relatively fast due to the caching scheme employed, but requires a lot of memory. We believe that a space-time tradeoff should be investigated and hope to pursue the regularization (pruning) issue in the future.

Acknowledgments We thank George John, Rob Holte, and Pat Langley for their suggestions. The LazyDT algorithm was implemented using the $\mathcal{MLC}$++ library, partly funded by ONR grant N00014-95-1-0669. Jerome H. Friedman's work was supported in part by the Department of Energy under contract number DE-AC03-76SF00515 and by the National Science Foundation under grant number DMS-9403804.

References

Aha, D. W. to appear. AI review journal: Special issue on lazy learning.

Breiman, L.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees.* Wadsworth International Group.

Cover, T. M., and Thomas, J. A. 1991. *Elements of Information Theory.* John Wiley & Sons, Inc.

Dasarathy, B. V. 1990. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, California.

Dougherty, J.; Kohavi, R.; and Sahami, M. 1995. Supervised and unsupervised discretization of continuous features. In Prieditis, A., and Russell, S., eds., *Machine Learning: Proceedings of the Twelfth International Conference*, 194–202. Morgan Kaufmann.

Fayyad, U. M., and Irani, K. B. 1993. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 1022–1027. Morgan Kaufmann Publishers, Inc.

Friedman, J. H. 1994. Flexible metric nearest neighbor classification. Technical Report 113, Stanford University Statistics Department.

Geman, S.; Bienenstock, E.; and Doursat, R. 1992. Neural networks and the bias/variance dilemma. *Neural Computation* 4:1–48.

Hastie, T., and Tibshirani, R. 1995. Discriminant adaptive nearest neighbor classification. Technical report, Stanford University Statistics Department.

Holte, R. C.; Acker, L. E.; and Porter, B. W. 1989. Concept learning and the problem of small disjuncts. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 813–818.

Holte, R. C. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine Learning* 11:63–90.

John, G.; Kohavi, R.; and Pfleger, K. 1994. Irrelevant features and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, 121–129. Morgan Kaufmann.

Kohavi, R., and Wolpert, D. H. 1996. Bias plus variance decomposition for zero-one loss functions. In Saitta, L., ed., *Machine Learning: Proceedings of the Thirteenth International Conference.* Morgan Kaufmann Publishers, Inc. Available at http://robotics.stanford.edu/users/ronnyk.

Kohavi, R.; John, G.; Long, R.; Manley, D.; and Pfleger, K. 1994. MLC++: A machine learning library in C++. In *Tools with Artificial Intelligence*, 740–743. IEEE Computer Society Press.

Murphy, P. M., and Aha, D. W. 1996. UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn`.

Pagallo, G., and Haussler, D. 1990. Boolean feature discovery in empirical learning. *Machine Learning* 5:71–99.

Quinlan, J. R. 1991. Improved estimates for the accuracy of small disjuncts. *Machine Learning* 6:93–98.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* Los Altos, California: Morgan Kaufmann Publishers, Inc.

Quinlan, J. R. 1994. Comparing connectionist and symbolic learning methods. In Hanson, S. J.; Drastal, G. A.; and Rivest, R. L., eds., *Computational Learning Theory and Natural Learning Systems*, volume I: Constraints and Prospects. MIT Press. chapter 15, 445—456.

Schaffer, C. 1993. Selecting a classification method by cross-validation. *Machine Learning* 13(1):135–143.

Schaffer, C. 1994. A conservation law for generalization performance. In *Machine Learning: Proceedings of the Eleventh International Conference*, 259–265. Morgan Kaufmann Publishers, Inc.

Smyth, P., and Goodman, R. 1992. An information theoretic approach to rule induction from databases. *IEEE Transactions on Knowledge and Data Engineering* 4(4):301–316.

Wettschereck, D. 1994. *A Study of Distance-Based Machine Learning Algorithms.* Ph.D. Dissertation, Oregon State University.

Wolpert, D. H. 1994. The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework. In Wolpert, D. H., ed., *The Mathemtatics of Generalization.* Addison Wesley.