Logistic Model Trees

Niels Landwehr[1,2], Mark Hall[2], and Eibe Frank[2]

[1]Department of Computer Science
University of Freiburg
Freiburg, Germany
landwehr@informatik.uni-freiburg.de

[2]Department of Computer Science
University of Waikato
Hamilton, New Zealand
{eibe, mhall}@cs.waikato.ac.nz

Abstract. Tree induction methods and linear models are popular techniques for supervised learning tasks, both for the prediction of nominal classes and continuous numeric values. For predicting numeric quantities, there has been work on combining these two schemes into 'model trees', i.e. trees that contain linear regression functions at the leaves. In this paper, we present an algorithm that adapts this idea for classification problems, using logistic regression instead of linear regression. We use a stagewise fitting process to construct the logistic regression models that can select relevant attributes in the data in a natural way, and show how this approach can be used to build the logistic regression models at the leaves by incrementally refining those constructed at higher levels in the tree. We compare the performance of our algorithm against that of decision trees and logistic regression on 32 benchmark UCI datasets, and show that it achieves a higher classification accuracy on average than the other two methods.

1 Introduction

Two popular methods for classification are linear logistic regression and tree induction, which have somewhat complementary advantages and disadvantages. The former fits a simple (linear) model to the data, and the process of model fitting is quite stable, resulting in low variance but potentially high bias. The latter, on the other hand, exhibits low bias but often high variance: it searches a less restricted space of models, allowing it to capture nonlinear patterns in the data, but making it less stable and prone to overfitting. So it is not surprising that neither of the two methods is superior in general — earlier studies [10] have shown that their relative performance depends on the size and the characteristics of the dataset (e.g., the signal-to-noise ratio).

It is a natural idea to try and combine these two methods into learners that rely on simple regression models if only little and/or noisy data is available and add a more complex tree structure if there is enough data to warrant such structure. For the case of predicting a numeric variable, this has lead to 'model trees', which are decision trees with linear regression models at the leaves. These have been shown to produce good results [11]. Although it is possible to use model trees for classification tasks by transforming the classification problem into a regression task by binarizing the class [4], this approach produces several trees (one per class) and thus makes the final model harder to interpret.

A more natural way to deal with classification tasks is to use a combination of a tree structure and logistic regression models resulting in a single tree. Another advantage of using logistic regression is that explicit class probability estimates are produced rather than just a classification. In this paper, we present a method that follows this idea. We discuss a new scheme for selecting the attributes to be included in the logistic regression models, and introduce a way of building the logistic models at the leaves by refining logistic models that have been trained at higher levels in the tree, i.e. on larger subsets of the training data.

We compare the performance of our method against the decision tree learner C4.5 [12] and logistic regression on 32 UCI datasets [1], looking at classification accuracy and size of the constructed trees. We also include results for two learning schemes that build multiple trees, namely boosted decision trees and model trees fit to the class indicator variables [4], and a different algorithm for building logistic model trees called PLUS [8]. From the results of the experiments we conclude that our method achieves a higher average accuracy than C4.5, model trees, logistic regression and PLUS, and is competitive with boosted trees. We will also show that it smoothly adapts the tree size to the complexity of the data set.

The rest of the paper is organized as follows. In Section 2 we briefly review logistic regression and the model tree algorithm and introduce logistic model trees in more detail. Section 3 describes our experimental study, followed by a discussion of results. We discuss related work in Section 4 and draw some conclusions in Section 5.

2 Algorithms

This section begins with a brief introduction to the application of regression for classification tasks and a description of our implementation of logistic regression. A summary of model tree induction is also provided as this is a good starting point for understanding our method.

2.1 Logistic Regression

Linear regression performs a least-squares fit of a parameter vector β to a numeric target variable to form a model

$$f(x) = \beta^T \cdot x,$$

where x is the input vector (we assume a constant term in the input vector to accommodate the intercept). It is possible to use this technique for classification by directly fitting linear regression models to class indicator variables. If there are J classes then J indicator variables are created and the indicator for class j takes on value 1 whenever class j is present and value 0 otherwise. However, this approach is known to suffer from masking problems in the multiclass setting [7].

1. Start with weights $w_{ij} = 1/n,\ i = 1, \ldots, n,\ j = 1, \ldots, J,\ F_j(x) = 0$ and $p_j(x) = 1/J \ \ \forall j$
2. Repeat for $m = 1, \ldots, M$:
   (a) Repeat for $j = 1, \ldots, J$:
      i. Compute working responses and weights in the jth class
$$z_{ij} = \frac{y^*_{ij} - p_j(x_i)}{p_j(x_i)(1 - p_j(x_i))}$$
$$w_{ij} = p_j(x_i)(1 - p_j(x_i))$$
      ii. Fit the function $f_{mj}(x)$ by a weighted least-squares regression of z_{ij} to x_i with weights w_{ij}
   (b) Set $f_{mj}(x) \leftarrow \frac{J-1}{J}(f_{mj}(x) - \frac{1}{J}\sum_{k=1}^{J} f_{mk}(x)), F_j(x) \leftarrow F_j(x) + f_{mj}(x)$
   (c) Update $p_j(x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}$
3. Output the classifier $\mathrm{argmax}_j F_j(x)$

Fig. 1. LogitBoost algorithm for J classes.

A better method for classification is *linear logistic regression*, which models the posterior class probabilities $Pr(G = j|X = x)$ for the J classes via linear functions in x while at the same time ensuring they sum to one and remain in [0,1]. The model has the form[1]

$$Pr(G = j|X = x) = \frac{e^{F_j(x)}}{\sum_{k=1}^{J} e^{F_k(x)}}, \quad \sum_{k=1}^{J} F_k(x) = 0,$$

where $F_j(x) = \beta_j^T \cdot x$ are linear regression functions, and it is usually fit by finding maximum likelihood estimates for the parameters β_j.

One way to find these estimates is based on the LogitBoost algorithm [6]. LogitBoost performs forward stage-wise fitting of *additive logistic regression models*, which generalize the above model to $F_j(x) = \sum_m f_{mj}(x)$, where the f_{mj} can be arbitrary functions of the input variables that are fit by least squares regression. In our application we are interested in linear models, and LogitBoost finds the maximum likelihood linear logistic model if the f_{mj} are fit using (simple or multiple) linear least squares regression and the algorithm is run until convergence. This is because the likelihood function is convex and LogitBoost performs quasi-Newton steps to find its maximum.

The algorithm (shown in Figure 1) iteratively fits regression functions f_{mj} to a 'response variable' (reweighted residuals). The $x_1, \ldots, x_n$ are the training examples and the y^*_{ij} encode the observed class membership probabilities for instance x_i, i.e. y^*_{ij} is one if x_i is labeled with class j and zero otherwise. One can build the f_{mj} by performing multiple regression based on all attributes present in the data, but it is also possible to use a simple linear regression,

[1] This is the *symmetric* formulation [6].

selecting the attribute that gives the smallest squared error. If the algorithm is run until convergence this will give the same final model because every multiple linear regression function can be expressed as a sum of simple linear regression functions, but using simple regression will slow down the learning process and thus give a better control over model complexity. This allows us to obtain simple models and prevent overfitting of the training data: the model learned after a few iterations (for a small M) will only include the most relevant attributes present in the data, resulting in automatic attribute selection, and, if we use cross-validation to determine the best number of iterations, a new variable will only be added if this improves the performance of the model on unseen cases.

In our empirical evaluation simple regression together with (five fold) cross-validation indeed outperformed multiple regression. Consequently, we chose this approach for our implementation of logistic regression. We will refer to it as the *SimpleLogistic* algorithm.

2.2 Model Trees

Model trees, like ordinary regression trees, predict a numeric value given an instance that is defined over a fixed set of numeric or nominal attributes. Unlike ordinary regression trees, model trees construct a piecewise linear (instead of a piecewise constant) approximation to the target function. The final model tree consists of a decision tree with linear regression models at the leaves, and the prediction for an instance is obtained by sorting it down to a leaf and using the prediction of the linear model associated with that leaf.

The M5' model tree algorithm [13] — a 'rational reconstruction' of Quinlan's M5 algorithm [11] — first constructs a regression tree by recursively splitting the instance space using tests on single attributes that maximally reduce variance in the target variable. After the tree has been grown, a linear multiple regression model is built for every inner node, using the data associated with that node and all the attributes that participate in tests in the subtree rooted at that node. Then the linear regression models are simplified by dropping attributes if this results in a lower expected error on future data (more specifically, if the decrease in the number of parameters outweighs the increase in the observed training error). After this has been done, every subtree is considered for pruning. Pruning occurs if the estimated error for the linear model at the root of a subtree is smaller or equal to the expected error for the subtree. After pruning has terminated, M5' applies a 'smoothing' process that combines the model at a leaf with the models on the path to the root to form the final model that is placed at the leaf.

2.3 Logistic Model Trees

Given this model tree algorithm, it appears quite straightforward to build a 'logistic model tree' by growing a standard classification tree, building logistic regression models for all nodes, pruning some of the subtrees using a pruning criterion, and combining the logistic models along a path into a single model in some fashion. However, the devil is in the details and M5' uses a set of heuristics at crucial points in the algorithm — heuristics that cannot easily be transferred to the classification setting.

Fortunately LogitBoost enables us to view the combination of tree induction and logistic regression from a different perspective: iterative fitting of simple linear regression interleaved with splits on the data. Recall that LogitBoost builds a logistic model by iterative refinement, successively including more and more variables as new linear models f_{mj} are added to the committee F_j. The idea is to recursively split the iterative fitting process into branches corresponding to subsets of the data, a process that automatically generates a tree structure.

As an example, consider a tree with a single split at the root and two leaves. The root node N has training data T and one of its sons N' has a subset of the training data $T' \subset T$. Following the classical approach, there would be a logistic regression model M at node N trained on T and a logistic regression model M' at N' trained on T'. For classification, the class probability estimates of M and M' would be averaged to form the final model for N'.

In our approach, the tree would instead be constructed by building a logistic model M at N by fitting linear regression models trained on T as long as this improves the fit to the data, and then building the logistic model M' at N' by taking M and adding more linear regression models that are trained on T', rather than starting from scratch. As a result, the final model at a leaf consists of a committee of linear regression models that have been trained on increasingly smaller subsets of the data (while going down the tree). Building the logistic regression models in this fashion by refining models built at higher levels in the tree is computationally more efficient than building them from scratch.

However, a practical tree inducer also requires a pruning method. In our experiments 'local' pruning criteria employed by algorithms like C4.5 and M5' did not lead to reliable pruning. Instead, we followed the pruning scheme employed by the CART algorithm [2], which uses cross-validation to obtain more stable pruning results. Although this increased the computational complexity, it resulted in smaller and generally more accurate trees.

These ideas lead to the following algorithm for constructing logistic model trees:

- Tree growing starts by building a logistic model at the root using the LogitBoost algorithm. The number of iterations (and simple regression functions f_{mj} to add to F_j) is determined using five fold cross-validation. In this process the data is split into training and test set five times, for every training set LogitBoost is run to a maximum number of iterations (we used 200) and the error rates on the test set are logged for every iteration and summed up over the different folds. The number of iterations that has the lowest sum of errors is used to train the LogitBoost algorithm on all the data. This gives the logistic regression model at the root of the tree.
- A split for the data at the root is constructed using the C4.5 splitting criterion [12]. Both binary splits on numerical attributes and multiway splits on nominal attributes are considered. Tree growing continues by sorting the

```
LMT(examples){
    root = new Node()
    alpha = getCARTAlpha(examples)
    root.buildTree(examples, null)
    root.CARTprune(alpha)
}

buildTree(examples, initialLinearModels) {
    numIterations = crossValidateIterations(examples, initialLinearModels)
    initLogitBoost(initialLinearModels)
    linearModels = copyOf(initialLinearModels)
    for i = 1...numIterations
        logitBoostIteration(linearModels,examples)
    split = findSplit(examples)
    localExamples = split.splitExamples(examples)
    sons = new Nodes[split.numSubsets()]
    for s = 1...sons.length
        sons.buildTree(localExamples[s], nodeModels)
}

crossValidateIterations(examples,initialLinearModels) {
    for fold = 1...5
        initLogitBoost(initialLinearModels)
        //split into training/test set
        train = trainCV(fold)
        test = testCV(fold)
        linearModels = copyOf(initialLinearModels)
        for i = 1...200
            logitBoostIteration(linearModels,train)
            logErrors[i] += error(test)
    numIterations = findBestIteration(logErrors)
    return numIterations
}
```

Fig. 2. Pseudocode for the LMT algorithm.

appropriate subsets of data to those nodes and building the logistic models of the child nodes in the following way: the LogitBoost algorithm is run on the subset associated with the child node, but starting with the committee $F_j(x)$, weights w_{ij} and probability estimates p_{ij} of the last iteration performed at the parent node (it is 'resumed' at step 2.a in Figure 1). Again, the optimum number of iterations to perform (the number of f_{jm} to add to F_j) is determined by five fold cross validation.

– Splitting continues in this fashion as long as more than 15 instances are at a node and a useful split can be found by the C4.5 splitting routine.
– The tree is pruned using the CART pruning algorithm as outlined in [2].

Figure 2 gives the pseudocode for this algorithm, which we call *LMT*. The method `LMT` constructs the tree given the training data `examples`. It first calls `getCARTAlpha` to cross-validate the 'cost-complexity-parameter' for the CART pruning scheme implemented in `CARTPrune`. The method `buildTree` grows the logistic model tree by recursively splitting the instance space. The argument `initialLinearModels` contains the simple linear regression functions already fit by LogitBoost at higher levels of the tree. The method `initLogitBoost` initializes the probabilities/weights for the LogitBoost algorithm as if it had already fitted the regression functions `initialLinearModels` (resuming LogitBoost at step 2.a). The method `crossValidateIterations` determines the number of LogitBoost iterations to perform, and `logitBoostIteration` performs a single iteration of the LogitBoost algorithm (step 2), updating the probabilities/weights and adding a regression function to `linearModels`.

Handling of Missing Values and Nominal Attributes To deal with missing values we calculate the mean (for numeric attributes) or mode (for categorical ones) based on all the training data and use these to replace them. The same means and modes are used to fill in missing values when classifying new instances.

When considering splits in the tree, multi-valued nominal attributes are handled in the usual way. However, regression functions can only be fit to numeric attributes. Therefore, they are fit to local copies of the training data where nominal attributes with k values have been converted into k binary indicator attributes.

Computational Complexity The asymptotic complexity for building a logistic regression model is $O(n \cdot v^2 \cdot c)$ if we assume that the number of LogitBoost iterations is linear in the number of attributes present in the data[2] (n denotes the number of training examples, v the number of attributes, and c the number of classes). The complexity of building a logistic model tree is $O(n \cdot v^2 \cdot d \cdot c + k^2)$, where d is the depth and k the number of nodes of the initial unpruned tree. The first part of the sum derives from building the logistic regression models, the second one from the CART pruning scheme. In our experiments, the time for building the logistic regression models accounted for most of the overall runtime. Compared to simple tree induction, the asymptotic complexity of LMT is only worse by a factor of v. However, the nested cross-validations (one to prune the tree, one to determine the optimum number of LogitBoost operations) constitute a large (albeit constant) multiplying factor.

In the algorithm outlined above, the optimum number of iterations is determined by a five fold cross-validation for every node. This is the most computationally expensive part of the algorithm. We use two heuristics to reduce the runtime:

- In order to avoid an internal cross-validation at every node, we determine the optimum number of iterations by performing *one* cross-validation in the beginning of the algorithm and then using that number *everywhere* in the tree. This approach works surprisingly well: it never produced results that were significantly worse than those of the original algorithm. This indicates that the best number of iterations for LogitBoost does depend on the dataset — just choosing a fixed number of iterations for all of the datasets lead to significantly worse results — but not so much on different subsets of a particular dataset (as encountered in lower levels in the tree).
- When performing the initial cross-validation, we have to select the number of iterations that gives the lowest error on the test set. Typically, the error will

[2] Note that in our implementation it is actually bounded by a constant (500 for standalone logistic regression and 200 at the nodes of the logistic model tree)

first decrease and later increase again because the model overfits the data. This allows the number of iterations to be chosen greedily by monitoring the error while performing iterations and stopping if the error starts to increase again. Because the error curve exhibits some spikes and irregularities, we keep track of the current minimum and stop if it has not changed for 25 iterations. Using this heuristic does not change the behavior of the algorithm significantly.

We included both heuristics in the final version of our algorithm, and all results shown for LMT refer to this final version.

3 Experiments

In order to evaluate the performance of our method and compare it against other state-of-the-art learning schemes, we applied it to several real-world problems. More specifically, we seek to answer the following questions in our experimental study:

1. How does LMT compare to the two algorithms that form its basis, i.e., logistic regression and C4.5? Ideally, we would never expect worse performance than either of these algorithms.
2. How does LMT compare to methods that build multiple trees? We include results for boosted C4.5 trees (using the AdaBoostM1 algorithm [5] and 100 boosting iterations), where the final model is a 'voting committee' of trees and for the M5' algorithm, which builds one tree per class when applied to classification problems.
3. How big are the trees constructed by LMT? We expect them to be much smaller than simple classification trees because the leaves contain more information. We also expect the trees to be pruned back to the root if a linear logistic model is the best solution for the dataset.

We will also give results for another recently developed algorithm for inducing logistic model trees called 'PLUS' (see Section 4 for a short discussion of the PLUS algorithm).

3.1 Datasets and Methodology

For our experiments we used 32 benchmark datasets from the UCI repository [1], given in the first column of Table 1. Their size ranges from under hundred to a few thousand instances. They contain varying numbers of numeric and nominal attributes and some contain missing values. For more information about the datasets, see for example [4].

For every dataset and algorithm we performed ten runs of ten fold stratified cross-validation (using the same splits into training/test set for every method). This gives a hundred data points for each algorithm and dataset, from which we calculated the average accuracy (percentage of correctly classified instances) and

Table 1. Average classification accuracy and standard deviation.

Data Set	LMT	C4.5	SimpleLogistic	M5'	PLUS	AdaBoost.M1
anneal	99.5±0.8	98.6±1.0 •	99.5±0.8	98.6±1.1	99.4±0.8 (c)	99.6±0.7
audiology	84.0±7.8	77.3±7.5 •	83.7±7.8	76.8±8.6 •	80.6±8.3 (c)	84.7±7.6
australian	85.0±4.1	85.6±4.0	85.2±4.1	85.4±3.9	85.2±3.9 (m)	86.4±4.0
autos	75.8±9.7	81.8±8.8	75.1±8.9	76.0±10.0	76.6±8.7 (c)	86.8±6.8 ◦
balance-scale	90.0±2.5	77.8±3.4 •	88.6±3.0	87.8±2.2 •	89.7±2.8 (m)	76.1±4.1 •
breast-cancer	75.6±5.4	74.3±6.1	75.6±5.5	70.4±6.8 •	71.5±5.7 (c) •	66.2±8.1 •
breast-w	96.3±2.1	95.0±2.7	96.2±2.3	95.9±2.2	96.4±2.2 (c)	96.7±2.2
german	75.3±3.7	71.3±3.2 •	75.2±3.7	75.0±3.3	73.3±3.5 (m)	74.5±3.3
glass	69.7±9.5	67.6±9.3	65.4±8.7	71.3±9.1	69.3±9.7 (c)	78.8±7.8 ◦
glass (G2)	76.5±8.9	78.2±8.5	76.9±8.8	81.1±8.7	83.2±11.1(c)	88.7±6.4 ◦
heart-c	82.7±7.4	76.9±6.6 •	83.1±7.4	82.1±6.7	78.2±7.4 (s)	80.0±6.5
heart-h	84.2±6.3	80.2±8.0	84.2±6.3	82.4±6.4	79.8±7.8 (c)	78.3±7.1 •
heart-statlog	83.6±6.6	78.1±7.4 •	83.7±6.5	82.1±6.8	83.7±6.4 (m)	80.4±7.1
hepatitis	83.7±8.1	79.2±9.6	84.1±8.1	82.4±8.8	83.3±7.8 (m)	84.9±7.8
horse-colic	83.7±6.3	85.2±5.9	82.2±6.0	83.2±5.4	84.0±5.8 (c)	81.7±5.8
hypothyroid	99.6±0.4	99.5±0.4	96.8±0.7 •	99.4±0.4	99.1±0.4 (c) •	99.7±0.3
ionosphere	92.7±4.3	89.7±4.4 •	88.1±5.3 •	89.9±4.2	89.5±5.2 (c)	94.0±3.8
iris	96.2±5.0	94.7±5.3	96.3±4.9	94.9±5.6	94.3±5.4 (c)	94.5±5.0
kr-vs-kp	99.7±0.3	99.4±0.4	97.4±0.8 •	99.2±0.5 •	99.5±0.4 (c)	99.6±0.3
labor	91.5±10.9	78.6±16.6•	91.9±10.4	85.1±16.3	89.9±11.5(c)	88.9±14.1
lymphography	84.7±9.6	75.8±11.0•	84.5±9.3	80.4±9.3	78.4±10.2(c)	84.7±8.4
pima-indians	77.1±4.4	74.5±5.3	77.1±4.5	76.6±4.7	77.2±4.3 (m)	73.9±4.8 •
primary-tumor	46.7±6.2	41.4±6.9 •	46.7±6.2	45.3±6.2	40.7±6.1 (c) •	41.7±6.5 •
segment	97.1±1.2	96.8±1.3	95.4±1.5 •	97.4±1.0	96.8±1.1 (c)	98.6±0.7 ◦
sick	98.9±0.6	98.7±0.6	96.7±0.7 •	98.4±0.6 •	98.6±0.6 (c)	99.0±0.5
sonar	76.4±9.4	73.6±9.3	75.1±8.9	78.4±8.8	71.6±8.0 (c)	85.1±7.8 ◦
soybean	93.6±2.5	91.8±3.2	93.5±2.7	92.9±2.6	93.6±2.7 (c)	93.3±2.8
vehicle	82.4±3.3	72.3±4.3 •	80.4±3.4	78.7±4.4 •	79.8±4.0 (m)	77.9±3.6 •
vote	95.7±2.8	96.6±2.6	95.7±2.7	95.6±2.8	95.3±2.8 (c)	95.2±3.3
vowel	94.1±2.5	80.2±4.4 •	84.2±3.7 •	80.9±4.7 •	83.0±3.7 (c) •	96.8±1.9 ◦
waveform-noise	87.0±1.6	75.3±1.9 •	86.9±1.6	82.5±1.6 •	86.7±1.5 (m)	85.0±1.6 •
zoo	95.0±6.6	92.6±7.3	94.8±6.7	94.5±6.4	94.5±6.8 (c)	96.3±6.1

◦, • statistically significant win or loss

standard deviation. To correct for the dependencies in the estimates we used the corrected resampled t-test [9] instead of the standard t-test on these data points to identify significant wins/losses of our method against the other methods at a 5% significance level.

Table 1 gives the average classification accuracy for every method and dataset, and indicates significant wins/losses compared to LMT. Table 2 gives the number of datasets on which a method (column) significantly outperforms another method (row). Apart from PLUS all algorithms are implemented in Weka 3.3.6 (including LMT and SimpleLogistic)[3]. Note that PLUS has three different modes of operation: one to build a simple classification tree, and two modes that build logistic model trees using simple/multiple logistic regression models. For all datasets, we ran PLUS in all three modes and selected the best result, indicated by (c),(s) or (m) in Table 1.

3.2 Discussion of Results

To answer our first question, we observe from Table 1 that the LMT algorithm indeed reaches at least accuracy levels comparable to both SimpleLogistic and

[3] Weka is available from www.cs.waikato.ac.nz/~ml

Table 2. Number of datasets where algorithm in column significantly outperforms algorithm in row

	LMT	C4.5	SimpleLogistic	M5'	PLUS	AdaBoost.M1
LMT	-	0	0	0	0	6
C4.5	13	-	6	5	5	13
SimpleLogistic	6	3	-	4	4	10
M5'	8	0	1	-	1	12
PLUS	4	1	1	2	-	0
AdaBoost.M1	7	1	3	1	0	-

Table 3. Tree size

Data Set	LMT	C4.5	PLUS	Data Set	LMT	C4.5	PLUS
anneal	1.8	38.0 ○	15.8 ○	ionosphere	4.6	13.9 ○	13.5 ○
audiology	1.0	29.9 ○	47.6 ○	iris	1.1	4.6 ○	6.1 ○
australian	2.5	22.5 ○	2.0	kr-vs-kp	8.0	29.3 ○	42.6 ○
autos	3.0	44.8 ○	42.2 ○	labor	1.0	4.2 ○	5.1 ○
balance-scale	5.3	41.6 ○	1.9 ●	lymphography	1.2	17.3 ○	14.9 ○
breast-cancer	1.1	9.8 ○	9.1 ○	pima-indians	1.0	22.2 ○	1.2
breast-w	1.4	12.2 ○	1.2	primary-tumor	1.0	43.8 ○	26.7 ○
german	1.0	90.2 ○	4.3 ○	segment	12.0	41.2 ○	65.9 ○
glass	7.0	23.6 ○	25.2 ○	sick	14.1	27.4 ○	30.0 ○
glass (G2)	4.6	12.5 ○	15.5 ○	sonar	2.7	14.5 ○	13.2 ○
heart-c	1.0	25.7 ○	6.2 ○	soybean	3.7	61.1 ○	42.4 ○
heart-h	1.0	6.3 ○	5.1	vehicle	3.5	69.5 ○	1.0 ●
heart-statlog	1.0	17.8 ○	1.0	vote	1.1	5.8 ○	5.8 ○
hepatitis	1.1	9.3 ○	1.5	vowel	5.2	123.3 ○	156.9 ○
horse-colic	3.7	5.9	6.6	waveform-noise	1.0	296.5 ○	1.0
hypothyroid	5.6	14.4 ○	12.9 ○	zoo	1.0	8.4 ○	9.5 ○

○, ● statistically significant increase or decrease

C4.5, it is never significantly less accurate. It outperforms SimpleLogistic on six and C4.5 on 13 datasets, and both methods simultaneously on two datasets. SimpleLogistic performs surprisingly well on most datasets, especially the smaller ones. However, on some larger datasets ('kr-vs-kp', 'sick', 'hypothyroid') its performance is a lot worse than that of any other method. Linear models are probably too restricted to achieve good performance on these datasets.

With regard to the second question, we can say that LMT achieves similar results as boosted C4.5 (although with strengths/weaknesses on different datasets). Comparing LMT with M5', we find better results for LMT on almost all datasets. Note that LMT also outperforms PLUS, even though the selection of the best result from the three modes for PLUS introduces an optimistic bias.

To answer our third question, Table 3 gives the observed average tree sizes (measured in number of leaves) for LMT, C4.5 and PLUS. It shows that the trees built by the LMT algorithm are always smaller than those built by C4.5 and mostly smaller than those generated by PLUS. For many datasets the average tree size for LMT is very close to one, which essentially means that the algorithm constructs a simple logistic model. To account for small random fluctuations, we will say the tree is pruned back to the root if the average tree size is less than 1.5. This is the case for exactly half of the 32 datasets, and consequently the results for LMT on these datasets are almost identical to those of SimpleLogistic.

It can be seen from Table 1 that on all datasets (with the exception of 'vote') where the tree is pruned back to the root, the result for LMT is better than that for C4.5, so it is reasonable to assume that for these datasets using a simple logistic regression model is indeed better than building a tree structure. Looking at the sixteen datasets where the logistic model tree is not pruned back to the root, we observe that on 13 of them LMT is more accurate than SimpleLogistic. This indicates a tree is only built if this leads to better performance than a single logistic model. From these two observations we can conclude that our method reliably makes the right choice between a simple logistic model and a more elaborate tree structure.

We conclude that the LMT algorithm achieves better results than C4.5, SimpleLogistic, M5' and PLUS, and results that are competitive with boosted C4.5. Considering that a single logistic model tree is easier to interpret than a boosted committee of C4.5 trees we think that LMT is an interesting alternative to boosting trees. Of course, one could also boost logistic model trees — but because building them takes longer than building simple trees this would be computationally expensive.

4 Related Work

As mentioned above, model trees form the basis for the ideas presented here, but there has also been some interest in combining regression and tree induction into 'tree structured regression' in the statistics community. For example, Chaudhuri et al. [3] investigate a general framework for combining tree induction with node-based regression models that are fit by maximum likelihood. Special cases include poisson regression trees (for integer-valued class variables) and logistic regression trees (for binary class variables only). Chaudhuri et al. apply their logistic regression tree implementation to one real-world dataset, but it is not their focus to compare it to other state-of-the-art learning schemes.

More recently, Lim presents an implementation of logistic regression trees called 'PLUS' [8]. There are some differences between the PLUS system and our method: first, PLUS does not consider nominal attributes when building the logistic regression models, i.e. it reverts to building a standard decision tree if the data does not contain numeric attributes. Second, PLUS uses a different method to construct the logistic regression models at the nodes. In PLUS, every logistic model is trained from scratch on the data at a node, whereas in our method the final logistic model consists of a committee of linear models trained on nested subsets of the data, thus naturally incorporating a form of 'smoothing'. Furthermore, our approach automatically selects the best attributes to include in a logistic model, while PLUS always uses all or just one attribute (a choice that has to be made at the command line by the user).

5 Conclusions

This paper introduces a new method for learning logistic model trees that builds on earlier work on model trees. This method, called LMT, employs an efficient and flexible approach for building logistic models and uses the well-known CART algorithm for pruning. Our experiments show that it is often more accurate than C4.5 decision trees and standalone logistic regression on real-world datasets, and, more surprisingly, competitive with boosted C4.5 trees. Like other tree induction methods, it does not require any tuning of parameters.

LMT produces a single tree containing binary splits on numeric attributes, multiway splits on nominal ones, and logistic regression models at the leaves, and the algorithm ensures that only relevant attributes are included in the latter. The result is not quite as easy to interpret as a standard decision tree, but much more intelligible than a committee of multiple trees or more opaque classifiers like kernel-based estimators.

Acknowledgments

Many thanks to Luc de Raedt and Geoff Holmes for enabling Niels to work on his MSc at Waikato. Eibe Frank was supported by Marsden Grant 01-UOW-019.

References

1. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. [www.ics.uci.edu/~mlearn/MLRepository.html].
2. L. Breiman, H. Friedman, J. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
3. P. Chaudhuri, W.-D. Lo, W.-Y. Loh, and C.-C. Yang. Generalized regression trees. *Statistica Sinica*, 5:641–666, 1995.
4. Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.
5. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc. Int. Conf. on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996.
6. Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistic*, 38(2):337–374, 2000.
7. Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.
8. T.-S. Lim. *Polytomous Logistic Regression Trees*. PhD thesis, Department of Statistics, University of Wisconsin, 2000.
9. C. Nadeau and Yoshua Bengio. Inference for the generalization error. In *Advances in Neural Information Processing Systems 12*, pages 307–313. MIT Press, 1999.
10. C. Perlich and F. Provost. Tree induction vs logistic regression. In *Beyond Classification and Regression (NIPS 2002 Workshop)*, 2002.
11. J. R. Quinlan. Learning with Continuous Classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, 1992.
12. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
13. Y. Wang and I. Witten. Inducing model trees for continuous classes. In *Proc of Poster Papers, European Conf. on Machine Learning*, 1997.