18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

Nonsmooth Optimization using Classification and Regression Trees

Robertson, B. L.[1], **C. J. Price**[1], **and M. Reale**[1]

[1] *Department of Mathematics and Statistics, University of Canterbury, Christchurch, New Zealand*
Email: bro29@student.canterbury.ac.nz

Abstract: Nonsmooth local optimization problems occur in many fields, including engineering, mathematics, and economics. In economics, nonsmooth problems can be found in finance, mathematical economics and production theory. Examples include, nonsmooth utility maximization and exact penalty functions. However there are few convergent optimization algorithms to solve general nonsmooth or discontinuous problems. Random search methods can be applied to such problems because they do not require gradient information. However such methods search for global, rather than local, solutions and are often computationally expensive to use in practice.

To apply random search methods to nonsmooth local minimization we employ techniques from classification theory, in particular classification and regression trees (CART). CART provides a way of partitioning an optimization region S into sub-regions. Imposing a classification on a set of random points $\{x : x \in S\}$ with respect to function values allows a partition to be formed. Here we consider points are of two categories, either high or low. Hence, each sub-region of the partition is classified as either high or low. The sub-regions are defined by hyperrectangles because binary classification trees are used.

A local minimization algorithm is introduced. The method is set up to solve nonsmooth or discontinuous problems in an n-dimensional box S. The algorithm alternates between a partition and sampling phase. Firstly the optimization region is partitioned with CART using a training data set T, identifying low and high sub-regions. A new batch of points is then distributed into the low regions with an increased probability distribution. The new points are added to the training data and the method repeats, until a stopping rule terminates the algorithm.

We have found that the use of CART in nonsmooth optimization is very effective. The CART procedure itself in computationally cheap to evaluate and identifies low sub-regions of S well. The hyperrectangular sub-regions are easy to resample, allowing for our iterative resampling algorithm. This method is applicable to nonsmooth and discontinuous objective functions.

Keywords: *Classification and regression trees (CART), nonsmooth optimization, random search.*

1 INTRODUCTION

Local optimization problems occur in many different fields including, engineering, mathematics and economics. Many efficient methods exist for solving local optimization problems when the objective function is smooth. For example, Quasi-Newton methods and the Simplex algorithm of Nelder et al. (1965). However when the objective function is nonsmooth or discontinuous, there are very few convergent algorithms for general functions.

In this paper we are interested in the constrained minimization problem

$$\min_{x \in S}(f(x)) \tag{1}$$

where the objective function f maps S into $R \cup \{+\infty\}$. Our primary interest lies with objective functions which are nonsmooth or discontinuous. The optimization region S is defined by an n-dimensional hyperrectangle of the form,

$$S = \{x \in R^n : a \leq x \leq b\},$$

where $a, b \in R^n$ are finite with $a < b$. Under appropriate scaling S can be modified to $S = [-1,1]^n$, which is used hereon in. Here we search for an essential local minimizer x_* of f, where $f(x_*)$ is assumed to be finite.

Definition 1 *An essential local minimizer x_* is a point for which the set*

$$\Psi(x_*, \varepsilon) = \{x \in R^n : f(x) < f(x_*) \text{ and } \|x - x_*\| < \varepsilon\}$$

has Lebesgue measure zero for all sufficiently small positive ε*.*

If the objective function is continuous at x_*, then x_* is also a local minimizer in the classical sense. The inclusion of $+\infty$ means the method can be applied to extreme barrier functions Audet et al. (2003). Thus a nonhyperrectanglar optimization region $\bar{S}$ can be considered by choosing a hyperrectangle S such that $\bar{S} \subset S$ and setting $f(x) = +\infty$ when $x \notin \bar{S}$.

Nonsmooth local optimization problems exist in the economics field including, finance, mathematical economics and production theory. Examples of such functions are exact penalty functions which replace constraints by introducing nonsmooth cost terms, and nonsmooth utility functions. Vinter et al. (2003) considers two nonsmooth finance problems. Firstly, the maximization of a nonsmooth utility in an investment problem and secondly, the calculation of the duration of a bond for general term structures of interest rates. Bouchard et al. (2004) also considers nonsmooth utility maximization, where nonsmooth analysis is applied to incomplete markets. In addition, Tanaka (2008) uses nonsmooth optimization in production theory to include a broader range of production functions.

Random search methods can be applied to nonsmooth functions because no gradient information is required. The simplest random search method is Pure Random Search. The idea is to simply sample points from a common distribution G over S until sufficiently confident that the lowest of these values is a global minimum to within ε. However random search methods search for global, rather than local, solutions and are often computationally expensive to use in practice. Furthermore, for the purposes of nonsmooth minimization, rather than sampling from a common G over S, it would be advantageous to sample from a conditional probability distribution. In particular, a distribution that adds mass to regions of S where relatively low function values have been obtained. Here we propose a new random search algorithm which partitions S into sub-regions and samples sub-regions with relatively low function values with an increased probability distribution. In sections 2 and 3 we use classification methods to identify low sub-regions of S. Section 4 describes a nonsmooth optimization algorithm and concluding remarks are given in section 5.

2 CLASSIFICATION

Classification is the task of assigning objects x to one of several predefined categories Ω. The finite set of categories has the form $\Omega = \{\omega_1, \omega_2, \ldots, \omega_M\}$. Hereon in objects are n-dimensional vectors $x = \{x_1, x_2, \ldots, x_n\}$. A classifier is a systematic approach to constructing a classification model from a training data set. More formally a classifier is the mapping $g(x) : S \subset R^n \to \Omega$, where S is the sample space. The classification model is used to classify future unknown objects under the mapping. Examples of classifiers include decision tree classifiers, neural networks, support vector machines, and k-nearest neighbor Duda et al. (2001), Tan et al. (2006). Here a new application of classification methods is proposed for optimization purposes. In our particular application we consider decision tree classifiers.

2.1 Classification Trees

An intuitive way to classify an object is through a sequence of questions, whereby the next question depends on the previous answer. Such a procedure is displayed in a directed decision tree, or simply a tree. A decision or classification tree represents a multi-stage decision process, whereby a decision is made at each node. Trees have either binary or multivalued decisions at each node. Binary decisions result in binary splits producing two descendent nodes, whereas multivalued decisions result in multiple (> 2) descendent nodes. However any tree can be represented using just binary decisions and thus only binary splits are required. We refer the reader to Devroye et al. (1996) for a particular mapping, oldest-child/next-sibling binary tree, which maps a multivalued tree onto an equivalent binary tree. Thus with out loss of generality we consider binary trees only.

The tree consists of *nodes* and *branches*, where by convention the first or *root* node is at the top of the tree. Nodes are linked to other nodes with branches. A node is either *internal* or *terminal*, where internal nodes have descendant or *child* nodes. Internal nodes are split into left and right child nodes, while terminal nodes have no descendant nodes. Each terminal node has an associated category and observations that end on a particular terminal node are assigned that category.

To classify a particular object x using a classification tree we begin at the root node. A binary decision "true/false" is made with respect to a particular feature $s \in R$. If the decision is true we proceed to the left child node, otherwise we proceed to the right child node. Continuing in this manner we eventually reach a terminal node. In doing so we assign x to the category $\omega_i \in \Omega$ of that terminal node.

Here all descendant nodes are numbered with reference to there parent node. Specifically, if node(D) is internal then the left and right child nodes are numbered node$(2D)$ and node$(2D+1)$ respectively. Hence if a node number is even, the decision at the *parent* node was true. This unique numbering facilitates a backtracking strategy to determine the unique path from the root node to each terminal node, using only the terminal node number (see section 3.4).

2.2 Optimization Application

Classification trees partition S into sub-regions when numerical data sets are used. Let the sample space be defined by $S = [-1,1]^n$ with all $x \in S$. Each node in a classification tree represents a sub-region of S. The root node represents S itself, and all descent nodes satisfy the following requirements. If node(D) represents the sub-region A, with descendent nodes $2D$, $2D+1$ with sub-regions a_1, a_2 respectively, then $A = a_1 \cup a_2$ and $a_1 \cap a_2 = \emptyset$. Hence the union of all sub-regions A_i defined by terminal nodes alone, where $i = 1,\ldots,|\text{terminal nodes}|$, partitions S into $|\text{terminal nodes}|$ nonempty sub-regions.

Each tree method results in a different partition of S based on the form of the decision or query at each node. The simplest approach to consider is binary classification trees, which use queries of the form: Is $x_j < s$? Such queries lead to hyperrectangle sub-regions parallel to the coordinate axes. Binary space partition trees (BSP) use queries of the form: Is $c_1x_1 + c_2x_2 + \ldots + c_nx_n < s$? This results in S be partitioned into convex polyhedral sub-regions. Although there is more flexibility in how S is partitioned, evaluating such queries can be computationally expensive in practice. Another approach, sphere trees, uses queries of the form: Is $\|x - z\| < s$? (where $z \in S$ chosen at each node). The resulting partition of S has sub-regions defined by pieces of spheres.

In our application we wish to partition an optimization space, given by S, into regions where the objective function is relatively low or relatively high. Here two categories are chosen $\Omega = \{\omega_L, \omega_H\}$, corresponding to low and high points respectively. These sub-regions are resampled and S is partitioned once more using both the new and existing sample points. This is the basis of our new nonsmooth optimization algorithm, see section 4. Thus a series of partitions are calculated and sub-region resampling is required. Hence, we want a partition that is computationally cheap to obtain, and sub-regions which facilitate resampling. The choice is obvious, binary classification trees, as they are computationally cheap and produce hyperrectanglar sub-regions which are simple to resample.

3 CART

We now turn to the practical question of how to build a binary classification tree using a training data set. In this section we use a training data set T of $N > 0$ sample points x, defined by

$$T = \{x^{(i)} \in [-1,1]^n : i = 1,\ldots,N\}.$$

T is assumed to be of two categories ω_L and ω_H, with both $|\omega_L|, |\omega_H| > 0$. The notation $x_j^{(i)}$ is used to denote the j^{th} coordinate of the i^{th} sample point.

In principle, many different binary classification trees can be constructed from a training data set, however some efficient algorithms exist. These algorithms proceed in a greedy manner, whereby a series of locally optimal decisions are made. Such algorithms include ID3, C4.5, and classification and regression trees (CART) Duda et al. (2001), and it is the latter which we consider here. CART provides a general framework that can be implemented in many ways to produce different classification trees. Here we consider a particular strategy designed for partitioning optimization spaces. In our approach, four general questions arise:

1. Where are potential splits in the training data?
2. Which feature should be used to split a node?
3. When should a node be declared a terminal node?
4. How are the bounds of each sub-region obtained?

We consider each of these questions in turn.

3.1 Locating Potential Splits

Given a total of N sample points, often all $n(N-1)$ possible splits are considered potential in CART tree growing procedures Duda et al. (2001). However as both N and n increase such a method can be computationally expensive. Furthermore all potential splits can only occur between elements of ω_L and ω_H, and so if $|\omega_H| \gg |\omega_L|$, computational efficiency is lost. Thus we employ a method which only considers splits of the form $s = (x_j + y_j)/2$ such that $x \in \omega_L$ and $y \in \omega_H$.

Consider locating all potential splits in the j^{th} dimension. The notation $\mathbf{1}_a$ is used to denote a vector $[1, \ldots, 1]$ of length a. Let $T_j = \{\omega_{Lj}, \omega_{Hj}\}$, and Y be the ordered set (ascending) with index vector I_Y such that $Y(I_Y) = T_j$. Setting three vectors $X_1 = [\mathbf{1}_{|\omega_L|}, 2.\mathbf{1}_{|\omega_H|}]$, $X_2 = [X_1(I_Y), 0]$, and $X_3 = [0, X_2]$, each potential split is located at minimal cost. Specifically each split occurs when $X_2 + X_3 = 3$. Let $Y_P = \{i : (X_2 + X_3)_i = 3\}$ an index set, then for each $i \in Y_P$ a potential split occurs at $s = (Y(i) - Y(i-1))/2$.

3.2 Feature Selection and Node Impurity

When growing a classification tree the fundamental principle is that of simplicity. Specifically, decisions that lead to a compact tree with few nodes are preferred. Thus at each node(D) we look for a feature that makes the descendant nodes as pure as possible. By convention we refer to a node's impurity rather than how pure it is. There are various measures of node impurities, all of which satisfy the following requirements. Let $i(D)$ denote the impurity at node(D), then $i(D)$ must be zero when node(D) is pure and a maximum when the categories are equally represented. Impurity measures include Gini, Classification Error, and Entropy. Here we choose to use the most popular measure Duda et al. (2001), entropy measure,

$$i(D) = -\sum_j P(\omega_j) \log_2 P(\omega_j),$$

where $P(\omega_j)$ is fraction of points at node(D) that are category ω_j and $j \in \{L, H\}$. We note here that $0 \log_2(0) = 0$ in entropy calculations.

Given a partial tree down to node(D), the question now arises: which feature gives the optimal split? Here we have n features to consider, one for each dimension of the objective function. Each feature test or query is of the form "Is $x_j^{(i)} \leq s$?", where $-1 < s < 1$ is the scalar splitting value. Using a greedy strategy, we choose the feature that decreases the impurity as much as possible. The drop in impurity is simply,

$$\Delta i(D) = i(D) - P_L i(D_L) - (1 - P_L) i(D_R), \tag{2}$$

where D_L and D_R are left and right child nodes, $i(D_L)$ and $i(D_R)$ are their impurities, and P_L is the fraction of points at node D that will go to D_L after the split. By means of exhaustive search over all potential splits, the optimal split is found. Sometimes there are several optimal splits which yield the same $\Delta i(D)$, in which case the first instance is usually chosen, as is done here. We note here that although each split is locally optimal, the fully grown tree is not necessarily optimal, i.e. a series of locally optimal decisions does not imply global optimality.

We mention here that two general splits can be considered. Firstly, the *forced split*, where a split in the j^{th} dimension is required. The j^{th} dimension split which minimizes (2) is chosen as the split. Secondly, the *free split*, where a split from any dimension is chosen to minimize (2). Here we only consider free splits.

3.3 When To Stop Node Splitting

We now consider the problem of when to stop node splitting. One strategy is to continue splitting nodes until all terminal nodes are pure. However, such a strategy can lead to large, complicated trees with many nodes, and hence a complicated partition. Another method is to stop splitting when a predefined maximum number of nodes for a tree is reached. Here we continue splitting until an *optimal partition* is achieved, defined formally in the following definition.

Definition 2 *Let* $|\omega_q(D)| : q \in \{L,H\}$ *denote the number of low/high points at node(D), and impurity tolerance* $0 < \tau < 1$. *Then an optimal partition is achieved if each terminal node*(D) *in the current tree satisfies one of the following conditions:*

- $i(D) = 0$*, the node is pure, or*
- $|\omega_L(D)| > |\omega_H(D)|$ **and** $i(D) < \tau$.

Definition 2 allows some terminal nodes to be impure by misclassifying high points. This onesided misclassification is used to potentially simplify the partition. Here we are concerned with function minimization, and hence if a few *rogue* high points over complicate our model they could be ignored. Misclassifying low points may simplify the partition but may be problematic for optimization purposes. Consider $\{x \in \omega_L : \|x - x_*\| < \varepsilon\}$ with $\varepsilon > 0$ and x_* an essential local minimizer. Then misclassifying x could result in the classification model asserting points sufficiently close to the solution are high, even though a low point has been sampled there.

A tolerance value $\tau = 0.45$ is used hereon in. Thus a terminal node with $|w_L| \geq 10$ and $|w_H| = 1$ would be split no further, and the corresponding low sub-region would contain one high point.

3.4 Defining CART Sub-regions

Each terminal node in the classification tree corresponds to a sub-region of the partition. As mentioned earlier each node has a unique node number associated with it, in particular each terminal node. Such numbering facilitates a backtracking procedure to obtain the bounds on each sub-region by retracing the unique path from each terminal node to the root node.

The unique path vector Φ to a terminal node, with node number D, is calculated as follows: Set $\Phi(1) = D$. Calculate each internal node number on the path sequentially using $\Phi(j) = \lfloor \Phi(j-1)/2 \rfloor$ for integer $j > 1$, until the root node is found, $\lfloor \Phi(j) \rfloor = 1$. Here $\lfloor a \rfloor$ is the largest integer not exceeding the real number a. Sorting Φ into ascending order gives the unique path from the root node to terminal node(D).

A matrix B is used to store the bounds on each sub-region A_i. The notation B_i is used to denote the i^{th} row of the matrix B. The size of B is |terminal nodes| by $2n$ and each row has the following structure,

$$B_i = [l_1, \ldots, l_n, L_1, \ldots, L_n],$$

where l_j and L_j denote the lower and upper bound in the j^{th} dimension respectively. Initially each row of B contains the bounds of the optimization region S,

$$B = \begin{bmatrix} -1 \ldots -1 \; 1 \ldots 1 \\ \vdots \qquad \vdots \; \vdots \quad \vdots \\ -1 \ldots -1 \; 1 \ldots 1 \end{bmatrix}.$$

Each B_i is updated iteratively using queries along the unique path Φ_i, the i^{th} path to the i^{th} terminal node. Each query at a internal node(D) is of the form: Is $x_j < s$. Thus if node$(2D) \in \Phi_i$, we traveled to the left child after the split, answering *yes* to our query. Hence s is an upper bound for the j^{th} dimension and $B_i(n+j) = s$. Otherwise node$(2D+1) \in \Phi$, in which case $B_i(j) = s$, a new lower bound. Elements of B_i are updated until the terminal node is reached. We note here that an element of B_i may be updated more than once.

4 NONSMOOTH LOCAL OPTIMIZATION METHOD

We now propose our new random search nonsmooth optimization algorithm to solve (1). Initially a batch of $N > 0$ points are drawn from a uniform distribution over S. Evaluating f at each sample point gives us our first training data set T. The algorithm then alternates between two phases, a partition and a sampling phase. We consider each in turn.

During the partition phase, low sub-regions of S are defined using the CART method described above. However, to yield such a partition, a classification must be imposed on the training data. We note there is a great deal of freedom when choosing this classification. Here we simply choose, $T = \{\omega_L, \omega_H\}$, where $\{\omega_L\}$ is the $0.8N$ elements of T with the lowest function values and $\{\omega_H\} = T \setminus \{\omega_L\}$. With T defined the partition is obtained.

The sampling phase samples low sub-regions identified by CART with an increased probability distribution. Again there are numerous possibilities here. For our purposes, the next batch of N points are simply drawn uniformly from the set,

$$\{x : x \in \bigcup A_i\}, \tag{3}$$

where each A_i is a low sub-region obtained from the partition. The new batch of points is added to T and used during the next partition phase.

The algorithm terminates when a stopping rule is satisfied. The sample point with the lowest function value is taken as the essential local minimizer x_*.

4.1 Remarks

With $|\omega_L| = 0.8N$ for all iterations, the maximum number of potential splits is fixed and does not increase with $|T|$. Furthermore, excluding the first iteration, $|\omega_H| > |\omega_L|$ which causes ω_L to cluster, eventually in the neighborhood of x_*.

To select a point uniformly from (3) a two stage process is used. Firstly an inverse transform method is used to select a low sub-region A_i,

$$A_i = \max(i : U \leq D_F(A_i)),$$

where $U \in [0,1]$ is a random variable, and D_F is the distribution function of $m(A_i)$ (Lebesgue measure). A point x is then drawn from a uniform distribution over A_i.

4.2 Convergence

It has been shown by Price et al. (2008) that finding a descent step for a nonsmooth optimization problem is closely related to a global optimization problem on a subset of S. Choosing a lower bound $\delta > 0$ on hyperrectangle side length for each A_i, ensures that $m(\cup A_i) > 0$ for all iterations. Convergence, in a probabilistic sense, can then be demonstrated for general nonsmooth problems.

5 DISCUSSION AND CONCLUSIONS

A random search local optimization method has been presented. The method uses classification and regression trees to partition the optimization region into sub-regions. These sub-regions are defined by hyperrectangles and classified as either high or low with respect to function value. Low sub-regions are sampled with an increased probability distribution. The partition, resample strategy is an effective method for minimizing nonsmooth or discontinuous objective functions.

Currently a more advanced algorithm (CARTopt) using the partition described above is being refined. A new transform technique is applied to the training data which potentially simplifies the partition at each iteration. Additional conditions are imposed on each low sub-region and the training data has a fixed maximum size. The method has a convergence proof for locating an essential local minimizer on general nonsmooth and discontinuous functions. In addition, a stopping rule based on the distribution of function values near essential local minimizers has been developed. Initial results show that CARTopt is competitive in practice, requiring relatively few function evaluations to solve a selection of nonsmooth test problems ranging in dimension from $n = 2 - 8$.

REFERENCES

Audet, C., and J. E Dennis Jr. (2003), Analysis of generalized pattern searches, *SIAM J. Opt.*, 13, 889-903.

Bouchard, B., N. Touzi, and A. Zeghal (2004), Dual formulation of the utility maximization problem: The case of nonsmooth utility, *The Annals of Applied Probability*, 14(2), 678-717.

Devroye, L., L. Györfi, and G. Lugosi (1991), A probabilistic theory of pattern recognition, Springer, 636 pp. New York.

Duda, R. O., P. E. Hart, and D. G. Stork (2001), Pattern Classification, Wiley Interscience, 645 pp., New York.

Nelder, J. A., and R. Mead (1965), A simplex method for function minimization, *The Computer Journal*, 7, 308-313.

Price, C. J., M. Reale, B. L. Robertson (2008), A direct search method for smooth and nonsmooth unconstrained optimization, *ANZIAM J.*, 48, 927-948.

Tan, P., M. Steinbach, and V. Kumar (2006), Introduction to Data Mining, Addison Wesley, 769 pp.

Tanaka, Y. (2008), Nonsmooth optimization for production theory, discussion paper from Graduate School of Economics and Business Administration, Hokkaido University, Japan.

Vinter, R. B., and H. Zheng (2003), Some finance problems solved with nonsmooth optimization techniques, *Journal of Optimization Theory and Applications*, 119(1), 1-18, October.