20

Logic Regression - Methods and Software

Ingo Ruczinski, Charles Kooperberg and Michael LeBlanc[1]

Summary

Logic Regression is an adaptive regression methodology that constructs predictors as Boolean combinations of binary covariates. This method, introduced by Ruczinski, Kooperberg, and LeBlanc [5] is particularly useful for problems where most covariates are binary, and the interactions between those predictors is of main interest. Here, we briefly review the methodology, describe the publicly available software, and give an example. The software is currently available from `http://bear.fhcrc.org/~ingor/logic`.

20.1 The Logic Regression Methodology

In most regression problems a model is developed that relates the main effects (the predictors or transformations thereof) to the response. Although interactions between predictors are sometimes considered as well, those interactions are typically kept simple (two- to three-way interactions at most). But often, especially when all predictors are binary, the interaction between many predictors is what is associated with differences in response. This issue arises, for example, in the analysis of SNP microarray data or in some data mining problems. Given a set of binary predictors X, we try to create new predictors for the response by considering combinations of those binary predictors. For example, if the response is binary as well

[1]Ingo Ruczinski is at the Department of Biostatistics, Bloomberg School of Public Health, Johns Hopkins University, Baltimore, MD 21205-2179; Charles Kooperberg and Michael LeBlanc are at the Division of Public Health Sciences, Fred Hutchinson Cancer Research Center, Seattle, WA 98109-1024 (E-mail: clk@fhcrc.org, mikel@swog.fhcrc.org). This research was supported in part by National Institutes of Health grants CA74841 and CA53996.

(which is not required in general), we attempt to find decision rules such as "if X_1, X_2, X_3 and X_4 are true", or "X_5 or X_6 but not X_7 are true", then the response is more likely to be in class 0. In other words, we try to find Boolean statements involving the binary predictors that enhance the prediction for the response. Formally: let $X_1, \dots, X_k$ be binary predictors, and let Y be a response variable. We try to fit regression models of the form $g(E[Y]) = b_0 + b_1 L_1 + \cdots + b_n L_n$, where L_j is a Boolean expression of the predictors X, such as $L_j = [(X_2 \wedge X_4^c) \vee X_7]$. The above framework includes many forms of regression, such as linear regression ($g(E[Y]) = E[Y]$) and logistic regression ($g(E[Y]) = \log(E[Y]/(1-E[Y]))$). For every model type, we define a score function that reflects the "quality" of the model under consideration. For linear regression the score could be the residual sum of squares and for logistic regression it could be the binomial deviance. We try to find the Boolean expressions in the regression model that minimize the scoring function associated with this model type, estimating the parameters b_j simultaneously with the Boolean expressions L_j. In the Logic Regression framework any type of model can be considered, as long as a scoring function can be defined. For example, we also implemented the Cox proportional hazards model, using the partial likelihood as the score.

There are some similarities between Logic Regression and some of the so called rule induction methods developed in the field of Machine Learning. Logic Regression differs from all methods that we are aware of in one or both of two important aspects: (i) the Logic Regression methodology places no restrictions on the form of the logic expressions L_j, and (ii) the Logic Regression methodology is not specifically designed for one particular problem (in machine learning often classification) but works with any scoring function. In our experience it performs better with continuous measures, such as log-likelihoods, than with discrete measures, such as misclassification. We refer to Ruczinski et al. [5] for a comparison of Logic Regression and machine learning methods.

Any Boolean statement can be represented as a binary tree (called Logic Tree), the variables being the leaves of the tree and the logic operators ($\vee, \wedge$) as the other knots (see [5] for details; Figure 20.3 later in this chapter displays Logic Trees). On the set of trees we define a move set by a collection of standard operations: alternating leaves, changing operators, splitting and deleting leaves, and growing and pruning the trees. The terminology used is similar to the terminology introduced by Breiman et al. [1]. Ruczinski et al. [5] also provide a comparison between CART and Logic models. Since the number of possible Logic Models for a given set of predictors can be very large, we rely on search algorithms to help us find the best scoring models. We implemented two algorithms: a greedy (stepwise) and a simulated annealing algorithm. While the greedy algorithm is very fast, it does not always find a good scoring model. Our preferred algorithm is the simulated annealing algorithm, which usually does find good scoring models, but is computationally more expensive.

As for many adaptive regression methodologies, the best scoring model often over-fits the data, and model selection is needed. We implemented several methods for model selection, using randomization tests and cross-validation. A detailed introduction to Logic Regression can be found in Ruczinski et al. [5]. See Kooperberg et al. [3] for an application of said methodology to single nucleotide polymorphism (SNP) data.

20.2 The Logic Regression Software

The Logic Regression program is a stand-alone program *xlogic* written in Fortran 90 that can be downloaded from `http://bear.fhcrc.org/~ingor/logic`. *xlogic* can be used to fit one logic regression model, to fit logic regression models of pre-specified sizes, to carry out cross-validation, or to do various randomization tests. Each application requires an input file, which can be edited manually or be generated from one of the online available menus. The results of *xlogic* are a number of ASCII files. These can be directly used as input to several S-Plus functions to generate graphical representations of the output.

Currently the Logic Regression methodology has scoring functions for linear regression (residual sum of squares), logistic regression (binomial deviance), classification (misclassification), and proportional hazards models (partial likelihood). A feature of the Logic Regression methodology is that it is easy to include and use ones own scoring function if that is desired. Online help is available from the website.

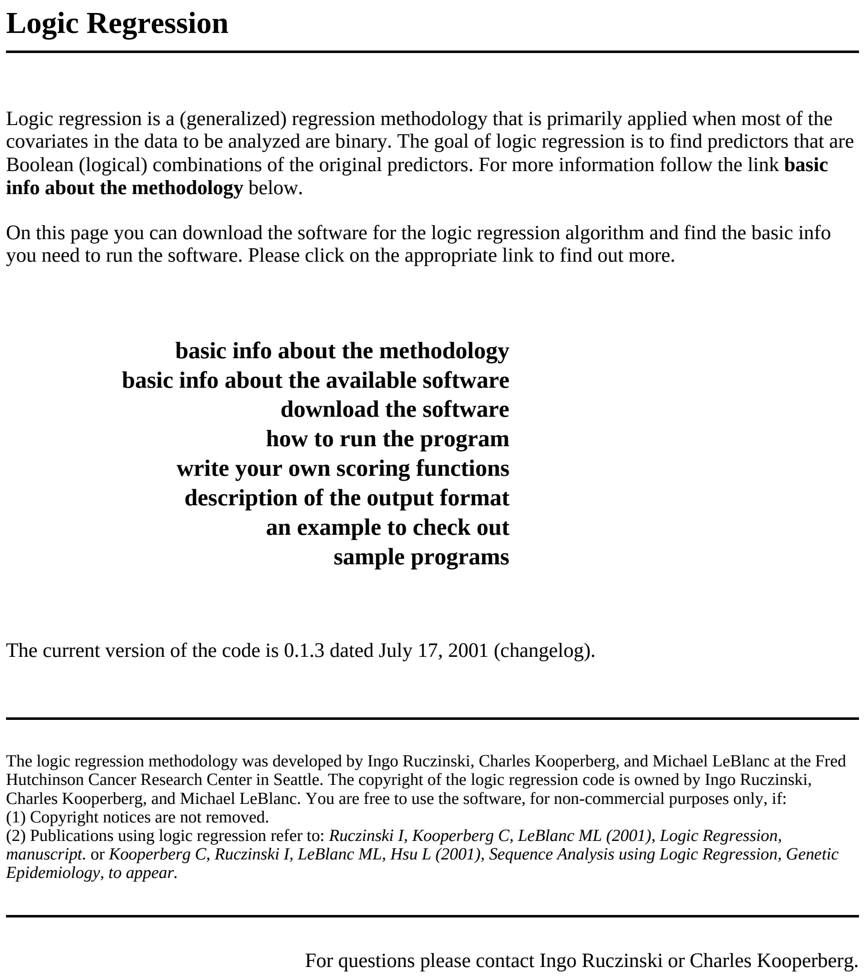
20.2.1 Running the Software

In the following sections we will focus on the current version of the program. A number of extensions of the methodology are planned for the near future. In Figure 20.2 is the online menu that one obtains after selecting **how to run the program** on the previous (main) menu. It displays the currently available features of the Logic Regression software.

There are currently five versions of the Logic Regression program, available on the web site. They are listed in Figure20.2. For each of these versions a menu is available, which guides the user through the selection of the various options. We now discuss the various versions of the program.

20.2.2 Find the best scoring model of any size

To select a good scoring Logic Regression model, we use a simulated annealing (see, for example, Otten and van Ginneken [4] and van Laarhoven and Aarts [6]) search algorithm. In general, simulated annealing operates on a state space S, which is a collection of individual states, representing



Logic Regression

Logic regression is a (generalized) regression methodology that is primarily applied when most of the covariates in the data to be analyzed are binary. The goal of logic regression is to find predictors that are Boolean (logical) combinations of the original predictors. For more information follow the link **basic info about the methodology** below.

On this page you can download the software for the logic regression algorithm and find the basic info you need to run the software. Please click on the appropriate link to find out more.

basic info about the methodology
basic info about the available software
download the software
how to run the program
write your own scoring functions
description of the output format
an example to check out
sample programs

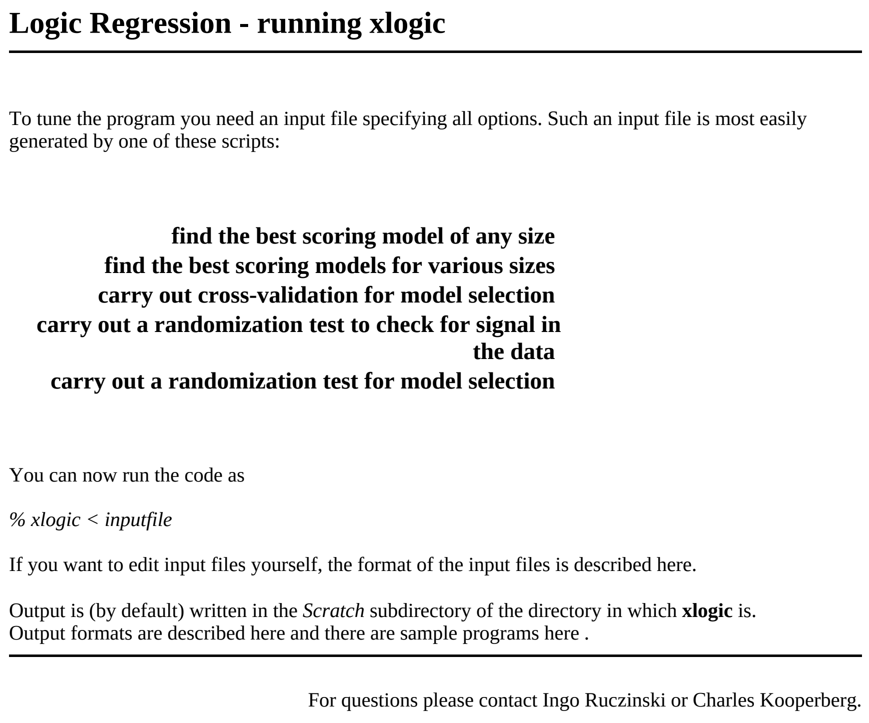
The current version of the code is 0.1.3 dated July 17, 2001 (changelog).

The logic regression methodology was developed by Ingo Ruczinski, Charles Kooperberg, and Michael LeBlanc at the Fred Hutchinson Cancer Research Center in Seattle. The copyright of the logic regression code is owned by Ingo Ruczinski, Charles Kooperberg, and Michael LeBlanc. You are free to use the software, for non-commercial purposes only, if:
(1) Copyright notices are not removed.
(2) Publications using logic regression refer to: *Ruczinski I, Kooperberg C, LeBlanc ML (2001), Logic Regression, manuscript.* or *Kooperberg C, Ruczinski I, LeBlanc ML, Hsu L (2001), Sequence Analysis using Logic Regression, Genetic Epidemiology, to appear.*

For questions please contact Ingo Ruczinski or Charles Kooperberg.

Figure 20.1. The Logic Regression menu, as of October 2001, available from `http://bear.fhcrc.org/~ingor/logic/` Online, you can click any of the links, indicated by the bold face fonts, to find out more about that topic.

a configuration of the problem under investigation. The states are related by a neighborhood system, and the set of neighboring pairs in S defines a substructure M in $S \times S$. The elements in M are called moves. Two states s, s' are called adjacent, if they can be reached by a single move (i. e. $(s, s') \in M$). Similarly, $(s, s') \in M^k$ are said to be connected via a set of k moves. In our application, the state space is finite. The basic idea of the annealing algorithm is: given the current state, pick a move according to a selection scheme from the set of permissible moves, which leads to yielding a new state. Compare the scores of the old and the new state. If the score



Logic Regression - running xlogic

To tune the program you need an input file specifying all options. Such an input file is most easily generated by one of these scripts:

find the best scoring model of any size
find the best scoring models for various sizes
carry out cross-validation for model selection
carry out a randomization test to check for signal in the data
carry out a randomization test for model selection

You can now run the code as

% xlogic < inputfile

If you want to edit input files yourself, the format of the input files is described here.

Output is (by default) written in the *Scratch* subdirectory of the directory in which **xlogic** is.
Output formats are described here and there are sample programs here .

For questions please contact Ingo Ruczinski or Charles Kooperberg.

Figure 20.2. The features of the software as of October 2001, available from `http://bear.fhcrc.org/∼ingor/logic/running/running.html` Online, you can click any of the links, indicated by the bold face fonts, to get the templates for the input file needed to run the program.

of the new state is better than the score of the old state, accept the move. If the score of the new state is not better than the score of the old state, accept the move with a certain probability. This acceptance probability depends on the difference of the scores of the two states under consideration and a parameter that reflects at which point in time the annealing chain is (this parameter is usually referred to as the temperature). For any pair of scores, this probability decreases during the algorithm. For infinitely long algorithms with slowly decreasing temperatures it can be established that the best state is reached. However, even when that is not the case, this algorithm generally leads to good-scoring states.

In our case, a state is a Logic Tree. Given the current tree, we randomly pick, following a pre-determined distribution, a candidate from the move set for this tree. We re-fit the parameters for the new model, and determine its score, which we then compare to the score of the previous state (Logic model), and repeat the process. There are various possibilities how to implement the annealing algorithm and fit the Logic models. This requires,

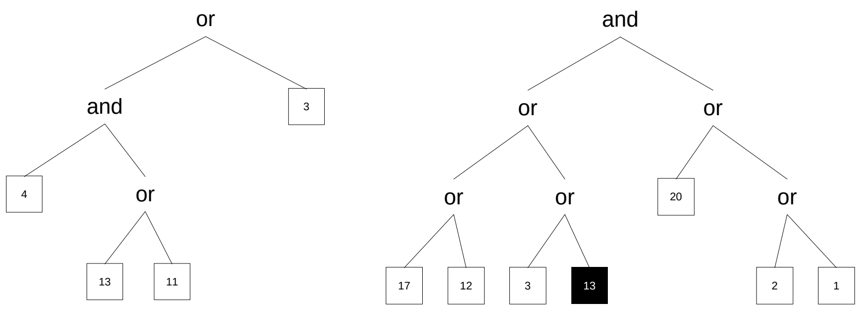


Figure 20.3. Results of letting *xlogic* find the best model of any size with two logic trees on the simulated data set

for computational reasons, that we pre-select the number t of trees. Unless we have an idea of how many trees we maximally want to fit, it may not be clear a priori what this number should be. We generally pick larger than necessary t and trim the model down if needed. Our simulated annealing algorithm has similarities with the Bayesian CART algorithm [2], in which a CART tree is optimized stochastically. Both of these algorithms are distinct from the greedy algorithm employed by CART, in that at any stage they not necessarily pick the move that improves the score the most.

Example

We simulated a data set with 500 cases and 20 binary predictors. Each predictor k is simulated from as an independent Bernoulli random variables, with success probability p_k between 0.1 and 0.9. The response variable is simulated from the model

$$Y = 3 + 1L_1 - 2L_2 + Z, \tag{20.1}$$

where $L_1 = (X_1 \vee X_2)$ and $L_2 = (X_3 \vee X_4)$, and Z is independent standard normal noise. We use linear regression within the logic regression framework to find L_1 and L_2. The results of letting *xlogic* find the best model of any size with two logic trees is shown in Figure 20.3. The logic trees in these figures are read upside down; for example, in the left-hand side of this figure $L_1 = (X_4 \wedge (X_{13} \vee X_{11})) \vee X_3$. As can be seen, these trees are too large, and model selection needs to be carried out.

While the example in this chapter uses linear regression, all modeling options can also be applied to any other regression model with an appropriately defined score function.

20.2.3 *Find the best scoring models for various sizes*

In certain situations it is of interest to know what the best scoring logic regression model of a certain size is. The size of a logic regression model is defined as the total number of leaves in all Logic Trees combined, thus the

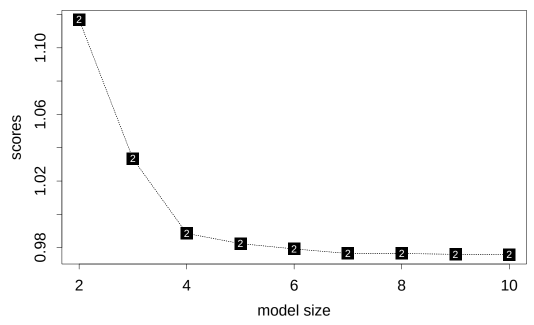


Figure 20.4. Scores of the best model of a specific size when two logic trees are fit to the simulated data set. The white 2 denotes that the logic model was fit with two trees. In general, models with various numbers of trees are fitted, but only the scores for two-tree models of sizes 2 through 10 are shown here.

model displayed in Figure 20.3 has size 11. Finding models of a fixed size is essential when using cross-validation to determine the best overall model size, as discussed below. For the simulated annealing algorithm described above, the tree or model size changes constantly, and the final model can be of any size. The straightforward solution to find the best scoring model of a fixed size would be to alter the move set, and only allow moves that keep the size of the model constant. However, this turns out to be computationally inefficient, as the resulting chains do not mix very well because the move set becomes more complicated. Thus, instead we do allow moves that increase or decrease the size of the Logic Regression model, but we prohibit moves that increase the model size when its desired size has been reached. Strictly speaking, this guarantees us only to find the best of **up to** the desired size. In reality, the maximum (desired) tree size almost always is reached, provided this size is not too large.

Example (cont.)

In Figure 20.4 we show the score of the best Logic Regression model of size for a variety of sizes two through ten, when two logic trees are fit. We note that the score improves considerably up to size four, and levels out after that. In fact,, the best logic model of size four has the correct L_1 and L_2 in model (20.1).

20.2.4 Carry out cross-validation for model selection

Searching for the globally best scoring model on the entire data, we know that the model with the best predictive capability may be smaller than the model we find via simulated annealing. We therefore want to compare the performances of the best models for different sizes. This can be done using an independent test set or by cross-validation. When sufficient data are available, we prefer the training set/test set approach. Otherwise, we can use cross-validation instead. Assume we want to assess how well the best model of size k performs in comparison to models of different sizes. We split the cases of the data set into m (approximately) equally sized groups. For each of the m groups of cases (say group i), we proceed as follows: remove the cases from group i from the data. Find the best scoring model of size k (as described in the previous section), using only the data from the remaining $m-1$ groups, and score the cases in group i under this model. This yields score ϵ_{ki}. The cross-validated (test) score for model size k is $\epsilon_k = \frac{1}{m}\sum_i \epsilon_{ki}$. We can compare the cross-validated scores for models of various sizes.

Example (cont.)

In Figure 20.5 we show both the average training and (cross-validation) test score. As can be seen, the training scores decrease as the model size increases, but the test scores are minimized for model sizes four and five.

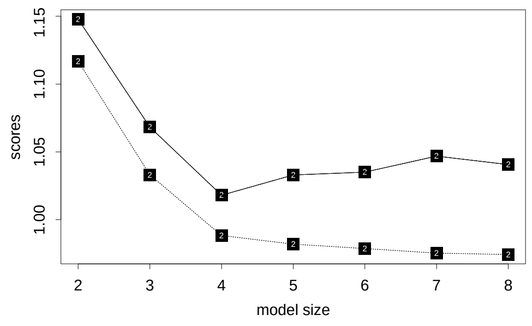


Figure 20.5. Training (dashed) and test (solid) scores of the cross-validation model selection for the simulated data set

20.2.5 *Carry out a randomization test to check for signal in the data*

The first step in our analysis usually is check for signal in the data. To do this, we first find the best scoring model, given the data. The null hypothesis which we want to test is: "there is no association between X the predictors and the response. If that hypothesis was true, the best model fit on the data with the response randomly permuted should yield about the same score as the best model fit on the original data. We carry out this randomization procedure as often as desired, and claim the proportion of scores better than the score of the best model on the original data as an p-value, indicating evidence against the null hypothesis.

Example (cont.)

In Figure 20.6 we show a histogram of 50 scores of the best model based on randomized data for the simulated example. As can be seen, these scores are considerably worse than the true best model, making us believe that there is signal in the data.

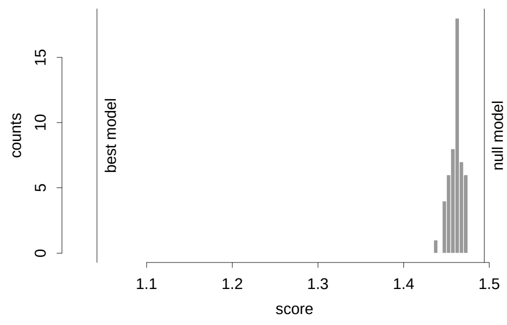


Figure 20.6. Histogram of 50 scores of the best model based on randomized data for the simulated example.

20.2.6 Carry out a randomization test for model selection

We can carry out a similar randomization test to find the model size. First, we find the best scoring model, with score ϵ^*, say. Assume that this model has size k. We also find the best scoring models of sizes 0 through k. The null hypothesis for each sequential randomization test is: "the optimal model has size j, the better score obtained by models of larger sizes is due to noise", for some $j \in \{0, \ldots, k\}$. Assume that such a null hypothesis is true, and the optimal model size is j, with score ϵ_j. We now "condition" on this model, considering the fitted values of the Logic model. For a model with p Logic Trees, there can be up to 2^p fitted classes (one for each combination of the p Logic Trees $L_1, \ldots, L_p$). We now randomly permute the response within each of those classes. The exact same model of size j considered still scores the same, say ϵ_j (other models of size j potentially could score better). If we now fit the overall best model (of any size), it will have a score ϵ_j^{**}, which is as least as good, but usually better, than ϵ_j. However, this is due to noise! If the null hypothesis was true, and the model of size j was indeed optimal, then ϵ^* would be a sample from the same distribution as ϵ_j^{**}. We can estimate this distribution as closely as desired by repeating this procedure multiple times. On the other hand, if the optimal model had a size larger than j, then the randomization would yield on average worse scores than ϵ^*.

We carry out a sequence of randomization tests, starting with the test using the null model, which is exactly the test for signal in the data as described in the previous subsection. We then condition on the best model of size one and generate randomization scores. Then we condition on the best model of size two, and so on. Comparing the distributions of the randomization scores, we can make a decision regarding which model size to pick.

Example (cont).

In Figure 20.7 we see the results of the randomization tests conditioning on models of size 3, 4 and 5. We note that the best score is considerably better than all scores based on randomized data sets conditioned on the model of size three, but that this is no longer true when we condition on the models of size four or five. This, again, suggests that the best model does indeed have the (correct) size four.

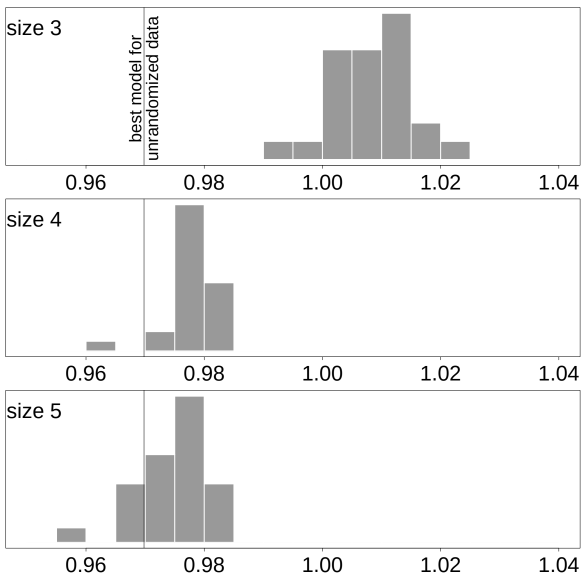


Figure 20.7. Histogram of 25 scores of the best model of fixed size for the conditional randomization tests the simulated example.

20.3 Conclusion

Logic Regression considers a novel class of models to detect interactions between binary predictors that are associated with a response variable. We developed the Logic Regression methodology with some statistical genetics problems in mind, but also found applications in other areas such as medicine and finance. Areas of current and future research include assessing model uncertainty using McMC, alternative ways for model selection, such as penalized scoring functions, and development of models that take familial dependence in genetic data into account. We also work on software improvements.

References

[1] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C.J. (1984), *Classification and Regression Trees*, Belmont, CA: Wadsworth.

[2] Chipman, H., George, E., and McCulloch, R. (1998). Bayesian CART model search (with discussion). *Journal of the American Statistical Association*, 93, 935–960.

[3] Kooperberg, C., Ruczinski, I., LeBlanc, M. L., and Hsu, L. (2001), "Sequence Analysis using Logic Regression", *Genetic Epidemiology*, 21 (S1), 626–631.

[4] Otten, R. H., and Ginneken, L.P. (1989), *The Annealing Algorithm*, Boston: Kluwer Academic Publishers.

[5] Ruczinski, I., Kooperberg, C., LeBlanc, M. L. (2001), "Logic Regression", (under review, draft available from `http://bear.fhcrc.org/~ingor/html/publications.html`).

[6] van Laarhoven, P. J., and Aarts, E. H. (1987), *Simulated Annealing: Theory and Applications*, Boston: Kluwer Academic Publishers.