Very Fast, Approximate Counterfactual Explanations for Decision Forests

Miguel Á. Carreira-Perpiñán, Suryabhan Singh Hada

Dept. Computer Science & Engineering, University of California, Merced
{mcarreira-perpinan, shada}@ucmerced.edu

Abstract

We consider finding a counterfactual explanation for a classification or regression forest, such as a random forest. This requires solving an optimization problem to find the closest input instance to a given instance for which the forest outputs a desired value. Finding an exact solution has a cost that is exponential on the number of leaves in the forest. We propose a simple but very effective approach: we constrain the optimization to only those input space regions defined by the forest that are populated by actual data points. The problem reduces to a form of nearest-neighbor search using a certain distance on a certain dataset. This has two advantages: first, the solution can be found very quickly, scaling to large forests and high-dimensional data, and enabling interactive use. Second, the solution found is more likely to be realistic in that it is guided towards high-density areas of input space.

1 Introduction

A counterfactual explanation (CE) seeks the minimum change to an input instance that will result in a desired outcome under a given predictive model. For example, "reducing your weight by 10 kg will reduce your risk of stroke by 80%" (regression) or "you will be eligible for the loan if you increase your annual salary by $10k" (classification). CEs extend the use of a machine learning model beyond just prediction to querying about potential scenarios. This is especially relevant in applications where interpretability or explainability is important, such as in financial, legal, human resources, government or health models. It can also make it possible to audit a model to find errors or bias, and to have an objective measure of the importance of the input features. CEs are also formally equivalent to adversarial attacks, but the latter have a different motivation: they seek to trick a model into making the wrong prediction by making imperceptible changes to the input.

CEs can be naturally formulated as an optimization problem over the input instance of the form "minimize the distance to a source instance subject to the model predicting a desired outcome". Here, we consider as model an ensemble of decision trees (a decision forest). We consider both axis-aligned trees, which are widely used in Random Forest (Breiman 2001), AdaBoost (Freund and Schapire 1997) and Gradient Boosting (Friedman 2001), but also oblique trees, which achieve state-of-the-art accuracy using fewer and shallower trees (Zharmagambetov and Carreira-Perpiñán 2020; Gabidolla and Carreira-Perpiñán 2022; Carreira-Perpiñán, Gabidolla, and Zharmagambetov 2023). The optimization problem is difficult because forests define a piecewise constant predictive function, so gradients are not applicable. The number of constant-value regions is exponential on the size of the forest (number of leaves and number of trees), so exhaustive search approaches (even making use of clever pruning and engineering heuristics) will not be able to scale to real-world forests, for which the number of leaves per tree and the number of trees each run into hundreds or thousands.

We propose a simple but effective approach: to limit the search to the set of regions containing actual (training) data points. This makes the search extremely fast, producing a good, feasible CE estimate in less than a second (for axis-aligned forests) or a few seconds (for oblique forests) even for the largest problems we experimented with. A secondary advantage is that it tends to produce realistic CEs, since the live regions can be seen as a nonparametric density estimate built in the forest. In section 3 we study the geometry of the forest predictive function and the number of nonempty and live regions. In sections 4–5 we give our algorithm (LIRE) and evaluate it in section 6.

2 Related Work

Much of the work about counterfactual explanations, particularly in the guise of adversarial attacks, has focused on deep neural nets (Szegedy et al. 2014; Goodfellow, Shlens, and Szegedy 2015). The optimization here is relatively easy because gradients of the model are available (at least approximately, with ReLU activations). That said, with a heavily nonlinear function such as a neural net, the optimization problem may have multiple local minima, some of them not even feasible (i.e., failing to produce the desired prediction). This makes finding a good solution difficult. Some work is agnostic to the model (Karimi et al. 2020; Guidotti et al. 2019; White and d'Avila Garcez 2020), requiring only function evaluations (and possibly constructing a mimic of the model), but this restricts its performance severely in terms of computational efficiency and quality or feasibility of the result.

For decision trees, whether axis-aligned or oblique, the problem can be solved exactly and efficiently (Carreira-Perpiñán and Hada 2021a,b; Hada and Carreira-Perpiñán 2021) by finding an optimal CE within each leaf's region and picking the closest one. This scales nicely because an individual tree, particularly an oblique tree, will rarely have more than several hundred leaves. For a decision forest, the problem is NP-hard. This was shown for a special case (a binary classification axis-aligned tree with discrete features) by reduction from a maximum coverage problem (Yang et al. 2006), and for a more general case by reduction from a DNF-MAXSAT problem (Cui et al. 2015).

Some works use heuristics to solve the CE for forests. Lucic et al. (2022) use a gradient-based algorithm that approximates the splits of the decision trees with sigmoid functions. Tolomei et al. (2017) propose an approximate algorithm based on propagating the source instance down each tree towards a leaf. As seen in our experiments, these approaches are slow for large forests and often fail to find a feasible solution as the number of features or trees grows.

Several approaches (Cui et al. 2015; Kanamori et al. 2007; Parmentier and Vidal 2021) are based on encoding, more or less efficiently, the CE problem as a mixed-integer program and then solving this using an existing solver (typically based on branch-and-bound). This capitalizes on the significant advances that highly-optimized commercial solvers have incorporated in the last decades (which, unfortunately, are not yet available to free or open-source solvers). This is guaranteed to find the global optimum but only if the solver terminates its essentially exhaustive search. Even with highly efficient solvers, the optimistic claims in some of these papers about scalability to large datasets and forests do not hold up, as seen in our experiments.

Finally, several approaches (not necessarily for forests) seek to generate CEs that are more plausible or realistic (Russell 2019; Ustun, Spangher, and Liu 2019; Karimi et al. 2020; Kanamori et al. 2007; Mothilal, Sharma, and Tan 2020; Van Looveren and Klaise 2021; Parmentier and Vidal 2021). They do this by adding distances to a set of training instances as a penalty to the cost function to encourage the solution to be close to the training data.

3 Counterfactual Explanation Problem: Definition, Geometry and Complexity

We define the counterfactual explanation (CE) problem as the following optimization:

$$\min_{\mathbf{x}\in\mathbb{R}^D} d(\mathbf{x},\overline{\mathbf{x}}) \quad \text{s.t.} \quad F(\mathbf{x}) \in \mathcal{S}. \tag{1}$$

Here, $\overline{\mathbf{x}} \in \mathbb{R}^D$ is the *source instance* (whose prediction under F we wish to change), $\mathbf{x} \in \mathbb{R}^D$ is the *solution instance*, and $d(\cdot,\cdot)$ is a *distance* in input (feature) space, which measures the cost of changing each feature. We will focus primarily on the ℓ_2^2 or ℓ_1 distances; weighted distances can be handled by appropriately rescaling the features. F is the *predictive function* of the model (a decision forest in our case), which maps an input instance $\mathbf{x}$ to either a value in $\{1,\dots,K\}$ for multiclass classification, or to a real value for regression (we can also include classification here if the forest output is a class probability). Finally, $\mathcal{S}$ is a set of target predictions, i.e., we want $\mathbf{x}$'s prediction to be a value in $\mathcal{S}$. For example, for classification $\mathcal{S}$ can be a specific class (or a subset of classes) in $\{1,\dots,K\}$; for regression, $\mathcal{S}$ can be an interval (e.g. $F(\mathbf{x}) \geq 7$) or a set of intervals. We may also have constraints on $\mathbf{x}$.

A decision forest is an ensemble of decision trees. We consider two types of trees: axis-aligned, where each decision node i has the form "$x_{d(i)} \geq \theta_i$" for some feature $d(i) \in \{1,\dots,D\}$ and bias $\theta_i \in \mathbb{R}$; and oblique trees, where each decision node has the form "$\mathbf{w}_i^T\mathbf{x} \geq w_{i0}$" for some weight vector $\mathbf{w}_i \in \mathbb{R}^D$ and bias $w_{i0} \in \mathbb{R}$. In both cases, each leaf of a tree outputs a constant value (class label in $\{1,\dots,K\}$ or value in $\mathbb{R}$). A forest of T trees computes its prediction $F(\mathbf{x})$ by finding the leaf l_t that $\mathbf{x}$ reaches in each tree $t \in \{1,\dots,T\}$ and applying a function to the leaf outputs (usually the majority vote for discrete labels or the average for real values). The forest is trained on a dataset using an algorithm to learn individual trees (such as CART, C4.5 or any of its variations) and an ensemble mechanism (bagging and random feature subsets in Random Forests, reweighted training set in AdaBoost, residual error fitting in Gradient Boosting, etc.) (Hastie, Tibshirani, and Friedman 2009). Although the vast majority of work on forests uses axis-aligned trees, here we also consider forests of oblique trees. These can be learned with any ensemble mechanism using as base learner the Tree Alternating Optimization (TAO) algorithm (Carreira-Perpiñán and Tavallali 2018;

ENUMERATING ALL NONEMPTY REGIONS

input forest F with T trees
$I_1 \leftarrow$ all-ones vector of dimension L_1
for $t = 2,\dots,T$
  $I_t \leftarrow$ all-zeros sparse array of $L_1 \times \cdots \times L_t$
  for each $(l_1,\dots,l_{t-1})$ with $I_{t-1}(l_1,\dots,l_{t-1}) = 1$
    for $l_t = 1,\dots,L_t$
      if region$(l_1,\dots,l_{t-1}) \cap$ region$(l_t) \neq \varnothing$ **then**
        $I_t(l_1,\dots,l_{t-1},l_t) \leftarrow 1$
  remove I_{t-1} from memory
return I_T

ENUMERATING ALL LIVE REGIONS

input forest F with T trees, dataset $\mathbf{X}_{D\times N}$
$I,\ Y \leftarrow$ all-zeros sparse array of $L_1 \times \cdots \times L_T$
for $n = 1,\dots,N$
  for $t = 1,\dots,T$
    $l_t \leftarrow$ leaf reached by $\mathbf{x}_n$ in tree t
  if $I(l_1,\dots,l_T) = 0$ **then**
    $I(l_1,\dots,l_T) \leftarrow 1$
    $Y(l_1,\dots,l_T) \leftarrow F(l_1,\dots,l_T)$
return I and Y, both sorted by Y value

Figure 1: Pseudocode for finding all nonempty regions (top) and all live regions (bottom), valid for both axis-aligned and oblique trees. We omit the construction of the arrays $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{R}$ (needed in fig. 4). *Top*: region$(l_1,\dots,l_t) \equiv \cap_{i=1}^t$region$(l_i)$ and region(l_i) is the input space region of leaf l_i in tree i (defined by the intersection of the decision node hyperplanes along the path from the root to leaf l_i). $F(l_1,\dots,l_T)$ means the forest output for region$(l_1,\dots,l_t)$.

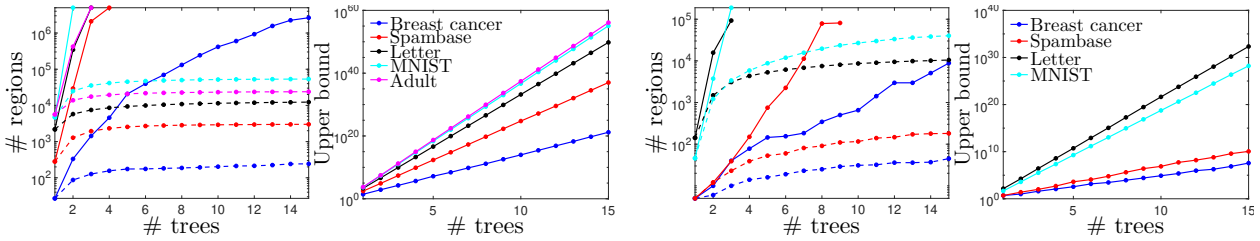

Figure 2: Growth of the number of regions of a forest as a function of the number of trees T, for different datasets, for axis-aligned trees (left 2 panels) and oblique trees (right 2 panels). Within each pair of panels, on the left panel we plot the number of nonempty regions (solid lines) and live regions (dashed lines); on the right panel, the upper bounds for the number of nonempty regions. All regions are capped to a maximum of $5 \cdot 10^6$ (axis-aligned) and 10^6 (oblique). The axis-aligned forests use fully-grown trees with an average depth of 8.9, 33.5, 27.6, 35.1 and 48.9, for Breast cancer, Spambase, Letter, MNIST and Adult, respectively. The oblique forests have a fixed depth of 8.

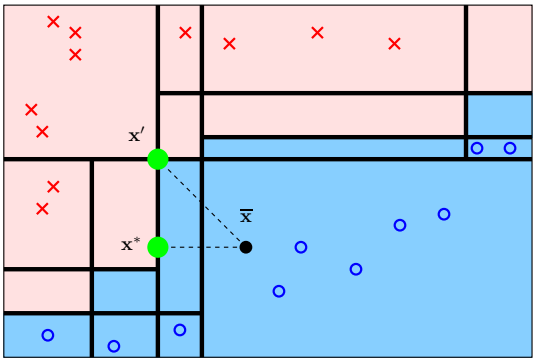

Figure 3: Illustration of the LIRE idea. We show the regions defined by a simulated forest of $T = 3$ trees of depth $\Delta = 2$, colored accordingly to the class label they predict. The live regions are those containing at least one data point. For the source instance $\overline{\mathbf{x}}$, the optimal counterfactual explanation is $\mathbf{x}^*$ (searching over all regions) and the approximate one with LIRE is $\mathbf{x}'$ (searching only over the live regions).

Carreira-Perpiñán 2022; Zharmagambetov et al. 2021), and have been recently shown to outperform axis-aligned forests in accuracy while resulting in forests having fewer and shallower trees (Carreira-Perpiñán and Zharmagambetov 2020; Zharmagambetov and Carreira-Perpiñán 2020; Gabidolla and Carreira-Perpiñán 2022; Gabidolla, Zharmagambetov, and Carreira-Perpiñán 2022; Carreira-Perpiñán, Gabidolla, and Zharmagambetov 2023). This is important here because, as shown later, oblique forests need to search far fewer regions. That said, the details of how a forest was constructed are irrelevant here. All we need is to be able to apply the forest to an input to compute two things: which leaf it reaches in each tree, and the forest output.

Geometry of the forest predictive function F A single tree with L leaves partitions $\mathbb{R}^D$ into L regions, since an input $\mathbf{x}$ reaches exactly one leaf, and each region outputs a constant value. For an axis-aligned tree, each region is a box and can be put in the form $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$ elementwise, where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^D$ contain the lower and upper bounds (including $\pm\infty$), respectively; this can be obtained from the (feature,bias) pairs in the decision nodes in the path from the root to the leaf. For an oblique tree, each region is a convex polytope bounded by the hyperplanes at the decision nodes in the root-leaf path.

SEARCH FOR CLOSEST LIVE REGION (AXIS-ALIGNED TREES)

input $\mathbf{A}_{D\times M}, \mathbf{B}_{D\times M}, \overline{\mathbf{x}} \in \mathbb{R}^D$
$\delta \leftarrow \infty$
for $n = 1, \dots, M$
  $\alpha \leftarrow d_{\text{box}}(\overline{\mathbf{x}}, \binom{\mathbf{a}_n}{\mathbf{b}_n})$
  if $\alpha < \delta$ **then**
    $i \leftarrow n$
    $\delta \leftarrow \alpha$
$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x}} d(\mathbf{x}, \overline{\mathbf{x}})$ s.t. $\mathbf{a}_i \leq \mathbf{x} \leq \mathbf{b}_i$
  $= \text{median}(\mathbf{a}_i, \mathbf{b}_i, \overline{\mathbf{x}})$
return i, $\mathbf{x}^*$, $d(\mathbf{x}^*, \overline{\mathbf{x}})$

SEARCH FOR CLOSEST LIVE REGION (OBLIQUE TREES)

input forest of T trees, $\mathbf{R}_{T\times M}$
$\delta \leftarrow \infty$
for $n = 1, \dots, M$
  $\alpha \leftarrow \min_{\mathbf{x}} d(\mathbf{x}, \overline{\mathbf{x}})$ s.t. constraints for $R(\cdot, n)$
  if $\alpha < \delta$ **then**
    $i \leftarrow n$
    $\delta \leftarrow \alpha$
$\mathbf{x}^* \leftarrow \arg\min_{\mathbf{x}} d(\mathbf{x}, \overline{\mathbf{x}})$ s.t. constraints for $R(\cdot, i)$
return i, $\mathbf{x}^*$, $d(\mathbf{x}^*, \overline{\mathbf{x}})$

Figure 4: Pseudocode for the search for the closest live region to a source instance $\overline{\mathbf{x}}$ for axis-aligned (top) and oblique trees (bottom). We assume there are M target regions which have been preselected into the lower/upper bound arrays $\mathbf{A}$ and $\mathbf{B}$ (for axis-aligned trees) or the array $\mathbf{R}$ (for oblique trees). $R(t, n)$ contains the index of the leaf in tree t that participates in region n and $R(\cdot, n)$ stands for $\{R(1, n), \dots, R(T, n)\}$. $d_{\text{box}}(\overline{\mathbf{x}}, \binom{\mathbf{a}_i}{\mathbf{b}_i}) = d(\overline{\mathbf{x}}, \text{median}(\mathbf{a}_i, \mathbf{b}_i, \overline{\mathbf{x}}))$ represents the distance-to-a-box.

A forest with T trees (where tree t has L_t leaves) partitions $\mathbb{R}^D$ into at most $L_1 L_2 \cdots L_T$ regions (L^T if each tree has L leaves), since an input $\mathbf{x}$ reaches exactly one leaf in each tree. We can encode each region as a tuple $(l_1, \dots, l_T)$ indicating the leaf reached in each tree. Hence, each region is the intersection of exactly T leaf regions, and it is a box for axis-aligned trees and a convex polytope for oblique trees.

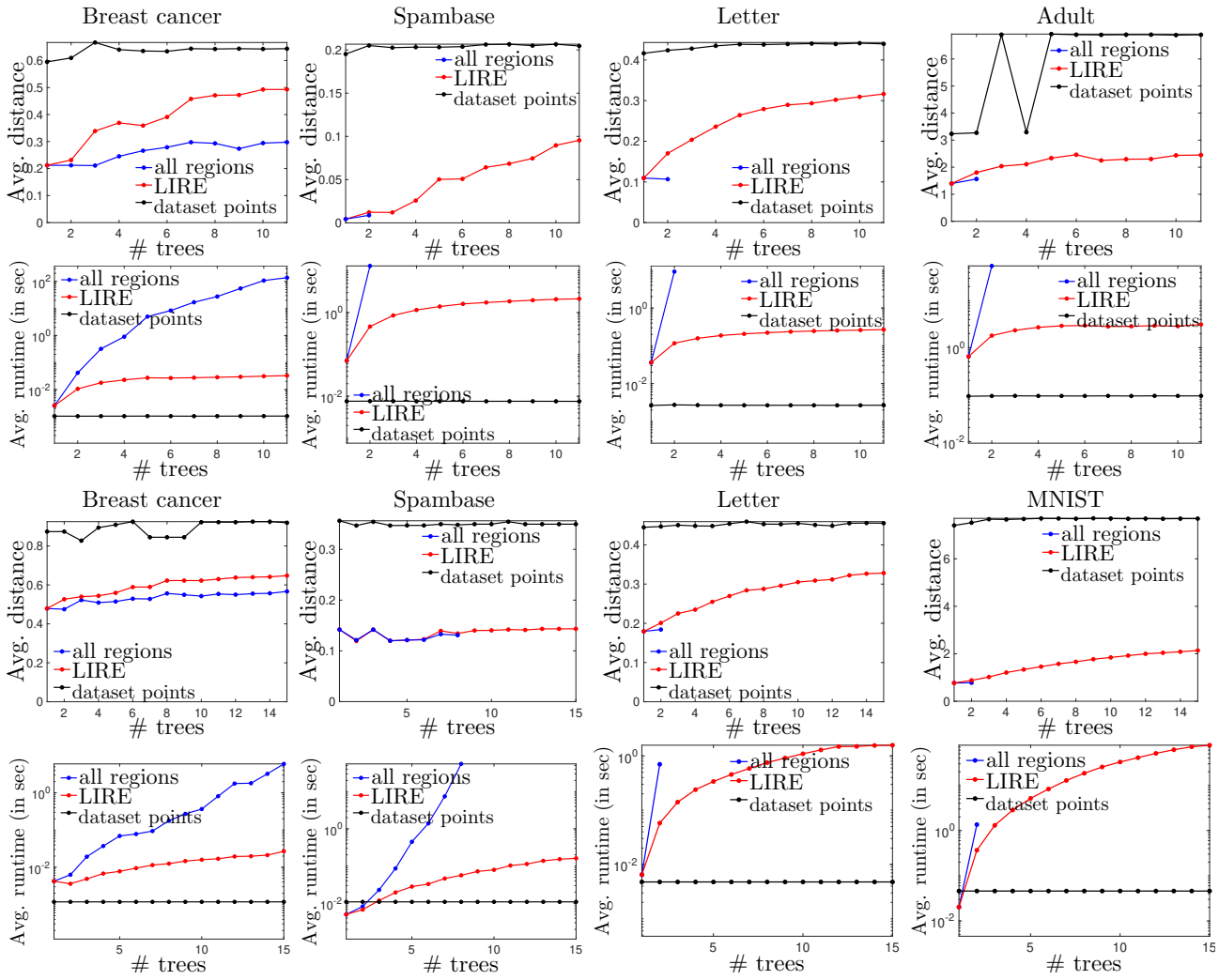

Figure 5: Performance of three types of search: on all nonempty regions, on the live regions (LIRE) and on the dataset points, as a function of the number of trees T, for different datasets, for axis-aligned trees (top 2 rows) and oblique trees (bottom 2 rows). We show the ℓ_2 distance of the CE found ($\|\mathbf{x}^* - \overline{\mathbf{x}}\|_2$) and the runtime (seconds) to solve the CE problem. The curves are the average for 5 source instances.

In each region, the forest output is constant, so the forest predictive function F is piecewise constant. Although many tuples $(l_1, \ldots, l_T)$ result in empty intersections, the number of (nonempty) regions is still exponential in general.

Number of regions in F in practice The fact that F is piecewise constant means that problem (1) can be solved exactly by enumerating all regions that satisfy the constraint $F(\mathbf{x}) \in \mathcal{S}$, finding the CE in each region[1], and returning the one with lowest distance to $\overline{\mathbf{x}}$. This was the approach in (Carreira-Perpiñán and Hada 2021a; Hada and Carreira-Perpiñán 2021) for single tree models, where it works very well because the number of leaves is always relatively small. But, how many nonempty regions can we expect with a forest, and how far is that from the upper bound L^T? It is difficult to answer this in general for practical forests, which are the result of a complex optimization algorithm, so we estimate this empirically. We can enumerate all nonempty regions with the constructive algorithm of fig. 1 (top). This proceeds sequentially to construct a sparse t-way tensor I_t, where $I_t(l_1, \ldots, l_t) = 1$ if tuple $(l_1, \ldots, l_t)$ defines a nonempty intersection and 0 otherwise. I_t is constructed by intersecting every nonempty region in I_{t-1} with every leaf in tree t. Its correctness relies on the fact that if $I_{t-1}(l_1, \ldots, l_{t-1}) = 0$ then $I_t(l_1, \ldots, l_{t-1}, l_t) = 0$ for any leaf l_t. The final tensor I_T has $L_1 \cdots L_T$ entries, but only those for which $I_T = 1$ are nonempty. The number of regions in I_t grows monotonically with t because, if $I_{t-1}(l_1, \ldots, l_{t-1}) = 1$ then $I_t(l_1, \ldots, l_t) = 1$ for at least one leaf l_t in tree t (since the leaves of each tree form a partition of the input space).

[1]This requires minimizing $d(\mathbf{x}, \overline{\mathbf{x}})$ over the region. As shown in (Carreira-Perpiñán and Hada 2021a), for axis-aligned trees, this is a box, and the exact solution is given, separately along each dimension, by the median of 3 numbers: the lower and upper bound of the box, and $\overline{x}_d$. For oblique trees, the region is a polytope, and the exact solution results from solving a quadratic program (QP) for the ℓ_2 distance and a linear program (LP) for the ℓ_1 distance.

Fig. 2 shows the results for Random Forests (Breiman 2001) on several small datasets, for which it is computationally feasible to count the regions. The actual number of regions depends in a complex way on the dataset (size and dimensionality) and type of forest. For axis-aligned forests, the number of nonempty regions, while far smaller than the upper bound, does grow exponentially quickly and exceeds a million for just a handful of trees. For oblique forests, the growth is significantly slower but still exponential. This shows than an exhaustive search, even with clever speedups, will be intractable unless the forest is impractically small.

4 An Approximation: Search Only over the "Live" Regions

Instead of considering all regions, we restrict the search to only those regions containing at least one actual data point (from the training, validation and test datasets used to train the forest). We call these *live* regions, and call the procedure LIRE (for LIve REgion search). LIRE results in an approximate but fast search and, intuitively, retrieves realistic CEs, as described next. Fig. 3 illustrates the idea. Let the dataset have N points and the number of live regions be $M \leq N$.

Faster computation The number of regions reduces from exponential to at most N, so the search is far faster and also has a very predictable runtime (unlike, for example, approaches based on mixed-integer programming, which have a wildly variable runtime). The number of live regions M is at most N because multiple points may belong to the same region (this is particularly so with oblique forests). A second reduction in the number of regions to search is due to the constraint $F(\mathbf{x}) \in \mathcal{S}$ (for example, we may need to search only on regions of one target class).

Fig. 2 shows the number of live regions. The growth behavior is very different from that of the nonempty regions, because it is upper bounded by N. For axis-aligned forests, the number of live regions reaches N with just a handful of trees, so we can expect about N regions with any practical forest (with nearly every region containing just one instance). For oblique forests, it takes well over 10 trees to approach N regions, so practical oblique forests (which do not require as many trees as axis-aligned forests) may have quite less than N regions, particularly with large datasets.

As expected, the resulting runtime of LIRE is very small (see experiments). For axis-aligned trees it takes less than 1 second in even our largest experiments; in this case, the search reduces to a special type of nearest-neighbor search, efficiently implementable using arrays and vectorization (see later). For oblique trees, each region requires solving a small QP or LP (having $T\Delta$ constraints on average, where Δ is the average leaf depth). Although this is more costly, the number of regions in an oblique forest is far smaller. In our experiments this takes at most a couple of minutes.

Approximate solution The CE found by searching only on the live regions is suboptimal: it has a larger distance than the exact CE. We estimate this in our experiments by comparing with the exact CE (for small forests) and with other existing algorithms for CEs.

It is instructive to consider also an even simpler approximation to CEs: to ignore entirely the forest regions, and search directly in a dataset of instances (say, the training set), but labeled per the forest (not the ground truth). While this is very fast, it is never better than LIRE, because the live regions contain the data instances. In fact, as shown in our experiments, this approach produces CEs with quite a larger distance than LIRE.

Realistic solution A difficult problem with counterfactual explanations with any type of model (not just forests) is that it is difficult to constrain the search space in problem (1) to find realistic instances. Although the input space is defined to be $\mathbb{R}^D$, most real-world data live in a manifold or subset of it. Some domain knowledge can be incorporated through simple constraints (e.g. a grayscale pixel should be in [0,1]), but this is insufficient to capture the subset of realistic instances. Intuitively, this requires estimating the density distribution of the data, a very hard problem in high dimensions, and then constraining problem (1) to that. We can see LIRE in this light as imposing a constraint based on a nonparametric, adaptive kernel density estimate: each live region (a box or polytope) sits on one point and has constant density; all other regions have zero density. The kernel is adaptive, rather than having a given form and bandwidth. This density estimate comes for free with the forest and blends conveniently into the optimization. This makes it more likely that a CE found by searching on the live regions will be more realistic than searching anywhere in the space.

In summary, LIRE can be seen either as an approximate solution to searching CEs in the entire space, or as an exact solution of a CE subject to lying in high density regions, using a nonparametric density estimate. Either way, LIRE is extremely fast and scales to forests with practical sizes.

5 Efficient Implementation of the Live Region Search

Constructing the set of live regions (offline) Obviously, we do not need to build the sparse tensor I_T and test every nonempty region. All we have to do is feed each input instance to the forest and determine which leaf it reaches in each tree. The resulting region is live. See the pseudocode in fig. 1 (bottom). This is done offline and has a complexity of $\mathcal{O}(NT\Delta)$ where Δ is the average depth of a tree. The result is a list of $M \leq N$ regions, each encoded by a leaf tuple $(l_1, \ldots, l_T)$. For axis-aligned trees, each forest region is a box and can be compactly represented by two vectors $\mathbf{a}_n, \mathbf{b}_n \in \mathbb{R}^D$ with $\mathbf{a}_n \leq \mathbf{b}_n$ elementwise, containing the lower and upper bounds along each dimension (or $\pm\infty$ if unbounded). For oblique trees, each region is defined by the intersection of all the constraints (hyperplanes) along one root-leaf path in each tree; computationally, it is better not to construct this list explicitly, instead reconstructing it on the fly during the live region search.

Sorting or indexing the set of live regions (offline) Of all M live regions, we need only search in those that satisfy the constraint $F(\mathbf{x}) \in \mathcal{S}$, which are usually far less

Dataset (N, D, K) (T, Δ, L)	LIRE regions	time (s)	ℓ_2 ℓ_1	dataset time (s)	ℓ_2 ℓ_1	Feature Tweak time (s)	ℓ_2 ℓ_1	feasible	OCEAN time (s)	ℓ_2 ℓ_1
breast cancer (559,9,2)	337	2×10^{-4}	1.00±0.95	1×10^{-4}	1.17±1.08	8.6±1.4	1.25±0.82	100%	2.1±1.0	**0.48±0.32**
(100,9.11,51.94)	337	2×10^{-4}	1.00±0.51	1×10^{-4}	1.25±0.56	8.4±1.6	1.04±0.16	100%	4.6±2.1	**0.45±0.15**
climate (432,18,2)	432	6×10^{-4}	1.00±0.68	2×10^{-4}	1.23±0.54	6.6±1.1	1.04±0.98	100%	3.3±2.9	**0.47±0.45**
(100,8.73,48.74)	432	6×10^{-4}	1.00±0.65	2×10^{-4}	2.67±0.29	7.2±1.8	1.14±0.95	100%	4.1±3.7	**0.38±0.28**
spambase (3.6k,57,2)	3211	6×10^{-3}	1.00±0.90	2×10^{-4}	1.09±0.90	89.6±13.2	0.54±0.36	100%	75.5±82.7	**0.09±0.04**
(100,31.75,596.38)	3211	6×10^{-3}	1.00±0.63	2×10^{-4}	1.15±0.65	91.2±14.6	**0.37±0.29**	100%	timeout	
yeast (1162,8,10)	1162	3×10^{-4}	**1.00±0.78**	2×10^{-4}	1.27±1.15	15.9±8.5	1.12±1.03	66.7%	timeout	
(100,23.59, 734.16)	1162	3×10^{-4}	**1.00±0.76**	2×10^{-4}	1.33±0.96	14.8±8.1	1.15±0.57	66.7%	timeout	
letter (16k,16,26)	14532	9×10^{-4}	**1.00±0.26**	6×10^{-5}	1.32±0.97	58.2±18.5	1.09±0.82	100%	timeout	
(100,27.91,4201.04)	14532	9×10^{-4}	**1.00±0.61**	6×10^{-5}	1.96±0.68	61.1±18.9	1.42±0.37	100%	timeout	
MNIST (55k,784,10)	55000	2×10^{-1}	**1.00±0.73**	4×10^{-2}	1.41±0.89	147.8±48.5	–	0%	timeout	
(100,34.11, 9369.3)	55000	2×10^{-1}	**1.00±0.44**	4×10^{-2}	1.88±0.58	151.8±51.4	–	0%	timeout	
MiniBooNE (104051,50,2)	103692	2×10^{-1}	**1.00±0.80**	1×10^{-2}	1.17±0.87	timeout			timeout	
(100,34.53, 9615.48)	103692	2×10^{-1}	**1.00±0.32**	1×10^{-2}	1.42±0.38	timeout			timeout	
Swarm (18647,2400,2)	8607	8×10^{-1}	**1.00±0.66**	4×10^{-1}	11.21±4.44	timeout			timeout	
(100,31.6, 1463.46)	8607	8×10^{-1}	**1.00±0.41**	4×10^{-1}	27.00±6.31	timeout			timeout	
Swarm (18647,2400,2)	12901	1.3	**1.00±0.70**	4×10^{-1}	5.31±2.10	timeout			timeout	
(1000,31.3, 1468.42)	12901	1.3	**1.00±0.53**	4×10^{-1}	18.59±4.34	timeout			timeout	

Table 1: Comparison of different CE algorithms: LIRE, search on dataset points, Feature Tweak (Tolomei et al. 2017) and OCEAN (Parmentier and Vidal 2021), for different datasets and Random Forests, for axis-aligned trees, and optimizing the ℓ_2 (above) and ℓ_1 (below) distance. We show the resulting distance $\|\mathbf{x}^* - \overline{\mathbf{x}}\|_2$ or $\|\mathbf{x}^* - \overline{\mathbf{x}}\|_1$ and the runtime in seconds (average $\pm$ stdev over 10 source instances). All distances are normalized so that LIRE has unit distance. For LIRE we give the number of live regions and for Feature Tweak the percentage of times the CE found is feasible i.e., it is predicted to be the desired class (all other algorithms are always feasible). For each dataset we give its size, dimensionality and number of classes (N, D, K); for each forest we give its number of trees and average tree depth and number of leaves (T, Δ, L). "timeout" means runtime over 500 s. The best (smallest) distance is in boldface.

Dataset	(N, D, K)	(T, Δ, L)	LIRE regions	time (s)	ℓ_2	dataset time (s)	ℓ_2
breast cancer	(559,9,2)	(30,2.3,5.8)	60	0.07	**1.00±0.52**	0.001	1.23±0.95
spambase	(3.6k,57,2)	(30,2.6,7.8)	214	0.36	**1.00±0.71**	0.011	2.57±2.42
letter	(16k,16,26)	(30, 8.0,289.2)	13238	2.63	**1.00±0.70**	0.004	1.21±0.83
MNIST	(55k,784,10)	(30, 8.0,148.6)	50711	151.97	**1.00±0.83**	0.040	3.03±1.81

Table 2: Like table 1 but using a Random Forest of oblique trees (trained with TAO (Carreira-Perpiñán and Tavallali 2018; Carreira-Perpiñán 2022)).

than M; this greatly accelerates the search. For example, in K-class classification, if our target is a specific class, we need only search $\frac{M}{K}$ regions (assuming uniform class populations). Determining the range of target regions to search can be done by a binary search in $\mathcal{O}(\log M)$ time if we presort the M regions by their value of F; this is useful in regression. In K-class classification, we can pre-index the M regions into K groups (one per class) and determine the target in constant time. This supports complex targets such as $F(\mathbf{x}) \in \{1, 3, 7\}$ or $[2, 5] \cup [7, \infty)$. Then, the actual search in the range of target regions is done sequentially as described next.

Searching in each target live region *Axes-aligned trees.* This can be implemented in a way that is very efficient in time and memory through arrays and vectorization, without any need for the tree structures. Firstly (Carreira-Perpiñán and Hada 2021a), for any distance that is separable over dimensions (e.g. ℓ_1, ℓ_2^2, possibly weighted), the solution to "$\mathbf{x}^* = \arg\min_{\mathbf{x}} d(\mathbf{x}, \overline{\mathbf{x}})$ s.t. $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$" (where $\mathbf{a}$, $\mathbf{b}$ are the lower and upper bounds of a particular box) can be given in closed form as $\mathbf{x}^* = \text{median}(\mathbf{a}, \mathbf{b}, \overline{\mathbf{x}})$ elementwise. (That is, for each dimension, x^* is a if $x \leq a$, b if $x \geq b$ and $\overline{x}$ otherwise.) However, it is more efficient to compute directly the *distance-to-a-box* $d_{\text{box}}(\overline{\mathbf{x}}, \binom{\mathbf{a}}{\mathbf{b}}) \equiv d(\mathbf{x}^*, \overline{\mathbf{x}})$. For the ℓ_2^2 and ℓ_1 distances this is:

$$\|\mathbf{x}^* - \overline{\mathbf{x}}\|_2^2 = \mathbf{1}^T(\max(\mathbf{a} - \overline{\mathbf{x}}, \mathbf{0}) + \max(\overline{\mathbf{x}} - \mathbf{b}, \mathbf{0}))^2,$$

$$\|\mathbf{x}^* - \overline{\mathbf{x}}\|_1 = \mathbf{1}^T(\max(\mathbf{a} - \overline{\mathbf{x}}, \mathbf{0}) + \max(\overline{\mathbf{x}} - \mathbf{b}, \mathbf{0}))$$

where $\max(\cdot, \cdot)$ applies elementwise. This holds by noting that, for each dimension $d = 1, \ldots, D$, $|x_d^* - \overline{x}_d| = a_d - \overline{x}_d$ if $a_d - \overline{x}_d \geq 0$, $\overline{x}_d - b_d$ if $\overline{x}_d - b_d \geq 0$, and 0 oth-

erwise. To preserve memory locality, this can be vectorized over the entire array of $\mathbf{A}_{D\times N} = (\mathbf{a}_1, \dots, \mathbf{a}_N)$ and $\mathbf{B}_{D\times N} = (\mathbf{b}_1, \dots, \mathbf{b}_N)$ to find the box n with the smallest distance to $\overline{\mathbf{x}}$.[2] This shows that, in effect, the problem reduces to *a form of nearest-neighbor search*, where we have a search set of N multidimensional points $\binom{\mathbf{a}_n}{\mathbf{b}_n} \in \mathbb{R}^{2D}$ (each representing a box), a query $\overline{\mathbf{x}} \in \mathbb{R}^D$, and a distance $d_{\text{box}}(\overline{\mathbf{x}}, \binom{\mathbf{a}_n}{\mathbf{b}_n})$ given by the distance-to-a-box. Fig. 4 (top) gives the pseudocode.

Oblique trees. In this case we cannot vectorize using arrays because each region $(l_1, \dots, l_T)$ has an irregular (polytope) shape, given by the constraints for each leaf l_t, $t = 1, \dots, T$ (which, in turn, are the constraints in the root-leaf path to l_t in tree t). So we have to loop through each region, solve its QP or LP, and return the one with minimum distance; see pseudocode in fig. 4 (bottom). As noted earlier, the advantage with oblique trees is that they use few, shallower trees, so the number of regions is much smaller.

Further accelerating the search The exhaustive search over all live regions is very fast. For example, sequentially searching $M = 10^6$ points in $D = 100$ dimensions takes less than a second on a laptop. However, for very large data sets (say, a billion points), this will be too slow. One way to speed this up while finding the exact solution is by parallelizing the search, which can be done trivially over subsets of regions. Another one is by using a search tree, decorated at each node with bounding boxes, to prune sets of regions that are guaranteed not to be optimal. If we allow the search to be inexact, a simple approach is to use live regions for a random sample of data points. It should also be possible to adapt fast techniques to find approximate nearest neighbors in high dimensions. Note that LIRE is an anytime algorithm in that we can stop at any time and return a feasible solution.

Computational complexity As noted earlier, determining the range of regions we need to search (say, the regions with a desired target class) takes negligible time: a logarithmic binary search if the list of regions has been sorted by forest output, or a constant-time lookup if it has been indexed. The cost is dominated by the exhaustive search over the range of regions. For axis-aligned trees, this is $\mathcal{O}(MD)$ with M regions and D features, with a small constant factor due to the distance-to-a-box computation. For oblique trees, we have to solve M QPs (ℓ_2^2) or LPs (ℓ_1). Each has D variables and $T\Delta$ constraints on average (assuming an average leaf depth Δ). In both cases, the search can be trivially parallelized.

6 Experiments

In this section, we used Random Forests (where each tree is grown in full, i.e., not pruned), with individual trees trained by CART (Breiman et al. 1984) if axis-aligned and by TAO (Carreira-Perpiñán and Tavallali 2018; Carreira-Perpiñán 2022) if oblique. All runtimes were obtained in a single core (without parallel processing). Carreira-Perpiñán and Hada (2023) give details about the experiments, as well as more results (e.g. with AdaBoost forests, Realistic CEs, and the training and test error of the forests we trained). Here, we comment on the main results.

LIRE as an approximate CE In order to estimate how good an approximation LIRE is to the exact solution, we do an exhaustive search on all the nonempty regions in small problems for which the latter is computationally feasible. Fig. 5 shows that the approximation (in terms of the distance to the source instance) is quite good, though it degrades as the number of trees increases—since the number of nonempty regions continues to increase exponentially while the number of live regions is capped at N. The approximation is quite better for oblique forests than for axis-aligned ones, in agreement with the fact that the number of regions grows more slowly for oblique forests. Importantly, note than LIRE is far better than searching directly on the dataset instances. This, and its very fast runtime, makes LIRE highly practical in order to get a fast, relatively accurate estimate of the optimal CE.

LIRE vs other algorithms Table 1 compares LIRE with searching on the dataset instances, Feature Tweak (Tolomei et al. 2017) and OCEAN (Parmentier and Vidal 2021). We use several classification datasets of different size, dimensionality and type, and axis-aligned forests (Random Forests) of different size. Unlike previous works on forest CEs, we consider quite larger, high-dimensional datasets and forests—having up to 1 000 trees, with thousands of leaves per tree. This is important because, to achieve competitive performance in practice, the number of trees may need to be quite large. For small problems, OCEAN (which does an exhaustive search) finds the best solution, but its runtime quickly shoots up and becomes intractable for most cases (see detailed comments in Carreira-Perpiñán and Hada (2023)). LIRE is extremely fast even for large problems, comparable to searching on the dataset instances, but finding better CEs. Also, it is guaranteed to find a feasible solution, i.e., producing the desired prediction.

Table 2, for oblique trees on some classification datasets, compares LIRE only with the dataset search, since no other algorithm is applicable. Again, LIRE finds better CEs and is reasonably fast, although for large problems its runtime grows from seconds to minutes.

7 Conclusion

Decision forests define a piecewise constant function with an exponential number of regions in feature space, so searching for a counterfactual explanation exhaustively is impractical unless the forest is very small (in number of trees and of leaves). However, if we restrict the search to only those regions containing at least an actual data point ("live" regions), then the search becomes not only practical but very fast, even suitable for interactive use in some cases. This can also be seen as a realistic formulation of counterfactual explanations where the solution is constrained to lie in high-density regions of feature space, and the live regions act as a nonparametric density estimate. We are working on scaling the search to even larger forests and datasets using pruning heuristics and approximate nearest-neighbor search techniques.

[2] For ex., in Matlab for ℓ_2^2: `[d,n] = min(sum((max(bsxfun(@minus,A,x),0)+max(bsxfun(@minus,x,B),0)).^2,2));`.

References

Breiman, L. 2001. Random Forests. *Machine Learning*, 45(1): 5–32.

Breiman, L. J.; Friedman, J. H.; Olshen, R. A.; and Stone, C. J. 1984. *Classification and Regression Trees*. Belmont, Calif.: Wadsworth.

Carreira-Perpiñán, M. Á. 2022. The Tree Alternating Optimization (TAO) Algorithm: A New Way To Learn Decision Trees and Tree-Based Models. ArXiv.

Carreira-Perpiñán, M. Á.; Gabidolla, M.; and Zharmagambetov, A. 2023. Towards Better Decision Forests: Forest Alternating Optimization. In *Proc. of the 2023 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'23)*. Vancouver, Canada.

Carreira-Perpiñán, M. Á.; and Hada, S. S. 2021a. Counterfactual Explanations for Oblique Decision Trees: Exact, Efficient Algorithms. In *Proc. of the 35th AAAI Conference on Artificial Intelligence (AAAI 2021)*, 6903–6911. Online.

Carreira-Perpiñán, M. Á.; and Hada, S. S. 2021b. Counterfactual Explanations for Oblique Decision Trees: Exact, Efficient Algorithms. ArXiv:2103.01096.

Carreira-Perpiñán, M. Á.; and Hada, S. S. 2023. Very Fast, Approximate Counterfactual Explanations for Decision Forests. ArXiv:2303.02883.

Carreira-Perpiñán, M. Á.; and Tavallali, P. 2018. Alternating Optimization of Decision Trees, with Application to Learning Sparse Oblique Trees. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, 1211–1221. MIT Press, Cambridge, MA.

Carreira-Perpiñán, M. Á.; and Zharmagambetov, A. 2020. Ensembles of Bagged TAO Trees Consistently Improve over Random Forests, AdaBoost and Gradient Boosting. In *Proc. of the 2020 ACM-IMS Foundations of Data Science Conference (FODS 2020)*, 35–46. Seattle, WA.

Cui, Z.; Chen, W.; He, Y.; and Chen, Y. 2015. Optimal Action Extraction for Random Forests and Boosted Trees. In *Proc. of the 21st ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2015)*, 179–188. Sydney, Australia.

Freund, Y.; and Schapire, R. 1997. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *J. Computer and System Sciences*, 55(1): 119–139.

Friedman, J. H. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5): 1189–1232.

Gabidolla, M.; and Carreira-Perpiñán, M. Á. 2022. Pushing the Envelope of Gradient Boosting Forests via Globally-Optimized Oblique Trees. In *Proc. of the 2022 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'22)*, 285–294. New Orleans, LA.

Gabidolla, M.; Zharmagambetov, A.; and Carreira-Perpiñán, M. Á. 2022. Improved Multiclass AdaBoost Using Sparse Oblique Decision Trees. In *Int. J. Conf. Neural Networks (IJCNN'22)*. Padua, Italy.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*. San Diego, CA.

Guidotti, R.; Monreale, A.; Giannotti, F.; Pedreschi, D.; Ruggieri, S.; and Turini, F. 2019. Factual and Counterfactual Explanations for Black Box Decision Making. *IEEE Access*, 34(6): 14–23.

Hada, S. S.; and Carreira-Perpiñán, M. Á. 2021. Exploring Counterfactual Explanations for Classification and Regression trees. In *ECML PKDD 3rd Int. Workshop and Tutorial on eXplainable Knowledge Discovery in Data Mining (XKDD 2021)*, 489–504.

Hastie, T. J.; Tibshirani, R. J.; and Friedman, J. H. 2009. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition.

Kanamori, K.; Takagi, T.; Kobayashi, K.; and Arimura, H. 2007. DACE: Distribution-Aware Counterfactual Explanation by Mixed-Integer Linear Optimization. In *Proc. of the 20th Int. Joint Conf. Artificial Intelligence (IJCAI'07)*, 2855–2862. Hyderabad, India.

Karimi, A.-H.; Barthe, G.; Balle, B.; and Valera, I. 2020. Model-Agnostic Counterfactual Explanations for Consequential Decisions. In *Proc. of the 23rd Int. Conf. Artificial Intelligence and Statistics (AISTATS 2020)*, 895–905. Online.

Lucic, A.; Oosterhuis, H.; Haned, H.; and de Rijke, M. 2022. FOCUS: Flexible Optimizable Counterfactual Explanations for Tree Ensembles. In *Proc. of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022)*, 5313–5322. Online.

Mothilal, R. K.; Sharma, A.; and Tan, C. 2020. Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2020)*, 607–617.

Parmentier, A.; and Vidal, T. 2021. Optimal Counterfactual Explanations in Tree Ensembles. In Meila, M.; and Zhang, T., eds., *Proc. of the 38th Int. Conf. Machine Learning (ICML 2021)*, 8422–8431. Online.

Russell, C. 2019. Efficient Search for Diverse Coherent Explanations. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2019)*, 20–28. Atlanta, GA.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2014. Intriguing Properties of Neural Networks. In *Proc. of the 2nd Int. Conf. Learning Representations (ICLR 2014)*. Banff, Canada.

Tolomei, G.; Silvestri, F.; Haines, A.; and Lalmas, M. 2017. Interpretable Predictions of Tree-based Ensembles via Actionable Feature Tweaking. In *Proc. of the 23rd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2017)*, 465–474. Halifax, Nova Scotia.

Ustun, B.; Spangher, A.; and Liu, Y. 2019. Actionable Recourse in Linear Classification. In *Proc. ACM Conf. Fairness, Accountability, and Transparency (FAT 2019)*, 10–19. Atlanta, GA.

Van Looveren, A.; and Klaise, J. 2021. Interpretable Counterfactual Explanations Guided by Prototypes. In Oliver, N.; Pérez-Cruz, F.; Kramer, S.; Read, J.; and Lozano, J. A., eds., *Proc. of the 32nd European Conf. Machine Learning (ECML–21)*. Bilbao, Spain.

White, A.; and d'Avila Garcez, A. 2020. Measurable Counterfactual Local Explanations for Any Classifier. In Giacomo, G. D.; Catala, A.; Dilkina, B.; Milano, M.; Barro, S.; Bugarín, A.; and Lang, J., eds., *Proc. 24th European Conf. Artificial Intelligence (ECAI 2020)*, 2529–2535.

Yang, Q.; Yin, J.; Ling, C. X.; and Pan, R. 2006. Extracting Actionable Knowledge from Decision Trees. *IEEE Trans. Knowledge and Data Engineering*, 18(1): 43–56.

Zharmagambetov, A.; and Carreira-Perpiñán, M. Á. 2020. Smaller, More Accurate Regression Forests Using Tree Alternating Optimization. In Daumé III, H.; and Singh, A., eds., *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, 11398–11408. Online.

Zharmagambetov, A.; Hada, S. S.; Gabidolla, M.; and Carreira-Perpiñán, M. Á. 2021. Non-Greedy Algorithms for Decision Tree Optimization: An Experimental Comparison. In *Int. J. Conf. Neural Networks (IJCNN'21)*. Virtual event.