Speeding up Logistic Model Tree Induction

Marc Sumner[1,2], Eibe Frank[2], and Mark Hall[2]

Institute for Computer Science
University of Freiburg
Freiburg, Germany
sumner@informatik.uni-freiburg.de

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

Abstract. Logistic Model Trees have been shown to be very accurate and compact classifiers [8]. Their greatest disadvantage is the computational complexity of inducing the logistic regression models in the tree. We address this issue by using the AIC criterion [1] instead of cross-validation to prevent overfitting these models. In addition, a weight trimming heuristic is used which produces a significant speedup. We compare the training time and accuracy of the new induction process with the original one on various datasets and show that the training time often decreases while the classification accuracy diminishes only slightly.

1 Introduction

Logistic Model Trees (LMTs) are born out of the idea of combining two complementary classification schemes: linear logistic regression and tree induction. It has been shown that LMTs perform competitively with other state-of-the-art classifiers such as boosted decision trees while being easier to interpret [8]. However, the main drawback of LMTs is the time needed to build them. This is due mostly to the cost of building the logistic regression models at the nodes. The LogitBoost algorithm [6] is repeatedly called for a fixed number of iterations, determined by a five fold cross-validation. In this paper we investigate whether cross-validation can be replaced by the AIC criterion without loss of accuracy. We also investigate a weight trimming heuristic and show that it improves training time as well.

The rest of this paper is organized as follows. In Section 2 we give a brief overview of the original LMT induction algorithm. Section 3 describes the modifications made to various parts of the algorithm. In Section 4, we evaluate the modified algorithm and discuss the results, and in Section 5 we draw some conclusions.

2 Logistic Model Tree Induction

The original LMT induction algorithm can be found in [8]. We give a brief overview of the process here, focusing on the aspects where we have made improvements. We begin this section with an overview of the underlying foundations and conclude it with a synopsis of the original LMT induction algorithm.

LogitBoost (J classes)

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 $\underset{j}{\text{argmax}}\ \ F_j(x)$

Fig. 1. LogitBoost algorithm.

2.1 Logistic Regression

Linear logistic regression models the posterior class probabilities $Pr(G = j|X = x)$ for the J classes via functions linear in x and ensures that they sum to one and remain in $[0, 1]$. The model is of the form

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

where $F_j(x) = \beta_j^T \cdot x$. Numeric optimization algorithms that approach the maximum likelihood solution iteratively are used to find the estimates for β_j.

One such iterative method is the LogitBoost algorithm [6], shown in Figure 1. In each iteration, it fits a least-squares regressor to a weighted version of the input data with a transformed target variable. Here, y^*_{ij} are the binary pseudo-response variables which indicate group membership of an observation like this

$$y^*_{ij} = \begin{cases} 1 & \text{if } y_i = j, \\ 0 & \text{if } y_i \neq j \end{cases}, \tag{2}$$

where y_i is the observed class for instance x_i.

If we constrain f_{mj} to be linear in x, then we achieve linear logistic regression if the algorithm is run until convergence. If we further constrain f_{mj} to be a linear function of only the attribute that results in the lowest squared error, then we arrive at an algorithm that performs automatic attribute selection. By using cross-validation to determine the best number of LogitBoost iterations M, only those attributes are included that improve the performance on unseen instances. This method is called "SimpleLogistic" [8].

2.2 Logistic Model Trees

For the details of the LMT induction algorithm, the reader should consult [8]. Here is a brief summary of the algorithm:

- First, LogitBoost is run on all the data to build a logistic regression model for the root node. The number of iterations to use is determined by a five fold cross-validation. In each fold, LogitBoost is run on the training set up to a maximum number of iterations (200). The number of iterations that produces the lowest sum of errors on the test set over all five folds is used in LogitBoost on all the data to produce the model for the root node and is also used to build logistic regression models at all nodes in the tree.
- The data is split using the C4.5 splitting criterion [10]. Logistic regression models are then built at the child nodes on the corresponding subsets of the data using LogitBoost. However, the algorithm starts with the committee $F_j(x)$, weights w_{ij} and probability estimates p_{ij} inherited from the parent.
- As long as at least 15 instances are present at a node and a useful split is found (as defined in the C4.5 splitting scheme), then splitting and model building is continued in the same fashion.
- The CART cross-validation-based pruning algorithm is applied to the tree [3].

3 Our Modifications

In the following we discuss the additions and modifications we made to the algorithms that make up LMT induction.

Weight Trimming The idea of weight trimming in association with LogitBoost is mentioned in [6]. It is a very simple, yet effective method for reducing computation of boosted models. In our case, only training instances carrying $100 \cdot (1 - \beta)\%$ of the total weight mass are used for building the simple linear regression model, where $\beta \in [0, 1]$. Typically $\beta \in [0.01, 0.1]$. We used $\beta = 0.1$. In later iterations more of the training instances become correctly classified with a higher confidence; hence, more of them receive a lower weight and the number of instances carrying $100 \cdot (1 - \beta)\%$ of the weight becomes smaller.

The computation needed to build the simple linear regression model thus decreases as the iterations proceed. Of course, the computational complexity is still $O(n \cdot a)$; however, n is reduced by a potentially large constant factor and in practice, a reduction in computation time is achieved without sacrificing predictive accuracy (see Section 4).

Automatic Iteration Termination In the original induction algorithm for LMTs the number of LogitBoost iterations for all nodes is determined by a five fold cross-validation at the root and used for all nodes. This, of course, is a time consuming process when the number of attributes and/or instances in the

training data is large. The aim is to terminate the LogitBoost iterations when the model performs well on the training data, yet does not overfit it.

A common alternative to cross-validation for model selection is the use of an in-sample estimate of the generalization error, such as Akaike's Information Criterion (AIC) [1]. We investigated its usefulness in selecting the optimum number of LogitBoost iterations and found it to be a viable alternative to cross-validation in terms of classification accuracy and far superior in training time.

AIC provides an estimate of the generalization error when a negative log-likelihood loss function is used. Let this function be denoted as $loglik$ and let N be the number of training instances. Then AIC is defined as

$$AIC = -\frac{2}{N} loglik + 2\frac{d}{N}, \tag{3}$$

where d is the number of inputs or basis functions. If the basis functions are chosen adaptively, then d must be adjusted upwards [7]. In this case, d denotes the effective number of parameters, or degrees of freedom, of the model. It is not clear what value to use for d in SimpleLogistic. In [4] the effective number of parameters is computed for boosting methods using the "boosting operator" which is a linear combination of the hat matrices of the basis functions. Our implementation of this method led to a large computational overhead which actually made it slower than cross-validation.

Intuitively, the logistic regression model built via SimpleLogistic should be penalized (i.e. d should increase) each time a new attribute is introduced to the model. We could just use the number of attributes used in a committee function F_j for any class j. However, when an iteration introduces no new attributes, the model is not penalized, although it has become more complex. At the other end of the spectrum, we could penalize each iteration equally which leads to another estimate, $d = i$, where i is the iteration number. As i increases, the first term in Equation 3 decreases (because LogitBoost performs quasi-Newton steps approaching the maximum log-likelihood [6]) and the second term (the penalty term) always increases. Empirically, this was found to be a good estimate (see Section 4).

The optimal number of iterations is the i which minimizes AIC. So, in order to determine the optimal number i^*, LogitBoost must be run up to a maximum number of iterations (in LMT induction, this is 200), and then run again for i^* iterations. Just as with the cross-validation process in the original LMT induction algorithm, we performed the AIC procedure at the root node of the logistic model tree and used i^* throughout the tree. Also, if no minimum was found for 50 iterations, then no more iterations were performed and the iteration that produces the minimum AIC was used as i^*. This is analogous to the heuristic employed for the cross-validation method in [8]. However, we found that we can modify this process to achieve a speed-up without loss of accuracy.

We tested the AIC-based model selection method on 13 UCI datasets [2] (see Section 4) and observed that, for every dataset, AIC only had one global minimum over all iterations. Hence we can attempt to speed up the aforementioned model selection method. We stop searching as soon as AIC no longer decreases.

Determining the optimal number of iterations in this fashion will be called the *First AIC Minimum* (FAM) method in the remainder of this paper.

FAM allows us to efficiently compute (an approximation to) the optimal number of iterations at *each* node in the tree. This is advantageous in two ways:

- Instead of iterating up to a maximum number of iterations five times (in the cross-validation case) or once (in the AIC case) and then building the model using the optimal number of iterations i^*, LogitBoost can be stopped immediately when i^* is found.
- As we move down the tree, the sets of training instances become smaller (they are subsets of the instances observed at the parent node). AIC is inversely related to the number of instances N, so as N becomes smaller, not only will simpler models be selected, but training time decreases as fewer iterations are needed. It is also much more intuitive that a different number of iterations is appropriate for different datasets occuring in a tree.

In addition, we no longer need to set a maximum number of iterations to be performed. As mentioned in [8] the limit of 200 was appropriate for the observed datasets; however it is not clear whether this is enough for all datasets.

4 Experiments

This section evaluates the modified LMT induction algorithm and its base learner SimpleLogistic by comparing them against the original implementation. For the evaluation we measure training times and classification accuracy. All experiments are ten runs of a ten-fold stratified cross-validation. The mean and standard deviation over the 100 results are shown in all tables presented here. In all experimental results, a corrected resampled t-test was used [9] instead of the standard t-test to test the difference in training times and accuracy, at a 5% significance level. This corrects for the dependencies in the estimates of the different data points, and is thus less prone to false-positives.

We used 13 datasets with a nominal class variable available from the UCI repository [2]. We used only datasets that have approximately 1000 training instances or more (vowel was the exception with 990 instances). Both numeric and nominal attributes appear in all of the UCI datasets.

All experiments were run using version 3.4.4 of the Weka machine learning workbench [11]. Almost all experiments were run on a pool of identical machines with an Intel Pentium 4 processor with 2.8GHz and 512MB ram[1], Linux kernel 2.4.28 and Java 1.5.0-b64.

4.1 SimpleLogistic

We first evaluated the SimpleLogistic learner for logistic regression, measuring the effects of weight trimming and investigating the use of FAM for determining the optimum number of LogitBoost iterations.

[1] All LMT algorithms required more memory for the adult dataset and were thus run on an Intel Pentium 4 processor with 3.0GHz and 1GB ram.

Dataset	Training Time SimpleLog.	SimpleLog. (WT)		Accuracy SimpleLog.	SimpleLog. (WT)
vowel	77.94±23.59	39.67±12.72	•	81.98±4.10	82.07±3.82
german-credit	7.97±1.94	6.79±1.55		75.37±3.53	75.35±3.48
segment	50.55±14.82	20.02±5.61	•	95.10±1.46	86.71±25.67
splice	253.96±38.83	79.02±9.55	•	95.86±1.17	95.87±1.09
kr-vs-kp	57.28±15.09	25.98±8.35	•	97.06±0.98	97.07±0.92
hypothyroid	104.76±27.17	47.88±10.72	•	96.61±0.71	96.55±0.72
sick	25.40±6.10	12.09±3.39	•	96.68±0.71	96.63±0.70
spambase	119.28±18.73	43.19±4.38	•	92.75±1.12	92.40±1.24
waveform	65.53±9.31	25.42±3.77	•	86.96±1.58	86.90±1.55
optdigits	659.33±123.68	111.32±21.35	•	97.12±0.67	97.17±0.67
pendigits	489.51±148.34	257.86±84.23	•	95.44±0.62	95.51±0.61
nursery	266.51±25.56	119.19±11.36	•	92.61±0.68	92.60±0.77
adult	2953.77±849.82	1866.15±344.05	•	85.61±0.38	85.56±0.38

• statistically significant improvement

Table 1. Training time and accuracy for SimpleLogistic and SimpleLogistic using weight trimming

Dataset	Training Time SimpleLog. (CV)	SimpleLog. (FAM)		Accuracy SimpleLog. (CV)	SimpleLog. (FAM)	
vowel	77.94±23.59	6.87±0.31	•	81.98±4.10	80.85±3.69	
german-credit	7.97±1.94	0.59±0.05	•	75.37±3.53	75.34±3.70	
segment	50.55±14.82	3.42±0.45	•	95.10±1.46	94.67±1.66	
splice	253.96±38.83	77.48±3.69	•	95.86±1.17	95.87±1.06	
kr-vs-kp	57.28±15.09	6.69±0.37	•	97.06±0.98	96.38±1.14	○
hypothyroid	104.76±27.17	8.89±1.16	•	96.61±0.71	95.89±0.65	○
sick	25.40±6.10	1.57±0.14	•	96.68±0.71	96.50±0.76	
spambase	119.28±18.73	15.74±1.28	•	92.75±1.12	92.69±1.19	
waveform	65.53±9.31	7.75±0.39	•	86.96±1.58	86.84±1.59	
optdigits	659.33±123.68	135.61±26.47	•	97.12±0.67	97.12±0.66	
pendigits	489.51±148.34	59.43±1.58	•	95.44±0.62	95.45±0.62	
nursery	266.51±25.56	49.36±1.42	•	92.61±0.68	92.58±0.68	
adult	2953.77±849.82	381.92±10.13	•	85.61±0.38	85.59±0.38	

•, ○ statistically significant improvement or degradation

Table 2. Training time and accuracy for SimpleLogistic using cross-validation and FAM

Weight Trimming in SimpleLogistic From Table 1 it can be seen that weight trimming consistently reduces the training time of SimpleLogistic on all datasets (with the exception of german-credit) while not affecting classification accuracy. The greatest effect of weight trimming was seen on the optdigits dataset. Here, a speedup of almost 6 was recorded. Overall, weight trimming is a safe heuristic (i.e. it does not affect accuracy) that can result in significant speedups.

FAM in SimpleLogistic This section deals with the evaluation of SimpleLogistic implemented with FAM, introduced in Section 3. SimpleLogistic with FAM is compared with the original cross-validation-based approach.

Table 2 shows the training time and classification accuracy for both algorithms on the 13 UCI datasets. FAM consistently produced a significant speedup on all datasets. This ranged from 3.3 on the splice dataset to 14.8 on the segment dataset. Looking at the classification accuracy, we can see that FAM performs significantly worse on two datasets (kr-vs-kp and hypothyroid), but the degradation is within reasonable bounds.

Dataset	Training Time			Accuracy	
	LMT	LMT (FAM+WT)		LMT	LMT (FAM+WT)
vowel	408.11±80.95	15.86±0.84	•	94.06±2.40	93.56±2.94
german-credit	32.74±10.87	3.25±0.16	•	75.50±3.65	71.83±3.40 ○
segment	143.75±52.64	10.58±1.77	•	97.06±1.31	97.06±1.25
splice	785.51±202.14	71.55±1.42	•	95.89±1.14	95.19±1.19 ○
kr-vs-kp	250.79±64.58	12.17±0.36	•	99.64±0.33	99.57±0.37
hypothyroid	405.73±94.04	7.39±0.64	•	99.54±0.36	99.61±0.30
sick	139.31±50.79	6.83±0.73	•	98.95±0.58	98.93±0.62
spambase	746.71±123.57	54.93±1.65	•	93.56±1.14	93.58±1.13
waveform	175.53±63.26	43.67±0.80	•	86.86±1.60	86.49±1.52
optdigits	3162.37±781.49	133.15±7.08	•	97.38±0.57	97.36±0.64
pendigits	3535.06±765.34	185.15±4.96	•	98.58±0.33	98.73±0.33
nursery	634.96±85.82	72.44±7.08	•	98.95±0.34	98.64±0.32 ○
adult	26935.85±9112.20	1429.93±54.76	•	85.58±0.42	85.43±0.37

•, ○ statistically significant improvement or degradation

Table 3. Training time and accuracy for LMT and LMT using FAM and weight trimming

4.2 Logistic Model Trees

We can now observe the impact of our modifications on the LMT induction algorithm. Table 3 compares the original LMT version with the modified LMT induction algorithm (using FAM and weight trimming). As expected, our modifications result in the algorithm being much faster on all of the datasets. The greatest speedup recorded was 55 on the hypothyroid dataset. Most common was a speedup between 10 and 25. Only on the german-credit, waveform, and nursery datasets was the speedup around 10 or less (10.1, 4.0, and 8.8, respectively).

On german-credit, splice and nursery the modified version's classification performance was significantly worse than that of the original version, although only on german-credit was the performance worse by more than one percent. Otherwise, the modified version performed competitively with the original version.

As a closing note to our experiments, we would like to compare the modified version of logistic model trees to boosted C4.5 decision trees. For the comparison we chose AdaBoost [5] using 100 iterations and the LMT version using FAM and weight trimming. The results can be seen in Table 4. 100 iterations are too many for a few of the datasets, but moving from 10 to 100 iterations results in an improvement in accuracy in many cases [8].

The training time of the two algorithms is fairly equal, with a slight advantage for the modified LMT algorithm. It was faster on 9 of the 13 datasets, with a speedup of usually between 1 and 2. The exceptions are the sick dataset (7.2) and the waveform dataset (10.6). AdaBoost was faster on three datasets, the greatest improvement being on the splice dataset (6.0). In terms of classification accuracy, the new LMT version exhibits results similar to those reported in [8] for the original LMT algorithm. On six datasets AdaBoost performed significantly better, while on two datasets LMT was the better classifier.

5 Conclusions

We have proposed two modifications to the SimpleLogistic algorithm employed by LMT that are designed to improve training time. The use of AIC instead

Dataset	Training Time			Accuracy		
	AdaBoost	LMT (FAM+WT)		AdaBoost	LMT (FAM+WT)	
vowel	29.32±0.38	15.86±0.84	•	96.74±1.89	93.56±2.94	○
german-credit	7.42±0.17	3.25±0.16	•	74.40±3.23	71.83±3.40	
segment	45.53±0.67	10.58±1.77	•	98.58±0.76	97.06±1.25	○
splice	11.89±5.46	71.55±1.42	○	94.94±1.24	95.19±1.19	
kr-vs-kp	21.14±6.44	12.17±0.36	•	99.60±0.31	99.57±0.37	
hypothyroid	19.07±11.46	7.39±0.64	•	99.70±0.31	99.61±0.30	
sick	49.40±2.32	6.83±0.73	•	99.06±0.45	98.93±0.62	
spambase	70.22±63.21	54.93±1.65		95.34±0.87	93.58±1.13	○
waveform	463.38±4.18	43.67±0.80	•	85.01±1.77	86.49±1.52	•
optdigits	402.52±3.06	133.15±7.08	•	98.55±0.50	97.36±0.64	○
pendigits	274.59±2.72	185.15±4.96	•	99.41±0.26	98.73±0.33	○
nursery	24.90±0.48	72.44±7.08	○	99.79±0.14	98.64±0.32	○
adult	796.01±64.89	1429.93±54.76	○	82.18±0.46	85.43±0.37	•

•, ○ statistically significant improvement or degradation

Table 4. Training time and accuracy for AdaBoost using C4.5 with 100 iterations and LMT using FAM and weight trimming

of cross-validation to determine an appropriate number of LogitBoost iterations resulted in a dramatic speedup. It resulted in a small but significant decrease in accuracy in only two cases when performing stand-alone logistic regression. The simple heuristic of weight trimming consistently improved the training time while not affecting accuracy at all.

The use of AIC and weight trimming in LMT have resulted in training times up to 55 times faster than the original LMT algorithm while, in most cases, not significantly affecting classification accuracy. These results were measured on datasets of relatively low size and dimensionality. We would expect the speedup to be even greater on larger and high-dimensional datasets.

References

1. H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second Int Symposium on Information Theory*, pages 267–281, 1973.
2. C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998. [www.ics.uci.edu/∼mlearn/MLRepository.html].
3. L. Breiman, H. Friedman, J. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.
4. Peter Bühlmann and Bin Yu. Boosting, model selection, lasso and nonnegative garrote. Technical Report 2005-127, Seminar for Statistics, ETH Zürich, 2005.
5. Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proc In. 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 Statistics*, 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. Niels Landwehr, Mark Hall, and Eibe Frank. Logistic model trees. *Machine Learning*, 59(1/2):161–205, 2005.
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. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
11. I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implemenations*. Morgan Kaufmann, San Francisco, 2000.