Binary Split Categorical Feature with Mean Absolute Error Criteria in CART

Peng Yu[1,2,3], **Yike Chen**[1], **Chao Xu**[1*], **Albert Bifet**[3,4], **Jesse Read**[5]
[1] University of Electronic Science and Technology of China
[2] Shopify
[3] Télécom Paris, Institut Polytechnique de Paris
[4] The University of Waikato
[5] École Polytechnique, Institut Polytechnique de Paris
peng.yu@shopify.com, cyike9982@gmail.com, cxu@uestc.edu.cn, albert.bifet@telecom-paris.fr, jesse.read@polytechnique.edu

Abstract

In the context of the Classification and Regression Trees (CART) algorithm, the efficient splitting of categorical features using standard criteria like GINI and Entropy is well-established. However, using the Mean Absolute Error (MAE) criterion for categorical features has traditionally relied on various numerical encoding methods. This paper demonstrates that unsupervised numerical encoding methods are not viable for the MAE criteria. Furthermore, we present a novel and efficient splitting algorithm that addresses the challenges of handling categorical features with the MAE criterion. Our findings underscore the limitations of existing approaches and offer a promising solution to enhance the handling of categorical data in CART algorithms.

Introduction

The CART family of algorithms (random forest, gradient boosting tree) is well-known for its top performance on tabular data. Real-world tabular data often contains not only numerical but also categorical features. The CART algorithm recursively partitions the input dataset with a binary split optimization step and terminates when reaching a minimum number of instances. While traditional machine learning models only work with numerical data, the CART family of algorithms can process categorical features directly. This flexibility is because the binary split optimization step in the CART algorithm only requires feature data types that allow for different subsets.

The binary split step is recognized as a major bottleneck regarding the computational efficiency of tree learning algorithms (Catlett 1991). Specifically, when processing categorical features, the associated discrete set topology can result in an exponential search space for binary splits. As a result, various numerical encoding methods have been developed to address this limitation. Consequently, many popular tree-based machine learning software packages (such as XGBoost (Chen and Guestrin 2016), LightGBM (Ke et al. 2017), and Catboost (Prokhorenkova et al. 2018)) only support numerical data or include automatic numerical encoding methods for categorical data. On the other hand, only a subset of splitting criteria (such as mean squared error and Gini impurity) have optimally guaranteed numerical encodings for categorical data (Hastie, Tibshirani, and Friedman 2001). The splitting criterion MAE (Mean Absolute Error) lacks a proven optimal numerical encoding. MAE is more robust when dealing with outliers and skewed distributions and is widely adopted in various statistical domains. The most successful and practical numerical encoding method for this criterion is a median-based heuristic numerical encoding. This heuristic has been implemented in `scikit-learn`, and takes $O(n^2)$ time, where n is the dataset size (jiangfeng 2017). Unfortunately, $O(n^2)$ running time is too slow for practical purposes. The community suggests using subsampling to avoid the running time issue at the cost of finding a worse split, which is the standard in `LightGBM`.

Our Contributions We are motivated by two open problems: whether there is a numerical encoding that works for MAE, and if not, does there exist a fast exact algorithm for binary split through MAE?

1. We prove that no unsupervised numerical encoding method is optimal for MAE, and show a median heuristic could be twice the optimal. While the proof itself is relatively straightforward, the significance of this result is reflected in the substantial effort invested in seeking the optimal numerical encoding method. For instance, dozens of unsupervised numerical encoding methods are under development (McGinnis et al. 2018).

2. We develop an *exact* and completely new algorithm to solve the binary split of categorical features with the MAE criterion in $O(n \log n + k \log k \log n)$ time without using numerical encoding, where n is the number of data points and k is the number of categories. The new algorithm is faster than the current heuristics, and also gives the exact result. So it both handles real-world size data without subsampling and is completely optimal. The new algorithm may also hold independent interest, as it solves the *unimodal cost* 2*-median problem*, which generalizes various problems studied in computational geometry literature.

*Corresponding author.

Preliminaries

In regression tree learning, during a node split computation, the goal is to find a binary partition S, S^c of $Y \in \mathbb{R}^n$, based on some feature X. Here, S^c is the complement of S with respect to Y. When X is categorical, the goal can be further simplified as finding a binary partition of $\mathcal{Y} = \{Y_1, Y_2, \dots Y_k\}$, where $Y_i \subseteq \mathbb{R}$ is the subset of the target data points with category i in feature X. We assume the collection of data points in $\mathcal{Y}$ is a set, and they are all disjoint. This is only for the benefit of exposition. Everything would still hold with proper definition if repeated data points are allowed. One easy way is to assume that if there are copies of the same element, we just perturbed each one of them by an infinitesimal amount, so all data points are unique.

Depending on the splitting criteria, the partition must maximize or minimize an objective function. For a given set $\mathcal{S} \subseteq \mathcal{Y}$, the MAE is defined as

$$\mathbf{MAE}(\mathcal{S}) = \min_{a \in \mathbb{R}} \sum_{x \in \cup \mathcal{S}} |x - a|. \tag{1}$$

The a achieving the minimum is $\mathbf{M}(\cup \mathcal{S})$, where $\mathbf{M}(\cdot)$ is the median of the input. Define the error of a split $\mathcal{S}$ and $\mathcal{S}^c$ as $\lambda(\mathcal{S})$, which is the sum of their MAE.

$$\lambda(\mathcal{S}) := \mathbf{MAE}(\mathcal{S}) + \mathbf{MAE}(\mathcal{S}^c). \tag{2}$$

When using MAE as the splitting criterion, the objective value would be the minimum over all splits:

$$\lambda := \min_{\mathcal{S} \subset \mathcal{Y}, \mathcal{S} \neq \emptyset} \lambda(\mathcal{S}). \tag{3}$$

The problem of computing the split S, S^c that achieves the minimum $\lambda(S)$ is referred to as the *MAE split problem*.

To solve for the minimum, one can enumerate all subsets of $\mathcal{Y}$, resulting in an undesirable $O(2^k)$ search space. On the other hand, the community has developed numerous heuristic numerical encoding methods.

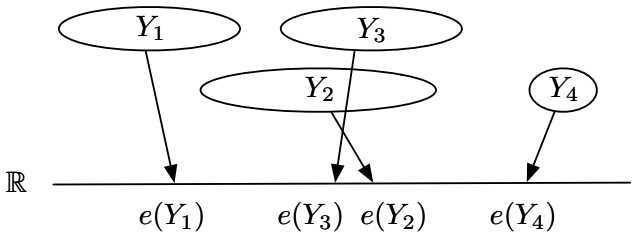

Figure 1: A example where $\mathcal{Y} = \{Y_1, Y_2, Y_3, Y_4\}$, and the encoding function e maps elements in $\mathcal{Y}$ to $\mathbb{R}$. There are 5 downward closed sets, and 3 of them splits, so $D(\mathcal{Y}, e) = \{\{Y_1\}, \{Y_1, Y_3\}, \{Y_1, Y_3, Y_2\}\}$.

Unsupervised Numerical Encoding Rather than enumerating all subsets, one can employ a numeric encoding/set function $e : 2^{\mathbb{R}} \to \mathbb{R}$ to establish an ordering $\preceq_e$ and select sets based on this ordering.

A numerical encoding function is considered unsupervised if specified independently of the data, meaning it is determined without observing the input.

The numerical encoding provides a natural ordering over the elements of $\mathcal{Y}$, where $A \preceq_e B$ if $e(A) \leq e(B)$. We define the *downward closed sets* of $\mathcal{Y}$ as $D'(\mathcal{Y}, e)$, which consists of sets of the form $\{A \mid e(A) \leq x, A \in \mathcal{Y}\}$ for some x.

Numerical encoding has been used as a heuristic to identify the optimal partition within the downward closed sets. Specifically, let $D(\mathcal{Y}, e) = D'(\mathcal{Y}, e) \setminus \{\mathcal{Y}, \emptyset\}$, which are downward closed sets that splits $\mathcal{Y}$ into two non-empty sets. See Figure 1 for example. Our goal is to find $\mathcal{S} \in D(\mathcal{Y}, e)$ that minimizes the objective λ. This modified problem leads to a more favorable $O(k)$ search space, although optimality is not guaranteed.

For instance, the median heuristic employs the encoding function $e = \mathbf{M}$ by arranging sets based on their medians. It then enumerates the downward closed sets and their complements as potential splits. Consequently, this approach yields only $k - 1$ potential splits, which can be tested one by one and return the optimum.

Piecewise-linear functions We also introduce a few useful functions and their properties. A function $f : \mathbb{R} \to \mathbb{R}$ is *unimodal*, if there exists a c such that for any $x \leq y \leq c$, we have $f(x) \geq f(y)$, and for $c \leq x \leq y$, $f(x) \leq f(y)$.

A function $h : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is *totally monotone*, if for any $x_1 \leq x_2 \leq y_1 \leq y_2$, $h(x_1, y_1) \geq h(x_1, y_2)$ then $h(x_2, y_1) \geq h(x_2, y_2)$. A positive linear combination of totally monotone functions is totally monotone. A matrix is totally monotone if the function $h(i, j) = M_{i,j}$ is totally monotone. The main property of a totally monotone matrix is the index for row minimum is non-decreasing. That is, if M_{i,a_i} is the minimum value of the ith row, then we have $a_1 \leq a_2 \leq \ldots \leq a_n$ (Park 1999).

We use $B(f)$ to denote the set of breakpoints for a piecewise-linear function. Recall the median function $\mathbf{M}(S)$ is an element $y \in S$, such that $\sum_{x \in S} |x - y|$ is minimized.

Splitting Criteria	Numerical Encoding	Optimal
MSE	Mean	✓
Gini impurity	Single class percentage	✓
MAE	One-hot	×
MAE	Median	×

Table 1: Comparison of unsupervised encoding for different criteria.

Numerical Encoding and Median Heuristic

Is there a numerical encoding that can be used to find the optimal binary split for MAE? Specifically, is there an encoding function e such that the following equality holds: λ is equal to $\min_{\mathcal{S} \in D(\mathcal{Y}, e)} \lambda(S)$?

Table 1 shows target mean-based numerical encoding for categorical features has been proven optimal in decision tree regression with mean squared error (MSE) in (Breiman et al. 1984), the same heuristic does not work with MAE.

Still, this does not rule out the existence of other unsupervised numerical encodings that work for MAE. Unfortunately, we prove that such encoding cannot exist.

Suppose a unique optimal partition of a dataset minimizes the MAE. In that case, any encoding that works for MAE must have the encoding of all elements in one partition strictly smaller than or greater than the encoding of the other partition. Formally, let $\{\mathcal{A}, \mathcal{B}\}$ forms the unique optimum partition of dataset $\mathcal{Y}$, and e is a encoding that *works* for MAE, then either $e(A) < e(B)$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$, or $e(A) > e(B)$ for all $A \in \mathcal{A}$ and $B \in \mathcal{B}$. If the encoding e works for MAE, then the optimal partition is in $D(\mathcal{Y}, e)$.

Theorem 1. *No numerical encoding function works for binary split with MAE splitting criteria.*

Proof. Assume such an encoding e exists and prove by contradiction via constructing a counter-example. Let $\epsilon > 0$ be some small and fixed value, say 0.01.

Let $a_1 = 0, a_2 = 2, a_3 = 3, a_4 = 5$. We define $A_i = \{a_i - \epsilon, a_i, a_i + \epsilon\}$, $A'_i = \{a_i - \epsilon, a_i + \epsilon, a_1\}$ if $i \in \{3, 4\}$, otherwise $A'_i = \{a_i - \epsilon, a_i + \epsilon, a_4\}$.

1. The unique optimum partition of $\{A_1, A'_1, A_4, A'_4\}$ is $\{A_1, A'_1\}, \{A_4, A'_4\}$, without loss of generality, let $e(A_1) < e(A_4)$.
2. The unique optimum partition of $\{A_2, A'_1, A_3, A'_4\}$ is $\{A_2, A'_1\}, \{A_3, A'_4\}$, hence $e(A'_1) < e(A'_4)$.
3. The unique optimum partition of $\{A_2, A'_2, A_3, A'_3\}$ is $\{A_2, A'_2\}, \{A_3, A'_3\}$, hence $e(A_2) < e(A_3)$.
4. The unique optimum partition of $\{A_1, A'_2, A'_3, A_4\}$ is $\{A_1, A'_3\}, \{A'_2, A_4\}$, hence $e(A_4) < e(A_1)$, a contradiction.

□

Now we know that any encoding-based heuristic would not give the correct result, but maybe it can still give a good enough result. To answer this, we examine the limitations of the most widely used encoding-based heuristic, the median heuristic.

Empirically, it has been shown that the median numerical encoding works most of the time for MAE splitting criteria (Torgo 1999). The conclusion is made based on experiments on limited datasets. The median encoding result in sub-optimal splits was only observed through some rare, randomly generated datasets.

However, we design an input so that the median heuristic is almost twice as bad as the optimum. Let n be an even integer. Consider 4 collections of data points Y_0, Y_1, Y_2, Y_3. Y_0, Y_1 consists of n copies of 0 and 1, respectively. Let Y_2 consist of $n/2$ copies of 0, $n/2 + 1$ copies of $0.5 + \epsilon$, and Y_3 consists of $n/2$ copies of 1 and $n/2 + 1$ copies of $0.5 - \epsilon$. Using the median heuristic, the potential points to split are $0, 0.5 - \epsilon, 0.5 + \epsilon, 1$. Observe that no matter which one is chosen, the output of the median heuristic would give a solution of value $n + 2\epsilon$. The actual optimal would give a value around $n/2 + 1$.

Methodology

Knowing that no unsupervised numerical encoding e works for MAE, there might still be efficient algorithms that give the exact solution and don't use any encoding. In the following section, we propose such an algorithm. The algorithm is a fairly complicated divide-and-conquer that uses tools in computational geometry.

We first transform the MAE split problem into a more manageable version, where instead of optimization of subsets (which can be as large as 2^k), it becomes optimizing over $O(n^2)$ points.

Lemma 1. *Let $\mathcal{Y}$ be a family of disjoint sets. Then*

$$\lambda = \min_{\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{Y}} \big(\textbf{\textit{MAE}}(\mathcal{S}) + \textbf{\textit{MAE}}(\mathcal{S}^c) \big)$$

can be equivalently written as

$$\lambda = \min_{a,b \in \mathbb{R}} \sum_{S \in \mathcal{Y}} \min\Big(\sum_{i \in S} |i - a|, \sum_{j \in S} |j - b| \Big).$$

Proof. By definition,

$$\textbf{MAE}(\mathcal{S}) = \min_{a \in \mathbb{R}} \sum_{i \in \bigcup \mathcal{S}} |i - a|.$$

Hence, for any subset $\mathcal{S} \subseteq \mathcal{Y}$ with complement $\mathcal{S}^c$, we have

$$\begin{aligned} \textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c) = &\min_{a \in \mathbb{R}} \sum_{i \in \bigcup \mathcal{S}} |i - a| \\ &+ \min_{b \in \mathbb{R}} \sum_{j \in \bigcup \mathcal{S}^c} |j - b|. \end{aligned}$$

Note that $\bigcup \mathcal{S}$ and $\bigcup \mathcal{S}^c$ partition all elements in $\bigcup \mathcal{Y}$. We may reorganize the sums *set by set* (i.e., over each $S \in \mathcal{Y}$), introducing two real parameters a and b. Gathering the terms for each S and observing that the choice

$$\min\Big(\sum_{i \in S} |i - a|, \sum_{j \in S} |j - b| \Big)$$

corresponds to placing S into $\mathcal{S}$ or its complement $\mathcal{S}^c$, respectively, yields

$$\begin{aligned} &\min_{\emptyset \subsetneq \mathcal{S} \subsetneq \mathcal{Y}} \big(\textbf{MAE}(\mathcal{S}) + \textbf{MAE}(\mathcal{S}^c) \big) \\ &\quad = \min_{a,b \in \mathbb{R}} \sum_{S \in \mathcal{Y}} \min\Big(\sum_{i \in S} |i - a|, \sum_{j \in S} |j - b| \Big). \end{aligned}$$

This confirms the claimed equivalence, completing the proof. □

After the transformation, we consider the following optimization problem.

Problem 1 (Median split problem)**.** *Given $\mathcal{Y}$, a family of subsets of $\mathbb{R}$, find $a, b \in \mathbb{R}$, such that $\sum_{S \in \mathcal{Y}} \min(\sum_{i \in S} |i - a|, \sum_{j \in S} |j - b|)$ is minimized.*

We define a few more substitutions to simplify (and at the same time, generalize) the problem even further.

Define $f_S(x) = \sum_{y \in S} |y - x|$. Observe that we try to optimize $g(a, b) = \sum_{S \in \mathcal{Y}} \min\{f_S(a), f_S(b)\}$. Note that each f_S is piecewise-linear and unimodal. Indeed, let $c = \mathbf{M}(S)$, when $x < c$, f_S is monotonically decreasing and when $x > c$, f_S is monotonically increasing. Hence, f_S is unimodal. We use this property to design a much faster algorithm. To this end, we introduce a much more general problem, the Unimodal Cost 2-Median problem (UC2M).

Algorithm 1: Binary MAE Split

Function BINARYMAESPLIT**:**
  Input: $data$
  $C \leftarrow$ list of categories of the feature
  foreach $c \in C$ **do**
    $G \leftarrow \emptyset$
    foreach $(r, c) \in data$ **do**
      add function $x \mapsto |r - x|$ to G
    end
    $f_c \leftarrow \sum_{g \in G} g$
  end
  return UNIMODAL2MEDIAN($\{ f_c \mid c \in C \}$)

Problem 2 (Unimodal Cost 2-Median (UC2M))**.** *Let $f_1, \ldots, f_k : \mathbb{R} \to \mathbb{R}$ be k piecewise-linear unimodal functions with a total of n breakpoints. Let $g(a, b) = \sum_{i=1}^{k} \min\{f_i(a), f_i(b)\}$. Find $a, b \in \mathbb{R}$ such that it minimizes g.*

Theorem 2. *Problem 1 reduces to Problem 2 in linear time.*

Proof. Let $f_S = \sum_{i \in S} |i - x|$. Let the input to the Problem 1 be $\mathcal{Y} = \{S_1, \ldots, S_k\}$. This reduces to Problem 2 with input functions $f_{S_1}, \ldots, f_{S_k}$. □

See Algorithm 1 for the entire reduction assuming inputs are data points of pairs (r, c), where r is the value and c is the category of the feature. Note for simplicity of presentation, we only compute the value instead of the actual split, but it is easy to extend it to return the entire solution.

Therefore, we shift gears and try to solve Problem 2 in the remainder of the paper.

Algorithm for Unimodal Cost 2-Median

UC2M is related to various classical 2-median problems in 1D, where there is a cost that is a function of the *distance* to the center (Hassin and Tamir 1991; Chen and Wang 2011). Those problems can be seen as a *symmetric* unimodal cost 2-median problem. A function $f : \mathbb{R} \to \mathbb{R}$ is *symmetric*, if there exists some $c \in \mathbb{R}$, such that $f(c-x) = f(c+x)$ for all $x \in \mathbb{R}$. Previous techniques do not help with our problem, as our functions are not always symmetric. Hence, we have to design the algorithm from the ground up. Note our algorithm would be a special case of the algorithm for unimodal cost facility location on a line problem (Zheng 2026), which we describe the design of the special case here.

For a piecewise-linear function f of n breakpoints, the representation consists of the sorted list of breakpoints $x_1, \ldots, x_n$, and the corresponding values $f(x_1), \ldots, f(x_n)$. Additionally, the initial slope and the final slope are also stored. Given i, one can find x_i, and evaluate f, the right slope of f, and the change of slope of f at x_i, all in $O(1)$ time. If we are interested in finding the value of $f(x)$ by giving x instead of any index, it takes $O(\log n)$ time by doing a binary search over the list and then interpolating adjacent breakpoints.

Properties of the Problem

Let $f_1, \ldots, f_k$ be the input of Problem 2, and $g(x, y) = \sum_{i=1}^{k} \min\{f_i(x), f_i(y)\}$. Evaluate g for all x, y takes $O(k)$ time each if we look at x, y in order. Hence, Problem 2 has an $O(n^2 k)$ time algorithm.

This algorithm is extremely naive, looking through all possible input pairs. One might guess that if we fix a, then the function $g_a(b) = g(a, b)$ is a unimodal function, and then a binary search-like procedure can be applied to find the minimum b for g_a. Unfortunately, this is false; g_a can have many local minima. Fortunately, g is a totally monotone function.

Theorem 3. *Let $f : \mathbb{R} \to \mathbb{R}$ be an unimodal function. The function $g : \mathbb{R}^2 \to \mathbb{R}$ defined as $g(x, y) = \min(f(x), f(y))$ if $x \leq y$, and $g(x, y) = \infty$ if $x > y$, is a totally monotone function.*

Proof. Consider for any $x_1 \leq x_2$ and $y_1 \leq y_2$.

In order to show h is totally monotone, we have to show that if $\min(f(x_1), f(y_1)) \leq \min(f(x_1), f(y_2))$, then $\min(f(x_2), f(y_1)) \leq \min(f(x_2), f(y_2))$.

If we do not have $x_1 \leq x_2 \leq y_1 \leq y_2$, we will obtain an infinity case, and one can see the inequalities hold.

Hence we assume $x_1 \leq x_2 \leq y_1 \leq y_2$.

There are two cases.

Case 1. If $f(y_1) \leq f(y_2)$, then $\min\{f(x_2), f(y_1)\} \leq \min\{f(x_2), f(y_2)\}$.

Case 2. Otherwise, assume $f(y_1) > f(y_2)$. Because $f(y_1) > f(y_2)$ but $y_1 \leq y_2$, so we must have y_1 is in the decreasing part of the function f. Therefore, we must have $f(x_1) \geq f(x_2) \geq f(y_1) > f(y_2)$. Hence, $f(y_1) = \min\{f(x_1), f(y_1)\} \leq \min\{f(x_1), f(y_2)\} = f(y_2) < f(y_1)$, a contradiction. □

By Theorem 3, and the fact that the sum of totally monotone functions is totally monotone (Park 1999), the function we try to optimize in Problem 2 is a totally monotone function. Let $x_1, \ldots, x_n$ be all the breakpoints of all the functions ordered from smallest to largest. Consider the matrix M, such that $M_{i,j} = g(x_i, x_j)$, where x_i is the ith breakpoint of g, then M is a totally monotone matrix. It is useful for us to consider the index-based version of the breakpoint instead of the breakpoint itself for ease of implementation.

For a totally monotone matrix M, the SMAWK algorithm finds the row minima of each row of M in $O(n)$ evaluations of entries in M (Aggarwal et al. 1987). Each evaluation takes $O(k \log k)$ time. Therefore, we obtain an $O(nk \log k)$ time algorithm. The SMAWK algorithm is known to use a minimum number of evaluations (up to a constant), so it seems there is no way to beat the current running time by much, as it is unlikely to do a single evaluation in less than $O(k)$ time in the worst case.

However, observe that evaluations are not all independent. After evaluating $M_{i,j}$, evaluating $M_{i+1,j}$ or $M_{i,j+1}$ would become easier, as the difference is only a single breakpoint, so the change in the function g is easy to describe. Hence, if one can arrange the order of evaluation and compute the "difference" with a better data structure, a fast algorithm can be designed. In the next section, we show this is possible by

massively speeding up the average time of each evaluation at the cost of *slightly increasing* the number of evaluations.

Slowing Down to Speed Up

Recall that we are interested in finding the x and y such that $g(x, y)$ is minimized, where x, y are from a lattice grid and $g(x, y)$ evaluated on the grid results in a totally monotone matrix. We can make the evaluation dependent on predecessors through an alternative divide-and-conquer algorithm other than the SMAWK. This new divide-and-conquer algorithm will take a total of $O(n \log n)$ evaluations of the matrix, so a *slow down* in the number of evaluations. However, a total *speed up* is obtained by speeding up each individual evaluation.

We outline the idea as follows: for a fixed i, let j be the value that minimizes $M_{i,j}$. The optimum of the entire matrix must be either $M_{i,j}$, or of the form $M_{i',j'}$ where $i' < i$ and $j' \leq j$, or $i' > i$ and $j' \geq j$ (Park 1999). Hence, this gives us a natural divide-and-conquer algorithm: find the row minimum of the center row and recursively solve the new problem on the two smaller matrices. Observe the total number of evaluations of a $n \times m$ matrix would be $T(n, m) = T(n/2, m_1) + T(n/2, m_2) + O(m) = O(m \log n)$. Since, in our case, $n = m$, we get an algorithm taking $O(n \log n)$ evaluations.

Naively, this would give us an $O(nk \log n \log k)$ time algorithm, which is even worse than the SMAWK algorithm. However, instead of the number of evaluations, we can show that the *running time* follows a similar recursion, and thus we obtain an $O(n \log n + k \log k \log n)$ time algorithm.

Naturally, we have to describe how to solve the two parts of the problem: Finding the row minimum and divide-and-conquer.

Find the Minimum over a Single Row

Finding a minimum of a given row in the matrix M, is equivalent to answer the following question: Given functions $f_1, \ldots, f_k$, and an fixed index a, how to find $\min_b \sum_{j=1}^{k} \min(f_j(x_a), f_j(x_b))$ quickly?

For a fixed a, define the *active set* at index b to be the set of function indices j, such that $f_j(x_a) > f_j(x_b)$. Let $A_1, \ldots, A_n$ be the sequence of active sets at $1, \ldots, n$, respectively. Each function f_j moves out of the active set only once. That is, for an index $j \in [k]$, there exists an index q_j, such that for each $i \geq q_j$, we have $j \in A_i$, and $j \notin A_i$ otherwise.

Define $f_A = \sum_{i \in A} f_i$. If we can quickly evaluate f_A and update A, then we can quickly evaluate g. Indeed, in order to evaluate $g(x_a, x_b)$, the idea is to break it down into evaluating $f_{\bar{A}}(x_a) + f_A(x_b)$ where A is the active set at index b.

It's not hard to describe a data structure that maintains the value of $f_A(x_a)$ under updates of A and a; this is precisely the evaluation data structure in Theorem 6. However, we must also decide which function has to be added or removed from A. This is done through a useful transformation. Let f be an unimodal function with the local minimum at c, define $f^\dagger : \mathbb{R} \to \mathbb{R}$ to be $f^\dagger(x) = \max\{x' \mid f(x') \leq f(x)\}$ if $x \in (-\infty, c]$ and $-\infty$ otherwise. Intuitively, this means for any $x \leq c$, if we have $x \leq x' \leq f^\dagger(x)$, then we know $f(x') \leq f(x)$. Namely, one can quickly observe if $f(x) \leq f(x')$ by checking if $x' \leq f^\dagger(x)$. See Figure 2.

If f is a piecewise-linear unimodal function of n breakpoints, then $f^\dagger$ is a piecewise-linear decreasing function in $(-\infty, c]$ of n breakpoints and can be computed from f in $O(n)$ time. We can find the value of $f^\dagger(x)$ in $O(\log n)$ time. Since $f^\dagger$ can be computed in linear time when f is created, we always assume that $f^\dagger$ is computed when we use f.

Knowing $f_i^\dagger(x_a)$ for each i, then we know precisely when i moves out of the active set: i moves out of the active set when $x_b > f_i^\dagger(x_a)$ for the first time. See Algorithm 2 for implementation of RowMinima.

Theorem 4. *If the input is k functions with a total of n breakpoints, row minima can be found in $O(n + k \log k)$ time.*

Proof. We analyze the running time of RowMinima. Ordering the functions by evaluation of $f_j^\dagger(x_a)$ for each j, which takes $\sum_{j=1}^{k} n_j \log n_j = O(k \log \frac{n}{k})$ time, where n_j is the number of breakpoints of f_j. Sorting the k functions by their $\dagger$ value takes $O(k \log k)$ time. The linear scan takes $O(n)$ time. Hence, going through each breakpoint takes $O(n)$ time. Note $k = O(n)$, hence $O(k \log \frac{n}{k}) = O(n)$. The total running time of RowMinima is $O(n + k \log k)$. □

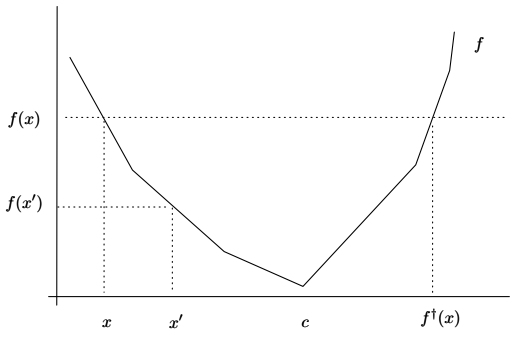

Figure 2: Intuition of the † transform.

Divide-and-Conquer

In the divide-and-conquer step, we split the problem into evaluations over 2 smaller submatrices, which are almost disjoint.

Observe that once we are searching for the optimum in a submatrix where the row index ranges from a_{min} to a_{max} and the column index ranges from b_{min} to b_{max}. All the values of the functions outside this range are irrelevant. Hence, we can safely assume we process all the functions passed into the recursive call by removing the breakpoints outside the ranges. In practice, this is not explicit but is done through implicit bookkeeping. This allows us to bound the running time related to traversing the functions by $O((b_{max} - b_{min}) + (a_{max} - a_{min}))$ inside each recursive call. See the recursive algorithm Optimum in Algorithm 2.

Dataset	Feature	Data size	Categories	LightGBM relative Accuracy	LightGBM Time (s)	Our Time (s)
Predict Droughts	TS	19,300,680	7,588	0.9957	5.40698	0.948483
	WS10M	19,300,680	1,740	0.9991	5.64445	0.737965
	QV2M	19,300,680	2,210	0.9734	5.13713	0.843196
	T2M_RANGE	19,300,680	3,029	0.9729	5.40553	0.847019
delays_zurich_transport	windspeed_avg	5,465,575	66	0.9996	1.25839	0.283073
	temp	5,465,575	143	0.9984	1.26260	0.289960
	stop_id	5,465,575	1,530	0.9766	1.29020	0.370960
	time	5,465,575	3,526	0.9870	1.23520	0.402581
gpu_kernel_performance	MWG	5,100,000	5	0.9234	2.25632	0.598805
	MDIMC	5,100,000	7	0.9903	2.33057	0.271983
	NWG	5,100,000	6	0.9583	2.36985	0.465714
diamonds	carat	53,940	273	0.6193	0.021328	0.016229
	table	53,940	127	0.9829	0.022840	0.008226
	x	53,940	554	0.6213	0.028240	0.019635
house_sales	sqft_living	21,613	1,038	0.8553	0.026370	0.009103
	zipcode	21,613	70	0.8322	0.016795	0.005777
	sqft_above	21,613	946	0.8932	0.036526	0.008170
wine	fixed acidity	1,143	91	0.8662	0.012946	0.000178
	density	1,143	388	0.7094	0.010698	0.000281
	volatile acidity	1,143	135	0.8067	0.005512	0.000193
	citric acid	1,143	77	0.8611	0.004860	0.000177
boston	ZN	506	26	0.8926	0.005560	0.000262
	INDUS	506	76	0.7886	0.005856	0.000444
	DIS	506	412	0.7320	0.018420	0.000626

Table 2: Comparison between LightGBM and our algorithm.

The main observation is that the sequence of functions is also split into two subproblems. Let $M_{a,b}$ be the optimum value on the row a. We consider the functions in two classes, $L = \{i | f_i(x_a) \leq f_i(x_b)\}$, and $R = \{i | f_i(x_a) > f_i(x_b)\}$. The algorithm would be correct if we pass down all functions. However, when we pass the function to the left recursion, during the evaluating values $M_{a',b'}$ where $a' \in [a_{min}, a]$ and $b' \in [b_{\min}, b]$, the contribution of the function f_i where $i \in R$ is apparent: $\min(f_i(x_{a'}), f_i(x_{b'})) = f_i(x_{b'})$. Namely, $\sum_{i \in R} \min(f_i(x_{a'}), f_i(x_{b'})) = \sum_{i \in R} f_i(x_{b'})$. A similar result holds for right recursion. Therefore, there is no need to pass down all the functions; instead, we sum the functions and pass them down. This ensures the sequence of functions passed down is partitioned into two, each with one more function appended.

Theorem 5. *Finding the optimum of Problem 2 for the input of k function of a total n breakpoints takes $O((n + k \log k) \log n)$ time.*

Proof. See appendix. □

Some additional optimization can be done that does not change the asymptotic worst-case running time but improves the running time in practice. For example, in the recursion, if at some point, the only function remaining is the function that came from a sum of original functions, then the recursion can stop earlier.

Data Structure for Piecewise-Linear Functions

We describe a data structure over a set of piecewise-linear functions, and it can return the sum quickly. This data structure is required to implement RowMinima, where two dynamic sums of piecewise-linear functions need to be maintained, and also Optimum, where the sum of piecewise-linear functions has to be computed and passed down.

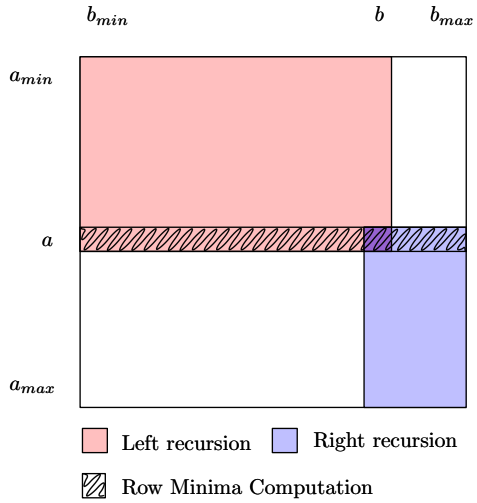

Figure 3: A demonstration of one step of the recursion algorithm.

Let $f_1, \ldots, f_k$ be piecewise-linear functions with a total of n distinct breakpoints. Although our algorithms can handle the case when breakpoints are not distinct, describing them does not provide additional insights.

Let $f_A = \sum_{i \in A} f_i$. Let a global set of points $x_1, \ldots, x_n$ contain all the breakpoints of each f_i. We aim to maintain a data structure that includes a set A and an index a, enabling the fast evaluation of $f_A(x_a)$.

Formally, the data structure should have the following op-

Algorithm 2: Finding Unimodal 2 Medians

Function UNIMODAL2MEDIAN**:**
  Input: f
  compute global list of breakpoints $x_1, \ldots, x_n$
  compute the † transform for each function
  return OPTIMUM$(1, n, 1, n, f)$

Function OPTIMUM**:**
  Input: $a_{\min}, a_{\max}, b_{\min}, b_{\max}, f$
  $a \leftarrow (a_{\min} + a_{\max})/2$
  $(v, b) \leftarrow$ ROWMINIMA$(a, b_{\min}, b_{\max}, f)$
  if $a_{\min} = a_{\max}$ **then**
    return v
  end
  $L \leftarrow \{\, i \mid f_i(x_a) \le f_i(x_b) \,\}$
  $R \leftarrow \{\, i \mid f_i(x_a) > f_i(x_b) \,\}$
  $f^L \leftarrow \{f_i \mid i \in L\} \cup \left(\sum_{i \in R} f_i\right)$
  $f^R \leftarrow \{f_i \mid i \in R\} \cup \left(\sum_{i \in L} f_i\right)$
  $v_L \leftarrow$ OPTIMUM$(a_{\min}, a, b_{\min}, b, f^L)$
  $v_R \leftarrow$ OPTIMUM$(a, a_{\max}, b, b_{\max}, f^R)$
  return $\min(v_L, v_R, v)$

Function ROWMINIMA**:**
  Input: $a, b_{\min}, b_{\max}, f$
  renumber $f_1, \ldots, f_k$ so that $f_j^\dagger(x_a) \le f_{j+1}^\dagger(x_a)$
  $p \leftarrow 1$ $A \leftarrow \{1, \ldots, k\}$
  for $i = b_{\min}$ **to** $b_{\max}$ **do**
    while $f_p^\dagger(x_a) < x_i$ **do**
      $A \leftarrow A \setminus \{p\}$ $p \leftarrow p + 1$
    end
    $v \leftarrow f_{\bar{A}}(x_a) + f_A(x_i)$
    if *v is the smallest seen value* **then**
      $b \leftarrow i$
    end
  end
  return b

erations.

1. INITIALIZE$(f_1, \ldots, f_k)$: Process the functions $f_1, \ldots, f_k$, and return a data structure for $f_\emptyset$ and $a = -1$.
2. ADD(i): Update A into $A \cup \{i\}$.
3. REMOVE(i): Update A into $A \setminus \{i\}$.
4. EVALUATE(): Return $f_A(x_a)$.
5. NEXT(): Update a to $a + 1$.

Such a data structure is standard, but we sketch it here for completeness.

Theorem 6. *Assuming all the breakpoints have been sorted, the data structure takes $O(n)$ time to construct, and any sequence of $O(n)$ queries takes $O(n)$ time.*

Proof. See appendix. □

Theorem 7. *Problem 2 can be solved in $O((n + k \log k) \log n)$ time.*

Theorem 8. *The median split problem on k categories and n data points can be solved in $O((n + k \log k) \log n)$ time.*

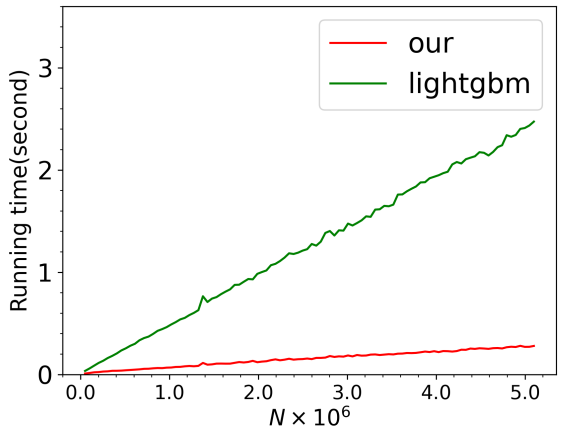

Figure 4: Running time in seconds vs. number of data points.

In particular, if the number of categories is small with respect to the number of datapoints, then our running time is simply $O(n \log n)$.

Experiments

We implemented the algorithm in C++ and ran it on an 8-core AMD Ryzen 7 5800H processor with 16GB of RAM.[1] We selected several regression datasets from OpenML: two large datasets (more than 500k data points) (OpenML 2017, 2023), two medium-sized datasets (around 50k data points) (OpenML 2019, 2020), and two small datasets (about 1000 data points) (Kaggle 2022; OpenML 2014). For each dataset, we attempted a binary split on each of its features. We used LightGBM, as it can handle large-scale data inputs, whereas scikit-learn often fails to complete in a reasonable amount of time.

We recorded the running time and MAE values of our algorithm and LightGBM, and calculated the relative accuracy (our result / LightGBM result) for each dataset, as shown in the table above (OpenML dataset IDs in parentheses).

To further evaluate performance under imbalanced data, we selected two datasets from Kaggle: Predict Droughts(Minixhofer 2021) and Wine. Their results are also included in the table.

Our algorithm achieves exact MAE-optimal splits in all cases. Interestingly, LightGBM is sufficiently accurate in most instances. However, since our algorithm is consistently faster, there is no need to use subsampled data to reduce computation. For the imbalanced datasets, our method maintains higher accuracy than LightGBM and also demonstrates superior time efficiency. Furthermore, since our evaluation focuses only on single-level binary splits, we have reason to believe that, in multi-level decision trees, the cumulative error from heuristic methods like LightGBM may grow significantly as the depth increases.

[1]Code is available at https://anonymous.4open.science/r/binary-split-F12B.

We also examined running time asymptotics by creating subsets of the `gpu_performance` dataset with feature `MDIMC` via sampling without replacement from 0.01 to 1.0 in 0.01 increments. We ran both our algorithm and LightGBM on each subset and compared the runtime. The result in Figure 4 confirms that our method scales nearly linearly with input size.

References

Aggarwal, A.; Klawe, M. M.; Moran, S.; Shor, P.; and Wilber, R. 1987. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1): 195–208.

Breiman, L.; Friedman, J.; Stone, C.; and Olshen, R. 1984. Classification and Regression Trees.

Catlett, J. 1991. Mega induction: a Test Flight. In Birnbaum, L. A.; and Collins, G. C., eds., *Machine Learning Proceedings 1991*, 596–599. San Francisco (CA): Morgan Kaufmann. ISBN 978-1-55860-200-7.

Chen, D. Z.; and Wang, H. 2011. New Algorithms for 1-D Facility Location and Path Equipartition Problems. In Dehne, F.; Iacono, J.; and Sack, J.-R., eds., *Algorithms and Data Structures*, 207–218. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-22300-6.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794. ACM.

Hassin, R.; and Tamir, A. 1991. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10(7): 395–402.

Hastie, T.; Tibshirani, R.; and Friedman, J. 2001. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

jiangfeng. 2017. Trees with MAE criterion are slow to train. https://github.com/scikit-learn/scikit-learn/issues/9626. Accessed: 2024-12-30.

Kaggle. 2022. Wine Quality Dataset. https://www.kaggle.com/datasets/yasserh/wine-quality-dataset. Accessed: 2025-07-26.

Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; and Liu, T.-Y. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances in Neural Information Processing Systems 30*, 3146–3154.

McGinnis, W.; hbghhy; Tao, W.; andrethrill; Siu, C.; Davison, C.; and Bollweg, N. 2018. scikit-learn-contrib/categorical-encoding: Release for zenodo.

Minixhofer, C. 2021. Predict Droughts using Weather & Soil Data. https://www.kaggle.com/datasets/cdminix/us-drought-meteorological-data. Accessed: 2025-07-26.

OpenML. 2014. boston dataset (v.1). https://www.openml.org/d/531. Accessed: 2025-1-7.

OpenML. 2017. delays zurich transport dataset (v.1). https://www.openml.org/d/40753. Accessed: 2025-1-9.

OpenML. 2019. diamonds dataset (v.1). https://www.openml.org/d/42225. Accessed: 2025-1-8.

OpenML. 2020. house sales dataset (v.3). https://www.openml.org/d/42731. Accessed: 2025-1-9.

OpenML. 2023. simulated sgemm gpu kernel performance dataset (v.2). https://www.openml.org/d/45662. Accessed: 2025-1-9.

Park, J. K. 1999. *The Monge Array: An Abstraction and Its Applications*. Ph.D. thesis, Massachusetts Institute of Technology.

Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A. V.; and Gulin, A. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems*, 6638–6648.

Torgo, L. F. R. A. 1999. *Inductive learning of tree-based regression models*. Ph.D. thesis, Universidade do Porto. Reitoria.

Zheng, D. W. 2026. Unimodal Cost Facility Location on a Line. Unpublished manuscript.