Interpretability With Accurate Small Models

Abhishek Ghose* and Balaraman Ravindran

1 Department of Computer Science and Engineering, IIT Madras, Chennai, India, 2 Department of Computer Science and Engineering, Robert Bosch Centre for Data Science and AI, IIT Madras, Chennai, India

Models often need to be constrained to a certain size for them to be considered interpretable. For example, a decision tree of depth 5 is much easier to understand than one of depth 50. Limiting model size, however, often reduces accuracy. We suggest a practical technique that minimizes this trade-off between interpretability and classification accuracy. This enables an arbitrary learning algorithm to produce highly accurate small-sized models. Our technique identifies the training data distribution to learn from that leads to the highest accuracy for a model of a given size. We represent the training distribution as a combination of sampling schemes. Each scheme is defined by a parameterized probability mass function applied to the segmentation produced by a decision tree. An Infinite Mixture Model with Beta components is used to represent a combination of such schemes. The mixture model parameters are learned using Bayesian Optimization. Under simplistic assumptions, we would need to optimize for O(d) variables for a distribution over a d-dimensional input space, which is cumbersome for most real-world data. However, we show that our technique significantly reduces this number to a fixed set of eight variables at the cost of relatively cheap preprocessing. The proposed technique is flexible: it is model-agnostic, i.e., it may be applied to the learning algorithm for any model family, and it admits a general notion of model size. We demonstrate its effectiveness using multiple real-world datasets to construct decision trees, linear probability models and gradient boosted models with different sizes. We observe significant improvements in the F1-score in most instances, exceeding an improvement of 100% in some cases.

Keywords: ML, interpretable machine learning, Bayesian optimization, infinite mixture models, density estimation

1. INTRODUCTION

As Machine Learning (ML) becomes pervasive in our daily lives, there is an increased desire to know how models reach specific decisions. In certain contexts this might not be important as long as the ML model itself works well, e.g., in product or movie recommendations. But for certain others, such as medicine and healthcare (Caruana et al., 2015; Ustun and Rudin, 2016), banking1,
defense applications\(^2\), and law enforcement\(^3\) model transparency is an important concern. Very soon, regulations governing digital interactions might necessitate interpretability (Goodman and Flaxman, 2017).

All these factors have generated a lot of interest around "model understanding." Approaches in the area may be broadly divided into two categories:

1. **Interpretability:** build models that are inherently easy to interpret, e.g., rule lists (Letham et al., 2013; Angelino et al., 2017), decision trees (Breiman et al., 1984; Quinlan, 1993, 2004), sparse linear models (Ustun and Rudin, 2016), decision sets (Lakkaraju et al., 2016), pairwise interaction models that may be linear (Lim and Hastie, 2015), or additive (Lou et al., 2013).

2. **Explainability:** build tools and techniques that allow for explaining black box models, e.g., locally interpretable models such as LIME, Anchors (Ribeiro et al., 2016, 2018), visual explanations for Convolutional Neural Networks such as Grad-CAM (Selvaraju et al., 2017), influence functions (Koh and Liang, 2017), feature attribution based on Shapley values (Lundberg and Lee, 2017; Ancona et al., 2019).

Our work addresses the problem of interpretability by providing a way to increase accuracy of existing models that are considered interpretable.

Interpretable models are preferably small in size: this is referred to as low explanation complexity in Herman (2017), is seen as a form of simulability in Lipton (2018), is a motivation for shrinkage methods (Hastie et al., 2009, section 3.4), and is often otherwise listed as a desirable property for interpretable models (Lakkaraju et al., 2016; Ribeiro et al., 2016; Angelino et al., 2017). For instance, a decision tree of depth = 5 is easier to understand than one of depth = 50. Similarly, a linear model with 10 non-zero terms might be easier to comprehend than one with 50 non-zero terms. This indicates an obvious problem: an interpretable model is often small in its size, and since model size is usually inversely proportional to the bias, a model often sacrifices accuracy for interpretability.

We propose a technique to minimize this tradeoff for any model family; thus our approach is model agnostic. Our technique adaptively samples the provided training data, and identifies a sample on which to learn a model of a given size; the property of this sample being that it is optimal in terms of the accuracy of the constructed model. What makes this strategy practically valuable is that the accuracy of this model may often be significantly higher than one learned on the training data as-is, especially when the model size is small.

Let,

1. \(\text{accuracy}(M, p)\) be the classification accuracy of model \(M\) on data represented by the joint distribution \(p(X, Y)\) of instances \(X\) and labels \(Y\). We use the term “accuracy” as a generic placeholder for a measure of model correctness. This may specifically measure F1-score, AUC, lift, etc., as needed.

2. \(\text{train}_\mathcal{F}(p, \eta)\) produce a model obtained using a specific training algorithm, e.g., CART (Breiman et al., 1984), for a given model family \(\mathcal{F}\), e.g., decision trees, where the model size is fixed at \(\eta\), e.g., trees with depth = 5. The training data is represented by the joint distribution \(p(X, Y)\) of instances \(X\) and labels \(Y\).

If we are interested in learning a classifier of size \(\eta\) for data with distribution \(p(X, Y)\), our technique produces the optimal training distribution \(p^*_\eta(X, Y)\) such that:

\[
\text{accuracy}(\text{train}_\mathcal{F}(p^*_\eta, \eta), p) \leq \text{accuracy}(\text{train}_\mathcal{F}(p, \eta), p) \tag{1}
\]

Here \(q(X, Y)\) ranges over all possible distributions over the data \((X, Y)\).

Training a model on this optimal distribution produces a model that is at least as good as training on the original distribution \(p\):

\[
\text{accuracy}(\text{train}_\mathcal{F}(p, \eta), p) \leq \text{accuracy}(\text{train}_\mathcal{F}(p^*_\eta, \eta), p) \tag{2}
\]

Furthermore, the relationship in Equation (2) may be separated into two regimes of operation. A model trained on \(p^*_\eta\) outperforms one trained on the original distribution \(p\) up to a model size \(\eta'\), with both models being comparably accurate beyond this point:

For \(\eta \leq \eta'\),

\[
\text{accuracy}(\text{train}_\mathcal{F}(p, \eta), p) < \text{accuracy}(\text{train}_\mathcal{F}(p^*_\eta, \eta), p) \tag{3}
\]

For \(\eta > \eta'\),

\[
\text{accuracy}(\text{train}_\mathcal{F}(p, \eta), p) = \text{accuracy}(\text{train}_\mathcal{F}(p^*_\eta, \eta), p) \tag{4}
\]

Our key contributions in this work are:

1. Postulating that the optimal training distribution may be different than the test distribution. This challenges the conventional wisdom that the training and test data must come from the same distribution, as in the LHS of Equations (2), (3), and (4).

2. Providing a model-agnostic and practical adaptive sampling based technique that exploits this effect to learn small models, that often possess higher accuracy compared to using the original distribution.

3. Demonstrating the effectiveness of our technique with different learning algorithms, \(\text{train}_\mathcal{F}()\), and multiple real world datasets. Note that our benchmark is not a specific algorithm that learns small models; the value of our approach is in its being model-agnostic: it works with arbitrary learners.

4. We show that learning the distribution, \(p^*_\eta\), in the \(d\) dimensions of the data, may be decomposed into a relatively cheap preprocessing step that depends on \(d\), followed by a core optimization step independent of \(d\): the optimization is over a fixed set of eight variables. This makes our technique scalable.

We do not impose any constraints on the specification of \(\text{train}_\mathcal{F}()\) for it to create interpretable models; our technique may be used

\(^2\)https://www.darpa.mil/program/explainable-artificial-intelligence

\(^3\)https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm
with any model family. But the fact that we see increased accuracy up to a model size ($\gamma'$ in Equation 3), makes the technique useful in setups where small sized models are preferred. Applications requiring interpretability are an example of this. There may be others, such as model compression, which we have not explored, but briefly mention in section 5.2.

2. OVERVIEW

This section provides an overview of various aspects of our work: we impart some intuition for why we expect the train and test distributions to differ for small-sized models, describe where our technique fits into a model building workflow, mention connections to previous work and establish our notation and terminology.

2.1. Intuition

Let’s begin with a quick demonstration of how modifying the training distribution can be useful. We have the binary class data, shown in Figure 1, that we wish to classify with decision trees with depth = 5.

![Figure 1](binary_class_dataset.png)

**FIGURE 1** | Binary class dataset for classification.

Our training data is a subset of this data (not shown). The training data is approximately uniformly distributed in the input space—see the 2D kernel density plot in the top-left panel in Figure 2. The bottom-left panel in the figure shows the regions of a decision tree with depth = 5 learns, using the CART algorithm. The top-right panel shows a modified distribution of the data (now the density seems to be relatively concentrated away from the edge regions of the input space), and the corresponding decision tree with depth = 5, also learned using CART, is visualized in the bottom-right panel. Both decision trees used the same learning algorithm and possess the same depth. As we can see, the F1 scores are significantly different: 63.58 and 71.87%, respectively.

Where does this additional accuracy come from?

All classification algorithms use some heuristic to make learning tractable, e.g.:

- Decision Trees—one step lookahead (note that the CART tree has a significantly smaller number of leaves than the possible $2^5 = 32$, in our example).
- Logistic Regression—local search, e.g., Stochastic Gradient Descent (SGD).
- Artificial Neural Networks (ANN)—local search, e.g., SGD, Adam.

Increasing the size allows for offsetting the shortcomings of the heuristic by adding parameters to the model till it is satisfactorily accurate: increasing depth, terms, hidden layers, or nodes per layer. Our hypothesis is, in restricting a model to a small size, this potential gap between the representational and effective capacities becomes pronounced. In such cases, modifying the data distribution guides the heuristic to focus learning on regions of the input space that are valuable in terms of accuracy. We are able to empirically demonstrate this effect for DTs in section 4.2.1.

2.2. Workflow

Figure 3 shows how our sampling technique modifies the model building workflow. In the standard workflow, we feed the data into a learning algorithm, $train(x)$, to obtain a model. In our setup, the data is presented to a system, represented by the dashed box, that is comprised of both the learning algorithm and our sampling technique.

This system produces the final model in an iterative fashion: the sampling technique (or sampler) produces a sample using its current distribution parameters, that is used by the learning algorithm to produce a model. This model is evaluated on a validation dataset and the validation score is conveyed back to the sampler. This information is used to modify the distribution parameters and generate a new training sample for the algorithm, and so on, till we reach a stopping criteria. The criteria we use is a specified number of iterations—we refer to this as our budget. The best model produced within the budget, as measured by the validation score, is our final model, and the corresponding distribution is presented as the ideal training distribution.

2.3. Previous Work

We are aware of no prior work that studies the relationship between data distribution and accuracy in the small model regime. In terms of the larger view of modifying the training distribution to influence learning, parallels may be drawn to the following methodologies:

1. When learning on data with class imbalance, using a different train distribution compared to test via over/under-sampling (Japkowicz and Stephen, 2002), is a commonly used strategy. Seen from this perspective, we are positing that modifying the original distribution is helpful in a wider set of circumstances, i.e., when there is no imbalance, as in **Figure 1**, but the model is restricted in size.

2. Among popular techniques, Active Learning (Settles, 2009; Dasgupta, 2011) probably bears the strongest resemblance...
to our approach. However, our problem is different in the following key respects:

(a) In active learning, we don’t know the labels of most or all of the data instances, and there is an explicit label acquisition cost that must be accounted for. In contrast, our work targets the traditional supervised learning setting where the joint distribution of instances and labels is approximately known through a fixed set of samples drawn from that distribution.

(b) Because there is a label acquisition cost, learning from a small subset of the data such that the resulting model approximates one learned on the complete dataset, is strongly incentivized. This economy in sample size is possibly the most common metric used to evaluate the utility of an active learner. This is different from our objective, where we are not interested in minimizing training data size, but in learning small-sized models. Further, we are interested in outperforming a model learned on the complete data.

3. Coreset construction techniques (Bachem et al., 2017; Munteanu and Schwiegelshohn, 2018) seek to create a “summary” weighted sample of a dataset with the property that a model learned on this dataset approximates one learned on the complete dataset. Here too, the difference in objectives is that we focus on small models, ignore training data size, and are interested in outperforming a model learned on the complete data.

This is not to say that the tools of analysis from the areas of active learning or coreset identification cannot be adapted here; but current techniques in these areas do not solve for our objective.
2.4. Terminology and Notation

Let's begin with the notion of “model size.” Even though there is no standard notion of size across model families, or even within a model family, we assume the term informally denotes model attribute(s) with the following properties:

1. size $\propto$ bias$^{-1}$
2. Smaller the size of a model, easier it is to interpret.

As mentioned earlier, only property 1 is strictly required for our technique to be applicable; property 2 is needed for interpretability.

Some examples of model size are depth of decision trees, number of non-zero terms in a linear model and number of rules in a rule set.

In practice, a model family may have multiple notions of size depending upon the modeler, e.g., depth of a tree or the number of leaves. The size might be even be determined by multiple attributes in conjunction, e.g., maximum depth of each tree and number of boosting rounds in the case of a gradient boosted model (GBM). It is also possible that while users of a model might agree on a notion of size they might disagree on the value for the size up to which the model stays interpretable. For example, are decision trees interpretable up to a depth of 5 or 10? Clearly, the definition of size and its admissible values might be subjective. Regardless, the discussion in this paper remains valid as long as the notion of size exhibits the properties above. With this general notion in mind, we say that interpretable models are typically small.

Here are the notations we use:

1. The matrix \( X \in \mathbb{R}^{N \times d} \) represents an ordered collection of \( N \) input feature vectors, each of which has \( d \) dimensions. We assume individual feature vectors \( x_i \in \mathbb{R}^{d \times 1} \) to be column vectors, and hence the \( i \)-th row of \( X \) represents \( x_i^T \). We occasionally treat \( X \) as a set and write \( x_i \in X \) to denote the feature vector \( x_i \) is part of the collection \( X \).

   An ordered collection of \( N \) labels is represented by the vector \( Y \in \mathbb{R}^N \).

   We represent a dataset with \( N \) instances with the tuple \( (X, Y) \), where \( X \in \mathbb{R}^{N \times d} \), \( Y \in \mathbb{R}^N \), and the label for \( x_i \) is \( Y_i \), where \( 1 \leq i \leq N \).

2. The element at the \( p \)-th row and \( q \)-th column indices of a matrix \( A \) is denoted by \( [A]_{pq} \).

3. We refer to the joint distribution \( p(X, Y) \) from which a given dataset was sampled, as the original distribution. In the context of learning a model and predicting on a held-out dataset, we distinguish between the \textit{train}, \textit{validation}, and \textit{test} distributions. In this work, the train distribution may or may not be identical to the original distribution, which would be made clear by the context, but the validation and test distributions are always identical to the original distribution.

4. The terms pdf and pmf denote probability density function and probability mass function, respectively. The term “probability distribution” may refer to either, and is made clear by the context. A distribution \( p \), parameterized by \( \theta \), defined over the variable \( x \), is denoted by \( p(x; \theta) \).

5. We use the following terms introduced before:

   - \textit{accuracy}(\( M, p \)) is the classification accuracy of model \( M \) on data represented by the joint distribution \( p(X, Y) \) of instances \( X \) and labels \( Y \). We often overload this term to use a dataset instead of distribution. In this case, we write \textit{accuracy}(\( M, (X, Y) \)) where \( (X, Y) \) is the dataset.

   - \textit{train}_F(\( p, \eta \)) produces a model obtained using a specific training algorithm for a model family \( F \), where the model size is fixed at \( \eta \). This may also be overloaded to use a dataset, and we write: \textit{train}_F((X, Y), \eta).

6. We denote the depth of a tree \( T \) by the function \textit{depth}(\( T \)).

7. \( \mathbb{R}, \mathbb{Z}, \) and \( \mathbb{N} \) denote the sets of \textit{reals}, \textit{integers}, and \textit{natural numbers}, respectively.

The rest of the paper is organized as follows: in section 3 we describe in detail two formulations of the problem of learning the optimal density. Section 4 reports experiments we have conducted to evaluate our technique. It also presents our analysis of the results. We conclude with section 5 where we discuss some of the algorithm design choices and possible extensions of our technique.

3. METHODOLOGY

In this section we describe our sampling technique. We begin with an intuitive formulation of the problem in section 3.1 to illustrate challenges with a simple approach. This also allows us to introduce the relevant mathematical tools. Based on our understanding here, we propose a much more efficient approach in section 3.3.

3.1. A Naive Formulation

We phrase the problem of finding the ideal density (for the learning algorithm) as an optimization problem. We represent the density over the input space with the pdf \( p(x; \psi) \), where \( \psi \) is a parameter vector. Our optimization algorithm runs for a budget of \( T \) time steps. Algorithm 1 lists the execution steps.

In Algorithm 1:

1. \textit{suggest()} is a call to the optimizer at time \( t \), that accepts past validation scores \( s_{t-1}, \ldots, s_t \) and values of the density parameter \( \psi_{t-1}, \ldots, \psi_t \). These values are randomly initialized for \( t = 1 \). Note that not all optimizers require this information, but we refer to a generic form of optimization that makes use of the entire history.

2. In Line 4, a sampled dataset \( (X_t, Y_t) \) comprises of instances \( x_i \in X_{\text{train}} \), and their corresponding labels \( y_i \in Y_{\text{train}} \). Denoting the sampling weight of an instance \( x_i \) as \( w(x_i) \), we use \( w(x_i) \propto p(x; \psi), \forall x_i \in X_{\text{train}} \).

   The sampling in Line 10 is analogous.

3. Although the training happens on a sample drawn based on \( \psi_t \), the validation dataset \((X_{\text{val}}, Y_{\text{val}})\) isn’t modified by the algorithm and always reflects the original distribution. Hence, \( s_t \) represents the accuracy of a model on the original distribution.


Algorithm 1: Naive formulation

**Data**: Learning algorithm $\text{train}_F()$, size of model $\eta$, data $(X, Y)$, iterations $T$

**Result**: Optimal density $\Psi^*$, accuracy on test set $s_{\text{test}}$

1. Create stratified subsets $(X_{\text{train}}, Y_{\text{train}}), (X_{\text{val}}, Y_{\text{val}}), (X_{\text{test}}, Y_{\text{test}})$ from $(X, Y)$;
2. for $t \leftarrow 1$ to $T$ do
   3. $\Psi_t \leftarrow \text{suggest}(s_t, \ldots, s_1, \Psi_{t-1}, \ldots, \Psi_1) / \text{// suggest() is described below}
      
   4. $(X_t, Y_t) \leftarrow \text{sample } N_t \text{ points from } (X_{\text{train}}, Y_{\text{train}}) \text{ based on } p(X_{\text{train}}; \Psi_t);
   5. $M_t \leftarrow \text{train}_F((X_t, Y_t), \eta);
   6. $s_t \leftarrow \text{accuracy}(M_t, (X_{\text{val}}, Y_{\text{val}}));$
3. end
4. $t^* \leftarrow \arg \max_{t} \{s_1, s_2, \ldots, s_T, s_{T-1}, s_{T+1}\};$
5. $\Psi^* \leftarrow \Psi_{t^*}, M^* \leftarrow M_{t^*};$
6. $s_{\text{test}} \leftarrow \text{accuracy}(M^*, (X_{\text{test}}, Y_{\text{test}}));$
7. return $\Psi^*, s_{\text{test}}$

4. In the interest of keeping the algorithm simple to focus on the salient steps/challenges, we defer a discussion of the sample size $N_t$ to our improved formulation in section 3.3.

Algorithm 1 represents a general framework to discover the optimal density within a time budget $T$. We refer to this as a “naive” algorithm, since within our larger philosophy of discovering the optimal distribution, this is the most direct way to do so. It uses accuracy() as both the objective and fitness function, where the score $s_t$ is the fitness value for current parameters $\Psi_t$. It is easy to see here what makes our technique model-agnostic: the arbitrary learner $\text{train}_F()$ helps define the fitness function but there are no assumptions made about its form. While conceptually simple, clearly the following key implementation aspects dictate its usefulness in practice:

1. The optimizer to use for $\text{suggest()}$.
2. The precise representation of the $pdf$ $p(x; \Psi)$.

We look at these next.

3.1. Optimization

The fact that our objective function is not only a black-box, but is also noisy, makes our optimization problem hard to solve, especially within a budget $T$. The quality of the optimizer $\text{suggest()}$ critically influences the utility of Algorithm 1.

We list below the characteristics we need our optimizer to possess:

1. **Requirement 1: It should be able to work with a black-box objective function.** Our objective function is accuracy(), which depends on a model produced by $\text{train}_F()$. The latter is an input to the algorithm and we make no assumptions about its form. The cost of this generality is that accuracy() is a black-box function and our optimizer needs to work without knowing its smoothness, amenability to gradient estimation etc.

2. **Requirement 2: Should be robust against noise.** Results of accuracy() may be noisy. There are multiple possible sources of noise, e.g.:
   
   (a) The model itself is learned on a sample $(X_t, Y_t)$.
   (b) The classifier might use a local search method like SGD whose final value for a given training dataset depends on various factors like initialization, order of points, etc.

3. **Requirement 3: Minimizes calls to the objective function.** The acquisition cost of a fitness value $s_t$ for a solution $\Psi_t$ is high: this requires a call to accuracy(), which in turn calls train(). Hence, we want the optimizer to minimize such calls, instead shifting the burden of computation to the optimization strategy. The number of allowed calls to accuracy() is often referred to as the fitness evaluation budget.

Some optimization algorithms that satisfy the above properties to varying degrees are the class of Bayesian Optimization (BO) (Brochu et al., 2010; Shahriari et al., 2016) algorithms; evolutionary algorithms such as Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001; Hansen and Kern, 2004) and Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995; Parsopoulos and Vrahatis, 2001); heuristics based algorithms such as Simulated Annealing (Kirkpatrick et al., 1983; Gelfand and Mitter, 1989; Gutjahr and Pflug, 1996); bandit-based algorithms such as Parallel Optimalistic Optimization (Grill et al., 2015) and Hyperband (Li L. et al., 2017).

We use BO here since it has enjoyed substantial success in the area of hyperparameter optimization (e.g., Bergstra et al., 2011; Snoek et al., 2012; Perez et al., 2018; Dai et al., 2019), where the challenges are similar to ours.

While a detailed discussion of BO techniques is beyond the scope of this paper (refer to Brochu et al., 2010; Shahriari et al., 2016 for an overview), we briefly describe why they meet our requirements: BO techniques build their own model of the response surface over multiple evaluations of the objective function; this model serves as a surrogate (whose form is known) for the actual black-box objective function. The BO algorithm relies on the surrogate alone for optimization, bypassing the challenges in directly working with a black-box function (Requirement 1 above). The surrogate representation is also probabilistic; this helps in quantifying uncertainties in evaluations, possibly arising due to noise, making for robust optimization (Requirement 2). Since every call to $\text{suggest()}$ is informed by this model, the BO algorithm methodically focuses on only the most promising regions in the search space, making prudent use of its fitness evaluation budget (Requirement 3).

The family of BO algorithms is fairly large and continues to grow (Bergstra et al., 2011; Hutter et al., 2011; Snoek et al., 2012, 2015; Wang et al., 2013; Gelbart et al., 2014; Hernández-Lobato et al., 2016; Levesque et al., 2017; Li C. et al., 2017; Rana et al., 2017; Malkomes and Garnett, 2018; Perrone et al., 2018; Alvi et al., 2019; Dai et al., 2019; Letham et al., 2019; Nayebi et al., 2019). We use the Tree Structured Parzen Estimator (TPE) algorithm (Bergstra et al., 2011) since it scales linearly with the number of evaluations (the runtime complexity of a naive BO algorithm is cubic in the number of evaluations; see Shahriari
et al., 2016) and has a popular and mature library: Hyperopt (Bergstra et al., 2013).

3.1.2. Density Representation

The representation of the pdf, \( p(x; \Psi) \) is the other key ingredient in Algorithm 1. The characteristics we are interested in are:

1. **Requirement 1:** It must be able to represent an arbitrary density function. This is an obvious requirement since we want to discover the optimal density.

2. **Requirement 2:** It must have a fixed set of parameters. This is for convenience of optimization, since most optimizers cannot handle the conditional parameter spaces that some pdf representations use. A common example of the latter is the popular Gaussian Mixture Model (GMM), where the number of parameters increases linearly with the number of mixture components.

This algorithm design choice allows for a larger scope of being able to use different optimizers in Algorithm 1; there are many more optimizers that can handle fixed compared to conditional parameter spaces. And an optimizer that works with the latter, can work with a fixed parameter space as well\(^4\).

The Infinite Gaussian Mixture Model (IGMM) (Rasmussen, 1999), a non-parametric Bayesian extension to the standard GMM, satisfies these criteria. It side-steps the problem of explicitly denoting the number of components by representing it using a Dirichlet Process (DP). The DP is characterized by a concentration parameter \( \alpha \), which determines both the number of components (also known as partitions or clusters) and association of a data point to a specific component. The parameters for these components are not directly learned, but are instead drawn from prior distributions. We use Beta distributions for our priors too: each shape parameter is drawn from a corresponding prior Beta distribution.

Since we have assumed that the dimensions are independent, we have two prior Beta for the shape parameters per dimension. We obtain the parameters \( \{A_j, B_j\} \) of a Beta for dimension \( j \), \( 1 \leq j \leq d \), by drawing \( A_j \sim \text{Beta}(a_j, b_j) \) and \( B_j \sim \text{Beta}(a'_j, b'_j) \), where \( \{a_j, b_j\} \) and \( \{a'_j, b'_j\} \) are the shape parameters of the priors.

There are a total of \( 4d \) prior parameters, with 4 prior parameters \( \{a_j, b_j, a'_j, b'_j\} \) per dimension \( j \), \( 1 \leq j \leq d \).

We refer to this mixture model as an Infinite Beta Mixture Model (IBMM)\(^5\). For \( d \) dimensional data, we have \( \Psi = \{a, a_1, b_1, a'_1, b'_1, \ldots, a_d, b_d, a'_d, b'_d\} \). This is a total of \( 4d + 1 \) parameters.

Algorithm 2 shows how we sample \( N_t \) points from \((X, Y)\) using the IBMM.

---

**Algorithm 2: Sampling using IBMM**

**Data:** number of points to sample \( N_t \), dataset \((X, Y)\), \( X \in \mathbb{R}^{N_t \times d} \), \( Y \in \mathbb{R}^N \)

**Result:** \((X_t, Y_t)\), \( X_t \in \mathbb{R}^{N_t \times d}, Y_t \in \mathbb{R}^N \)

1. \( X_t = [], Y_t = [] \);
2. \(((c_1, n_1), (c_2, n_2), \ldots, (c_k, n_k)) \leftarrow \text{partition } N_t \text{ using the DP} \quad \text{// Here } \sum_{i=1}^{N_t} n_i = N_t \)
3. for \( i = 1 \) to \( k \) do
4.   // Get the Beta parameters for component \( c_i \)
5.   for \( j = 1 \) to \( d \) do
6.     \( A_{ij} \sim \text{Beta}(a_j, b_j); \)
7.     \( B_{ij} \sim \text{Beta}(a'_j, b'_j); \)
8.   end
9.   for \( l = 1 \) to \( N \) do
10.   \( p(x|c_i) \leftarrow \prod_{j=1}^{d} \text{Beta}(x_{ij}|A_{ij}, B_{ij}); \)
11.   end
12.   \( X_{ti} \leftarrow \text{sample } n_i \text{ points from } X \text{ based on } p(x|c_i); \)
13.   \( Y_{ti} \leftarrow \text{labels corresponding to } X_{ti} \text{ from } Y; \)
14.   \( X_t \leftarrow [X_t \cup X_{ti}], Y_t \leftarrow [Y_t \cup Y_{ti}]; \)
15. end
16. return \((X_t, Y_t)\)

---

\(^4\)The optimizer we use, TPE, can handle conditional spaces. However, as mentioned, our goal is flexibility in implementation.

\(^5\)We justify this name by noting that there is more than one multivariate generalization of the Beta: the Dirichlet distribution is a popular one, but there are others (e.g., Olkin and Trikalinos, 2014).
We first determine the partitioning of the number \( N_x \), induced by the DP (line 2). We use Blackwell-MacQueen sampling (Blackwell and MacQueen, 1973) for this step. This gives us \( k \) components, denoted by \( c_i \), \( 1 \leq i \leq k \), and the corresponding number of points \( n_i, 1 \leq i \leq k \) to be assigned to each component. We then sample points one component at a time: we draw the Beta parameters per dimension—\( A_i, B_i \)—from the priors (lines 4–6), followed by constructing sampling weights \( p(x_i|c_i), \forall x_i \in X \) assuming independent dimensions (line 9).

We emphasize here that we use the IBMM purely for representational convenience. All the \( 4d + 1 \) parameters are learned by the optimizer, and we ignore the standard associated machinery for estimation or inference. These parameters cannot be learned from the data since our fundamental hypothesis is that the optimal distribution is different from the original distribution.

### 3.2. Challenges

The primary challenge with this formulation is the size of the search space. We have successfully tried out Algorithm 1 on small toy datasets as proof-of-concept, but for most real world datasets, optimizing over \( 4d + 1 \) variables leads to an impractically high run-time even using a fast optimizer like TPE.

One could also question the independence assumption for dimensions, but that doesn’t address the problem of the number of variables: learning a pdf directly in \( d \) dimensions would require at least \( O(d) \) optimization variables. In fact, a richer assumption makes the problem worse with \( O(d^2) \) variables to represent inter-dimension interactions.

### 3.3. An Efficient Approach Using Decision Trees

We begin by asking if we can prune the search space in some fashion. Note that we are solving a classification problem, measured by accuracy(); however, the IBMM only indirectly achieves this goal by searching the complete space \( \Psi \). The search presumably goes through distributions with points from only one class, no points close to any or most of the class boundary regions, etc; distributions that decidedly result in poor fitness scores. Is there a way to exclude such “bad” configuration values from the search space?

One strategy would be to first determine where the class boundaries lie, and penalize any density \( \Psi_i \) that doesn’t have at least some overlap with them. This is a common optimization strategy used to steer the search trajectory away from bad solutions. However, implementation-wise, this leads to a new set of challenges:

1. How do we determine, and then represent, the location of class boundaries?
2. What metric do we use to appropriately capture our notion of overlap of \( \Psi_i \) and these locations?
3. How do we efficiently execute the previous steps? After all, our goal is to either (a) reduce the number of optimization variables OR (b) significantly reduce the size of the search space for the current \( O(d) \) variables.

We offer a novel resolution to these challenges that leads to an efficient algorithm by making the optimization “class boundary sensitive.”

Our key insight is an interesting property of decision trees (DT). A DT fragments its input space into axis-parallel rectangles. Figure 4 shows what this looks like when we learn a tree using CART on the dataset from Figure 1. Leaf regions are shown with the rectangles with the black edges.

Note how regions with relatively small areas almost always occur near boundaries. This happens here since none of the class boundaries are axis-parallel, and the DT, in being constrained in representation to axis-parallel rectangles, must use multiple small rectangles to approximate the curvature of the boundary. This is essentially piecewise linear approximation in high dimensions, with the additional constraint that the “linear pieces” be axis-parallel. Figure 5 shows a magnified view of the interaction of leaf edges with a curved boundary. The first panel shows how hypothetical trapezoid leaves might closely approximate boundary curvature. However, since the DT may only use axis-parallel rectangles, we are led to multiple small rectangles as an approximation, as shown in the second panel.

We exploit this geometrical property: in general, leaf regions with relatively small areas (volumes, in higher dimensions) produced by a DT, represent regions close to the boundary. Instead of directly determining an optimal pdf on the input space, we now do the following:

![Image](309x430 to 547x499)

**FIGURE 4 |** Tessellation of space produced by leaves of a decision tree.

![Image](309x554 to 547x710)

**FIGURE 5 |** We see leaves of small areas because DT is forced to approximate curvature with them.
1. Learn a DT, with no size restrictions, on the data \((X_{\text{train}}, Y_{\text{train}})\).
   Assume the tree produces \(m\) leaves, where the region encompassed by a leaf is denoted by \(R_i\), \(1 \leq i \leq m\).
2. Define a \(pmf\) over the leaves, that assigns mass to a leaf in inverse proportion to its volume. Let \(L = \{1, 2, \ldots, m\}\) be a random variable denoting a leaf. Our \(pmf\) is \(P_L(i) = P(L = i) = f(R_i)\), where \(f(R_i) \propto \text{vol}(R_i)^{-1}\).
   The probability of sampling outside any \(R_i\) is set to 0.
3. To sample a point, sample a leaf first, based on the above \(pmf\), and then sample a point from within this leaf assuming a uniform distribution:
   (a) Sample a leaf, \(i \sim P_L\).
   (b) Sample a point within this leaf, \(x \sim U(R_i)\).
   (c) Since leaves are characterized by low entropy of the label distribution, we assign the majority label of leaf \(i\), denoted by \(\text{label}(i)\), to the sampled point \(x\).
   Assuming we have \(k\) unique labels, \(\text{label}(i)\) is calculated as follows:
   Let \(S_i = \{y_j : y_j \in Y_{\text{train}}, x_j \in X_{\text{train}}, x_j \in R_i\}\). Then,
   \[
   \text{label}(i) = \arg\max_k \hat{p}_{ik}
   \] (5)
   where,
   \[
   \hat{p}_{ik} = \frac{1}{|S_i|} \sum_{y_j \in S_i} I(y_j = k)
   \] (6)
   Note here that because of using \(U(R_i)\) we may generate points \(x \notin X_{\text{train}}\). Also, since a point \(x \in R_i \cap X_{\text{train}}\) gets assigned \(\text{label}(i)\), the conditional distribution of labels approximately equals the original distribution:
   \[
   p(Y_j | x_i) \approx p(Y_{\text{train}} | X_{\text{train}})
   \] (7)
   We call such a DT a density tree\(^6\) which we formally define as follows.

**Definition 3.1.** We refer to a DT as a density tree if (a) it is learned on \((X_{\text{train}}, Y_{\text{train}})\) with no size restrictions (b) there is a \(pmf\) defined over its leaves s.t. \(P_L(i) = P(L = i) = f(R_i)\), where \(f(R_i) \propto \text{vol}(R_i)^{-1}\).

Referring back to our desiderata, it should be clear how we address some of the challenges:

1. The location of class boundaries are naturally produced by DTs, in the form of (typically) low-volume leaf regions.
2. Instead of penalizing the lack of overlap with such boundary regions, we sample points in way that favors points close to class boundaries.

Note that in relation to Equation (3) (reproduced below), \(q\) no longer ranges over all possible distributions; but over a restricted set relevant to the problem:
   \[
   p^*_q = \arg\max_q \text{accuracy}(\text{train}_F(q, \eta), p)
   \] (8)

\(^6\)We use this term since this helps us define a pdf over the input space \(\mathbb{R}^d\). We don’t abbreviate this term to avoid confusion with “DT.” DT always refers to a decision tree in this work, and the term “density tree” is used as-is.

We visit the issue of efficiency toward the end of this section.

This simple scheme represents our approach at a high-level. However, this in itself is not sufficient to build a robust and efficient algorithm. We consider the following refinements to our approach:

1. **pmf at the leaf level.** What function \(f\) must we use to construct our \(pmf\)? One could just use \(f(R_i) = c \cdot \text{vol}(R_i)^{-1}\) where \(c\) is the normalization constant \(c = 1/\sum_{i=1}^m \text{vol}(R_i)^{-1}\). However, this quantity changes rapidly with volume. Consider a hypercube with edge-length \(a\) in \(d\) dimensions; the ratio of the (non-normalized) mass between this and another hypercube with edge-length \(a/2\) is \(2^d\). Not only is this change drastic, but it also has potential for numeric underflow.

   An alternative is to use a function that changes more slowly like the inverse of the length of the diagonal, \(f(R_i) = c \cdot \text{diag}(R_i)^{-1}\) where \(c = 1/\sum_{i=1}^m \text{diag}(R_i)^{-1}\). Since DT leaves are axis-parallel hyperrectangles, \(\text{diag}(R_i)\) is always well defined. In our hypercube example, the probability masses are \(\propto 1/(a\sqrt{d})\) and \(\propto 1/(a\sqrt{d}/2)\) when the edge-lengths are \(a\) and \(a/2\), respectively. The ratio of the non-normalized masses between the two cubes is now 2.

   This begs the question: is there yet another \(pmf\) we can use, that is optimal in some sense? Instead of looking for such an optimal \(pmf\), we adopt the more pragmatic approach of starting with a “base” \(pmf\)—we use the inverse of the diagonal length—and then allowing the algorithm to modify it, via smoothing, to adapt it to the data.

2. **Smoothing.** Our algorithm may perform smoothing over the base \(pmf\) as part of the optimization. We use Laplace smoothing (Jurafsky and Martin, 2019, section 3.4), with \(\lambda\) as the smoothing coefficient. This modifies our \(pmf\) thus:
   \[
   f'(R_i) = c \left( f(R_i) + \frac{\lambda}{m} \right)
   \] (9)
   Here, \(c\) is the normalization constant. The optimizer discovers the ideal value for \(\lambda\).

   We pick Laplace smoothing because it is fast. Our framework, however, is general enough to admit a wide variety of options (discussed in section 5.2).

3. **Axis-aligned boundaries.** A shortcoming of our geometric view is if a boundary is axis-aligned, there are no leaf regions of small volumes along this boundary. This foils our sampling strategy. An easy way to address this problem is to transform the data by rotating or shearing it, and then construct a decision tree (see Figure 6). The image on the left shows a DT with two leaves constructed on the data that has an axis-parallel boundary. The image on the right shows multiple leaves around the boundary region, after the data is transformed (the transformation may be noticed at the top left and bottom right regions).

   The idea of transforming data by rotation is not new (Rodriguez et al., 2006; Blaser and Fryzlewicz, 2016). However, a couple of significant differences in our setup are:
(a) We don’t require rotation per se as our specific transformation; any transformation that produces small leaf regions near the boundary works for us.

(b) Since interpretability in the original input space is our goal, we need to transform back our sample. This would not be required, say, if our only goal is to increase classification accuracy.

The need to undo the transformation introduces an additional challenge: we cannot drastically transform the data since sampled points in the transformed space might be outliers in the original space. Figure 7 illustrates this idea, using the same data as in Figure 6.

The first panel shows leaves learned on the data in the transformed space. Note how the overall region covered by the leaves is defined by the extremities—the top-right and bottom-left corners—of the region occupied by the transformed data. Any point within this rectangle is part of some leaf in a DT learned in this space. Consider point $P$—it is valid for our sampler to pick this. The second panel shows what the training data and leaf-regions look like when they are transformed back to the original space. Clearly, the leaves from the transformed space may not create a tight envelope around the data in the original space, and here, $P$ becomes an outlier.

Sampling a significant number of outliers is problematic because:

(a) The validation and test sets do not have these points and hence learning a model on a training dataset with a lot of outliers would lead to sub-optimal accuracies.

(b) There is no way to selectively ignore points like $P$ in their leaf, since we uniformly sample within the entire leaf region. The only way to avoid sampling $P$ is to ignore the leaf containing it (using an appropriate pmf); which is not desirable since it also forces us to ignore the non-outlier points within the leaf.

Note that we also cannot transform the leaves back to the original space first and then sample from them, since (1) we lose the convenience and low runtime of uniform sampling $U(R_i)$; the leaves are not simple hyperrectangles any more; (2) for leaves not contained within the data bounding box in the
original space, we cannot sample from the entire leaf region without risking obtaining outliers again—see point Q in ABCD, in Figure 7.

A simple and efficient solution to this problem is to only slightly transform the data, so that we obtain the small volume leaves at class boundaries (in the transformed space), but also, all valid samples are less likely to be outliers. This may be achieved by restricting the extent of transformation using a “near identity” matrix $A \in \mathbb{R}^{d \times d}$:

\begin{align}
[A]_{pq} &= 1, \text{ if } p = q \\
[A]_{pq} &\sim \mathcal{U}([0, \epsilon]), \text{ if } p \neq q, \text{ where } \epsilon \in \mathbb{R}_{>0} \text{ is a small number.} 
\end{align}

With this transformation, we would still be sampling outliers, but:

(a) Their numbers are not significant now.
(b) The outliers themselves are close to the data bounding box in the original space.

These substantially weaken their negative impact on our technique.

The tree is constructed on $AX$, where $X$ is the original data, and samples from the leaves, $X'_t$, are transformed back with $A^{-1}X'_t$. Figure 6 is actually an example of such a near-identity transformation.

A relevant question here is how do we know when to transform our data, i.e., when do we know we have axis-aligned boundaries? Since this is computationally expensive to determine, we always create multiple trees, each on a transformed version of the data (with different transformation matrices), and uniformly sample from the different trees. It is highly unlikely that all trees in this bagging step would have axis-aligned boundaries in their respective transformed spaces. Bagging also provides the additional benefit of low variance.

We denote this bag of trees and their corresponding transformations by $B$. Algorithm 3 details how $B$ is created. Our process is not too sensitive to the choice of epsilon, hence we set $\epsilon = 0.2$ for our experiments.

4. Selective Generalization. Since we rely on geometric properties alone to define our pmf, all boundary regions receive a high probability mass irrespective of their contribution to classification accuracy. This is not desirable when the classifier is small and must focus on a few high impact regions. In other words, we prioritize all boundaries, but not all of them are valuable for classification; our algorithm needs a mechanism to ignore some of them. We refer to this desired ability of the algorithm as selective generalization.

Figure 8 illustrates the problem and suggests a solution. The data shown has a small green region, shown with a dashed blue circle in the first panel, which we may want to ignore if we had to pick between learning its boundary or the relatively significant vertical boundary. The figure shows two trees of different depths learned on the data—leaf boundaries are indicated with solid black lines. A small tree, shown on the left, automatically ignores the circle boundary, while a larger tree, on the right, identifies leaves around it.

Thus, one way to enable selective generalization is to allow our technique to pick a density tree of appropriate depth.

But a shallow density tree is already part of a deeper density tree—we can just sample at the depth we need. Instead of constructing density trees with different depths, we learn a
“depth distribution” over fully grown density trees; drawing a sample from this tells us what fraction of the tree to consider.

Figure 9A illustrates this idea. The depth distribution is visualized vertically and adjacent to a tree. We sample a depth \( r \in [0,1] \) from the distribution, and scale and discretize it to reflect a valid value for the depth. Let \( \text{depth}(r) \) be the scaling/discretizing function for a tree \( T \). Taking the tree in the figure as our example, \( r = 0 \) implies we sample our data instances from the nodes at \( \text{depth}(r) = 0 \) i.e. at the root, and \( r = 0.5 \) implies we must sample from the nodes at \( \text{depth}(r) = 1 \). We refer to the pmf for the nodes at a depth to be the sampling scheme at that depth. \( T \) has 4 sampling schemes—each capturing class boundary information at a different granularity, ranging from the root with no information and the leaves with the most information.

We use an IBMM for the depth distribution. Similar to the one previously discussed in section 3.1.2, the depth-distribution has a parameter \( \alpha \) for the DP and parameters \( \{a, b, a', b'\} \) for its Beta priors. The significant difference is we have just one dimension now: the depth. The IBMM is shared across all trees in the bag; Algorithm 4 provides details at the end of this section.

5. Revisiting label entropy. When we sampled only from the leaves of a density tree, we could assign the majority label to the samples owing to the low label entropy. However, this is not true for nodes at intermediate levels—which the depth distribution might lead us to sample from. We deal with this change by defining an entropy threshold \( E \). If the label distribution at a node has \( \text{entropy} \leq E \), we sample uniformly from the region encompassed by the node (which may be a leaf or an internal node) and use the majority label. However, if the \( \text{entropy} > E \), we sample only among the training data instances that the node covers. Like \( \epsilon \), our technique is not very sensitive to a specific value of \( E \) (and therefore, need not be learned), as long as it is reasonably low: we use \( E = 0.15 \) in our experiments.

6. Incomplete trees. Since we use CART to learn our density trees, we have binary trees that are always full, but not necessarily complete, i.e., the nodes at a certain depth alone might not represent the entire input space. To sample at such depths, we “back up” to the nodes at the closest depth. Figure 9B shows this: at \( \text{depth} = 0 \) and \( \text{depth} = 1 \), we can construct our pmf with only nodes available at these depths, \( \{A\} \) and \( \{B, C\} \), respectively, and still cover the whole input space. But for \( \text{depth} = 2 \) and \( \text{depth} = 3 \), we consider nodes \( \{B, D, E\} \) and \( \{B, D, F, G\} \), respectively. The dotted red line connects the nodes that contribute to the sampling scheme for a certain depth.

Algorithm 4 shows how sampling from \( B \) works.

**Algorithm 4: Sampling from a bag of density trees, \( B \)**

**Data:** \( \# \) points to sample \( N \), bag of density trees \( B \), depth distribution \( \Psi \), smoothing parameter \( \lambda \)

**Result:** \( (X, Y), X \in \mathbb{R}^{N \times d}, Y \in \mathbb{R}^m \)

1. \( X = [], Y = []; \)
2. for \( i \leftarrow 1 \) to \( N \) do
3. \( r \sim \Psi \);
4. \( T, A \leftarrow \) randomly pick a tree and the corresponding transformation from \( B \);
5. \( \Theta \leftarrow \) construct pmf over the nodes at \( \text{depth}(r) \), smooth with \( \lambda \)/ back-up if depth is incomplete
6. \( L \sim \Theta \) // \( L \) is a node at \( \text{depth}(r) \)
7. \( S_L \leftarrow \{(x_j, y_j): x_j \in X_{\text{train}} \cap R_L \text{ and } y_j \in Y_{\text{train}} \text{ is its label}\}; \)
8. if \( \text{entropy}(L) \leq E \) then
9. \( x \sim L(L), y \leftarrow \text{label}(L) \) // label() defined in Equation 3
10. else
11. \( (x, y) \sim S_L \) // notation: sample a point from \( S_L \)
12. end
13. \( X \leftarrow [X; A^{-1}x], Y \leftarrow [y]; \)
14. end
15. return \((X, Y)\)
Figure 10 illustrates some of the distributions we obtain using our mechanism. Figure 10A shows our data—note, we only have axis-aligned boundaries. In Figures 10B–D, we show the depth distribution at the top, going from favoring the root in Figure 10B, to nodes halfway along the height of the tree in Figure 10C, finally to the leaves in Figure 10D. The contour plot visualizes the distributions, where a lighter color indicates relatively higher sample density. We see that in Figure 10B, we sample everywhere in the data bounding box. In Figure 10C, the larger boundary is identified. In Figure 10D, the smaller boundary is also identified. A bag of size 5 was used and the smoothing coefficient $\lambda$ was held constant at a small value.

This completes the discussion of the salient details of our sampling technique. The optimization variables are summarized below:

1. $\lambda$, the Laplace smoothing coefficient.
2. $\alpha$, the DP parameter.
3. \(\{a, b, a', b'\}\), the parameters of the Beta priors for the IBMM depth distribution. A component/partition \(i\) is characterized by the distribution Beta\((A_i, B_i)\), where \(A_i \sim \text{Beta}(a, b), B_i \sim \text{Beta}(a', b')\).

The IBMM and its parameters, \(\{a, a, b, a', b'\}\), are shared across all trees in the bag \(B\), and \(\lambda\) is shared across all sampling schemes.

We also introduced two additional parameters: \(\epsilon\) and \(E\). As mentioned previously, we do not include them in our optimization since our process is largely insensitive to their precise values as long as these are reasonably small. We use \(\epsilon = 0.2\) and \(E = 0.15\) for our experiments.

The above parameters exclusively determine how the sampler works. In addition, we propose the following parameters:

4. \(N_o \in \mathbb{N}\), sample size. The sample size can have a significant effect on model performance. We let the optimizer determine the best sample size to learn from. We constrain \(N_o\) to be larger than the number of points needed for statistically significant results.

Note that we can allow \(N_o > |X_{\text{train}}|\). This larger sample will be created by either repeatedly sampling points at nodes where the label entropy > \(E\)—or by generating synthetic points when entropy \(\leq E\).

5. \(p_o \in [0, 1]\)—proportion of the sample from the original distribution. Given a value for \(N_o\), we sample \((1 - p_o)N_o\) points from the density tree(s) and \(p_oN_o\) points (stratified) from our training data \((X_{\text{train}}, Y_{\text{train}})\).

Recall that our hypothesis is that learning a distribution helps until a size \(\eta^*(\text{Equation } 3)\). Beyond this size, we need to provide a way for the sampler to reproduce the original distribution. While it is possible the optimizer finds a \(\Psi_t\) that corresponds to this distribution, we want to make this easier: now the optimizer can simply set \(p_o = 1\). Essentially, \(p_o\) is way to “short-circuit” the discovery of the original distribution.

This variable provides the additional benefit that observing a transition \(p_o = 0 \rightarrow 1\), as the model size increases, would empirically validate our hypothesis.

We have a total of eight optimization variables in this technique. The variables that influence the sampling behavior are collectively denoted by \(\Psi = \{\alpha, a, b, a', b'\}\). The complete set of variables is denoted by \(\Phi = \{\Psi, N_o, \lambda, p_o\}\).

This is a welcome departure from our naive solution: the number of optimization variables does not depend on the dimensionality \(d\) at all! Creating density trees as a preprocessing step gives us a fixed set of 8 optimization variables for any data. This makes the algorithm much more efficient than before, and makes it practical to use for real world data.

Algorithm 5 shows how we modify our naive solution to incorporate the new sampler.

As before, we discover the optimal \(\Phi\) using TPE as the optimizer and accuracy() as the fitness function. We begin by constructing our bag of density trees, \(B\), on transformed versions of \((X_{\text{train}}, Y_{\text{train}})\), as described in Algorithm 3. At each iteration in the optimization, based on the current value \(p_o\), we sample data from \(B\) and \((X_{\text{train}}, Y_{\text{train}})\), train our model on it, and evaluate it on \((X_{\text{val}}, Y_{\text{val}})\). In our implementation, lines 7–11 are repeated (thrice, in our experiments) and the accuracies are averaged to obtain a stable estimate for \(s_t\).

Additional details pertaining to Algorithm 5:

1. We use a train : val : test split ratio of 60 : 20 : 20.
2. The training step to build model \(M_t\) in line 10, takes into account class imbalance: it either balances the data by sampling (this is the case with a Linear Probability Model), or it uses an appropriate cost function or instance weighting, to simulate balanced classes (this is case with DTs or Gradient Boosted Models).

However, it is important to note that both \((X_{\text{val}}, Y_{\text{val}})\) and \((X_{\text{test}}, Y_{\text{test}})\) represent the original distribution, and thus indeed test the efficacy of our technique on data with varying degrees of class imbalance.

4. EXPERIMENTS

This section discusses experiments that validate our technique and demonstrate its practical utility. We describe our experimental setup in section 4.1 and present our observations and analysis in section 4.2.
4.1. Setup

We evaluate Algorithm 5 using 3 different learning algorithms, i.e., train_{\mathcal{F}}(), on 13 real-world datasets. We construct models for a wide range of sizes, \( \eta \), to comprehensively understand the behavior of the algorithm. For each combination of dataset, learning algorithm and model size, we record the percentage relative improvement in the F1(macro) score on \((X_{test}, Y_{test})\) compared to the baseline of training the model on the original distribution:

\[
\delta F1 = \frac{100 \times (F1_{new} - F1_{baseline})}{F1_{baseline}}
\]

We specifically choose the F1 macro metric as it accounts for class imbalance, e.g., it penalizes the score even if the model performs well on a majority class but poorly on a minority class.

Since the original distribution is part of the optimization search space, i.e., when \( p_{\delta} = 1 \), the lowest improvement we report is 0\%, i.e., \( \delta F1 \in [0, \infty) \). All reported values of \( \delta F1 \) are averaged over five runs of Algorithm 5. As mentioned before, in each such run, lines 7–11 in the algorithm are repeated thrice to obtain a robust estimate for accuracy(), and thus, \( s_i \).

We also perform upper-tailed paired sample t-tests, with a \( p_{value} \) threshold of 0.1, to assess if the mean of the F1_{new} scores are higher than the mean of the F1_{baseline} scores, in a statistically significant way.

### 4.1.1. Data

We use a variety of real-world datasets, with different dimensionalities, number of classes and different class distributions to test the generality of our approach. The datasets were obtained from the LIBSVM website (Chang and Lin, 2011), and are described in Table 1. The column “Label Entropy,” quantifies the extent of class imbalance, and is computed for a dataset with \( C \) classes in the following way:

\[
\text{Label Entropy} = \frac{1}{\log C} \sum_{j=1,2,...,C} -p_j \log p_j
\]

Here, \( p_j = \frac{|\{x_i|y_i = j\}|}{N} \)

Values close to 1 imply classes are nearly balanced in the dataset, while values close to 0 represent relative imbalance.

### 4.1.2. Models

We use the following model families, \( \mathcal{F} \), and learning algorithms, train_{\mathcal{F}}(), in our experiments:

1. **Decision Trees**: We use the implementation of CART in the scikit-learn library (Pedregosa et al., 2011). Our notion of size here is the depth of the tree.

   **Sizes**: For a dataset, we first learn an optimal tree \( T_{opt} \) based on the F1-score, without any size constraints. Denote the depth of this tree by \( \text{depth}(T_{opt}) \). We then try our
algorithm for these settings of CART’s max_depth parameter: \(\{1, 2, \ldots, \min(\text{depth}(T_{\text{opt}}), 15)\}\), i.e., we experiment only up to a model size of 15, stopping early if we encounter the optimal tree size. Stopping early makes sense since the model has attained the size needed to capture all patterns in the data; changing the input distribution is not going to help beyond this point.

Note that while our notion of size is the actual depth of the tree produced, the parameter we vary is max_depth; this is because decision tree libraries do not allow specification of an exact tree depth. This is important to remember since CART produces trees with actual depth up to as large as the specified max_depth, and therefore, we might not see actual tree depths take all values in \(\{1, 2, \ldots, \min(\text{depth}(T_{\text{opt}}), 15)\}\), e.g., max_depth = 5 might give us a tree with depth = 5, max_depth = 6 might also result in a tree with depth = 5, but max_depth = 7 might give us a tree with depth = 7. We report relative improvements at actual depths.

2. Linear Probability Model (LPM) (Mood, 2010): This is a linear classifier. Our notion of size is the number of terms in the model, i.e., features from the original data with non-zero coefficients. We use our own implementation based on scikit-learn. Since LPMs inherently handle only binary class data, we construct a one-vs-rest model, comprising of as many binary classifiers as there are distinct labels. The given size is enforced for each binary classifier. For instance, if we have a 3-class problem, and we specify a size of 10, then we construct 3 binary classifiers, each with 10 terms. We did not use the more common Logistic Regression classifier because: (1) from the perspective of interpretability, LPMs provide a better sense of variable importance (Mood, 2010) (2) we believe our effect is equally well illustrated by either linear classifier.

We use the Least Angle Regression (Efron et al., 2004) algorithm, that grows the model one term at a time, to enforce the size constraint.

Sizes: For a dataset with dimensionality \(d\), we construct models of sizes: \(\{1, 2, \ldots, \min(d, 15)\}\). Here, the early stopping for LPM happens only for the dataset cod-rna, which has \(d = 8\). All other datasets have \(d > 15\) (see Table 1).

3. Gradient Boosted Model (GBM): We use decision trees as our base classifier in the boosting. Our notion of size is the number of trees in the boosted forest for a fixed maximum depth of the base classifiers. We use the LightGBM library (Ke et al., 2017) for our experiments.

We run two sets of experiments with the GBM, with maximum depths fixed at 2 and 5. This helps us compare the impact of our technique when the model family \(\mathcal{F}\) inherently differs in its effective capacity, e.g., we would expect a GBM with 10 trees and a maximum depth of 5 to be more accurate than a GBM with 10 trees and a maximum depth of 2.

Sizes: If the optimal number of boosting rounds for a dataset is \(r_{\text{opt}}\), we explore the model size range: \(\{1, 2, \ldots, \min(r_{\text{opt}}, 10)\}\). We run two sets of experiments with GBM—one using base classification trees with max_depth = 2, and another with max_depth = 5. Both experiments use the same range for size/boosting rounds.

The density trees themselves use the CART implementation in scikit-learn. We use the Beta distribution implementation provided by the SciPy package (Jones et al., 2001).

4.1.3. Parameter Settings
Since TPE performs optimization with box constraints, we need to specify our search space for the various parameters in Algorithm 5:

1. \(\lambda\): this is varied in the log-space such that \(\log_{10} \lambda \in [-3, 3]\).
2. \(p_{o}\): We want to allow the algorithm to arbitrarily mix samples from \(B\) and \((X_{\text{train}}, Y_{\text{train}})\). Hence, we set \(p_{o} \in [0, 1]\).
3. \(N_{i}\): We set \(N_{i} \in [1000, 10000]\). The lower bound ensures that we have statistically significant results. The upper bound is set to a reasonably large value.
4. \(\alpha\): For a DP, \(\alpha \in \mathbb{R}_{>0}\). We use a lower bound of 0.1.

We rely on the general properties of a DP to estimate an upper bound, \(\alpha_{\text{max}}\). Given \(\alpha\), for \(N\) points, the expected number of components \(k\) is given by:

\[
E[k|\alpha] = O(\alpha H_{N})
\]

\[
E[k|\alpha] \leq \alpha H_{N}
\]

\[
\alpha \geq \frac{E[k|\alpha]}{H_{N}}
\]

Here, \(H_{N}\) is the \(N\)th harmonic sum (see Blei, 2007).

Since our distribution is over the depth of a density tree, we already know the maximum number of components possible, \(k_{\text{max}} = 1 + \text{depth of density tree}\). We use \(N = 1,000\), since this is the lower bound of \(N_{i}\), and we are interested in the upper bound of \(\alpha\) (note \(H_{N} \propto N\)—see section 1.3). We set \(k_{\text{max}} = 100\) (this is greater than any of the density tree depths in our experiments) to obtain a liberal upper bound, \(\alpha_{\text{max}} = 100/H_{1000} = 13.4\). Rounding up, we set \(\alpha \in [0.1, 14]\).

We draw a sample from the IBMM using Blackwell-MacQueen sampling (Blackwell and MacQueen, 1973).

5. \(a, b, a', b'\): Each of these parameters are allowed a range \([0.1, 10]\) to admit various shapes for the Beta distributions.

We need to provide a budget \(T\) of iterations for the TPE to run. In the case of DT, GBM and binary class problems for LPM, \(T = 3,000\). Since multiclass problems in LPM require learning multiple classifiers, leading to high running times, we use a lower value of \(T = 1,000\). We arrived at these budget values by trial and error; not low enough to lead to inconclusive results, not unreasonably high to run our experiments.

4.2. Observations and Analysis
We break down our results and discussion by the train\(_X\)() used.

4.2.1. DT Results
The DT results are shown in Table 2. A series of unavailable scores, denoted by “-,” toward the right end of the table for a dataset denotes we have already reached its optimal size. For example, in Table 2, cod-rna has an optimal size of 10.

\footnote{We later observe from our experiments that this upper bound is sufficient since nearly all depth distributions have at most 2 dominant components (see Figures 14–17).}
TABLE 2 | Classification Results with DTs.

| Datasets       | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| cod-rna        | 4.55| 0.82| 0.24| 2.18| 0.29| 0.12| 0.22| 0.22| 0.17| 0.00| –   | –   | –   | –   | –   |
| ijcnn1         | 3.20| 12.96| 8.65| 9.04| 3.38| 1.20| 2.36| 1.22| 0.93| 0.00| 0.25| 0.39| 0.60| 0.00| 0.68|
| higgs          | 3.75| 0.94| 0.94| 0.45| 0.00| 0.47| –   | –   | –   | –   | –   | –   | –   | –   | –   |
| covtype.binary | 0.41| 0.33| 0.79| 0.78| 0.76| 0.76| 0.41| 0.57| 0.63| 0.82| 0.00| 1.20| 0.00| 0.68| –   |
| phishing       | 0.10| 0.70| 0.29| 0.24| 0.62| 0.11| 0.11| 0.35| 0.07| 0.10| 0.00| 0.00| 0.00| 0.00| 0.00|
| a1a            | 0.00| 0.59| 5.60| 4.77| 4.09| 2.40| 3.45| 0.82| 1.83| –   | 0.65| 0.00| 2.35| –   | 0.92|
| pendlgts      | 10.49| 3.08| 4.46| 9.81| 4.66| 2.18| 1.04| 0.07| 0.09| 0.00| 0.00| 0.00| 0.00| 0.00| 0.10|
| letter         | 1.04| 10.52| 32.46| 46.79| 45.96| 20.70| 9.35| 5.05| 4.21| 2.14| 0.54| 0.00| 0.00| 0.00| 0.00|
| Sensorless     | 0.00| 41.97| 71.69| 81.43| 38.33| 16.76| 7.86| 4.30| 2.20| 1.10| 0.63| 0.16| 0.75| 0.28| 0.00|
| senseit_aco    | 19.64| 0.57| 3.09| 1.47| 1.49| 0.47| 0.39| 0.93| –   | –   | –   | –   | –   | –   | –   | –   |
| senseit_sei    | 2.18| 1.04| 2.15| 0.53| 0.97| 1.02| 0.43| –   | –   | –   | –   | –   | –   | –   | –   | –   |
| covtype        | 27.80| 101.80| 18.62| 7.01| 6.30| 2.41| 2.11| 2.40| 1.64| 2.01| 2.58| 1.82| 0.00| 0.69| 0.52|
| connect-4      | 181.33| 32.51| 18.12| 12.78| 7.33| 11.14| 4.19| 6.10| 2.71| 1.19| 2.78| 2.08| 2.13| 2.08| 1.55|

Values indicate improvements $\delta F1$, averaged over five runs. Entries in bold are statistically significant. Underlined entries denote the best improvement for a dataset.

For each dataset, the best improvement across different sizes is shown underlined. As mentioned before, we perform upper-tailed paired sample t-tests at a $p$-value threshold of 0.1, to compare the original and new $F1$ scores. Table 2 shows the statistically significant entries in bold, and entries that are not statistically significant are shown in regular font. The horizontal line separates binary datasets from multiclass datasets.

This data is also visualized in Figure 11. The x-axis shows a scaled version of the actual tree depths for easy comparison: if the largest actual tree depth explored is $\eta_{\text{max}}$ for a dataset, then a size $\eta$ is represented by $\eta/\eta_{\text{max}}$. This allows us to compare a dataset like cod-rna, which only has models up to a size of 10, with covtype, where model sizes go all the way up to 15.

We observe significant improvements in the $F1$-score for at least one model size for majority of the datasets. The best improvements themselves vary a lot, ranging from 0.70% for phishing to 181.33% for connect-4. More so, these improvements seem to happen at small sizes: only one best score—for covtype.binary—shows up on the right half of Table 2. This is inline with Equations (3) and (4): beyond a model size $\eta'_{\text{max}}$, $\delta F1 = 0$.

It also seems that we do much better with multiclass data than with binary classes. Because of the large variance in improvements, this is hard to observe in Figure 11. However, if we separate the binary and multiclass results, as in Figure 12, we note that there are improvements in both the binary and multiclass cases, and the magnitude in the latter are typically higher (note the y-axes). We surmise this happens because, in
general, DTs of a fixed depth have a harder problem to solve when the data is multiclass, providing our algorithm with an easier baseline to beat.

Class imbalance itself doesn’t seem to play a role. As per Table 1, the datasets with most imbalance are *ijcnn1*, *covtype*, *connect-4*, for which we see best improvements of 12.96, 101.80, 181.33%, respectively.

Most of the statistically significant results occur at small model sizes (note how most bold entries are on the left half of the table), reinforcing the validity of our technique. Since some of the models go up to (or close to) the optimal model size—the last column in Table 2 for these datasets are either empty or tending to 0 (all datasets except *ijcnn1*, *a1a*, *covtype*, *connect-4* satisfy this condition)—a significant $\delta F_1$ is also not expected.

Figure 13 shows the behavior of $p_o$, only for the datasets where our models have grown close to the optimal size. Thus, we exclude *ijcnn1*, *a1a*, *covtype*, *connect-4*. We observe that indeed $p_o \rightarrow 1$ as our model grows to the optimal size. This empirically validates our hypothesis from section 2.1, that smaller models prefer a distribution different from the original distribution to learn from, but the latter is optimal for larger models. And we gradually transition to it as model size increases.

Demonstrating this effect is a key contribution of our work. We are also interested in knowing what the depth-distribution IBMM looks like. This is challenging to visualize for multiple datasets in one plot, since we have an optimal IBMM learned by our optimizer, for each model size setting. We summarize this information for a dataset in the following manner:

1. Pick a sample size of $N$ points to use.
2. We allocate points to sample from the IBMM for a particular model size, in proportion of $\delta F_1$. For instance, if we have experimented with three model sizes, and $\delta F_1$ are 7, 11, and 2%, we sample 0.35, 0.55, and 0.1 $N$ points, respectively from the corresponding IBMMs.
3. We fit a Kernel Density Estimator (KDE) over these $N$ points, and plot the KDE curve. This plot represents the IBMM across model sizes for a dataset weighted by the improvement seen for a size.

$N$ should be large enough that the visualization is robust to sample variances. We use $N = 10,000$. 

---

**FIGURE 12** | Performance on binary vs. multi-class classification problems using CART. This is an elaboration of Figure 11.

**FIGURE 13** | Variation of $p_o$ with increasing model size.
Figure 14 shows such a plot for DTs. The x-axis represents the depth of the density tree normalized to [0, 1]. The smoothing by the KDE causes some spillover beyond these bounds.

We observe that, in general, the depth distribution is concentrated either near the root of a density tree, where we have little or no information about class boundaries and the distribution is nearly identical to the original distribution, or at the leaves, where we have complete information of the class boundaries. An intermediate depth is relatively less used. This pattern in the depth distribution is surprisingly consistent across all the models and datasets we have experimented with. We hypothesize this might be because of the following reasons:

1. The information provided at an intermediate depth—where we have moved away from the original distribution, but have not yet completely discovered the class boundaries—might be relatively noisy to be useful.

2. The model can selectively generalize well enough from the complete class boundary information at the leaves.

Note that while fewer samples are drawn at intermediate depths, the number is not always insignificant—as an example, see `pendigits` in Figure 14; hence using a distribution across the height of the density tree is still a useful strategy.

4.2.2. LPM Results

The results for LPM are shown in Table 3. The improvements look different from what we observed for DT, which is to be expected across different model families. Notably, compared to DTs, there is no prominent disparity in the improvements.
between binary class and multiclass datasets. Since the LPM builds one-vs.-rest binary classifiers in the multiclass case, and the size restriction—number of terms—applies to each individually, this intuitively makes sense. This is unlike DTs where the size constraint was applied to a single multiclass classifier. However, much like DTs, we still observe the pattern of the greatest improvements occurring at relatively smaller model sizes.

Figure 15 shows the plots for improvement in the F1-score and the weighted depth distribution. The depth distribution plot displays concentration near the root and the leaves, similar to the case of the DT in Figure 14.

Note that unlike the case of the DT, we haven’t determined how many terms the optimal model for a dataset has; we explore up to \(\min(d, 15)\). Nevertheless, as in the case of DTs, we note the pattern that the best improvements typically occur at smaller sizes: only higgs exhibits its largest improvements at a relatively large model size in Table 3. Here too, class imbalance doesn’t seem to play a role (datasets with most imbalance—ijcnn1, covtype, connect-4—show best improvements of 17.9, 27.84, 76.68%, respectively, and most results at small imbalance don’t seem to play a role (datasets with most imbalance—ijcnn1, covtype, connect-4—show best improvements of 17.9, 27.84, 76.68%, respectively, and most results at small imbalance). The complete results are made available in Table A1 in the Appendix. We present both the improvement in the F1 score, \(\delta F1\), and its new value, \(F1_{\text{new}}\).

Figures 16, 17 show the improvement and depth distribution plots for the GBMs with \(\max \_ \text{depth} = 2\) and \(\max \_ \text{depth} = 5\), respectively. The cells highlighted in blue in Table 4 are where the GBM with \(\max \_ \text{depth} = 2\) showed a larger improvement than a GBM with \(\max \_ \text{depth} = 5\) for the same number of boosting rounds. The cells highlighted in red exhibit the opposite case. Clearly, both factors manifest themselves. Comparing the relative improvement plots in Figures 16, 17, we see that improvements continue up to larger sizes when \(\max \_ \text{depth} = 2\) (also evident from Table 4). This is not surprising: we expect a stronger model to extract patterns from data at relatively smaller sizes, compared to a weaker model.

1. A weak model family implies it might not learn sufficiently well from the samples our technique produces. Hence, we expect to see smaller improvements than when using a stronger model family.
2. A weak model family implies there is a lower baseline to beat. Hence, we expect to see larger improvements.

We present an abridged version of the GBM results in Table 4 in the interest of space. The complete results are made available in Table A1 in the Appendix. We present both the improvement in the F1 score, \(\delta F1\), and its new value, \(F1_{\text{new}}\).

4.2.3. GBM Results

An interesting question to ask is how, if at all, the bias of the model family of \(F\) in Algorithm 5, influences the improvements in accuracy. We cannot directly compare DTs with LPMs since we don’t know how to order models from different families: we cannot decide how large a DT to compare to a LPM with, say, 4 non-zero terms.

To answer this question we look at GBMs where we identify two levers to control the model size. We consider two different GBM models—with the \(\max \_ \text{depth}\) of base classifier trees as 2 and 5, respectively. The number of boosting rounds is taken as the size of the classifier and is varied from 1 to 10. We refer to the GBMs with base classifiers with \(\max \_ \text{depth} = 2\) and \(\max \_ \text{depth} = 5\) as representing weak and strong model families, respectively.

We recognize that qualitatively there are two opposing factors at play:

1. A weak model family implies it might not learn sufficiently well from the samples our technique produces. Hence, we expect to see smaller improvements than when using a stronger model family.
2. A weak model family implies there is a lower baseline to beat. Hence, we expect to see larger improvements.

Observe that in Table 4, for the same number of boosting rounds, the new scores \(F1_{\text{new}}\) for the weaker GBMs are up to as large (within some margin of error) as the scores for the stronger GBMs. This is to be expected since our sampling technique diminishes the gap between representational and effective capacities (when such a gap exists); it does not improve the representational capacity itself. Hence a weak classifier using our method is not expected to outperform a strong classifier that is also using our method.

The depth distribution plots for the GBMs show a familiar pattern: high concentration at the root or the leaves. Also, similar to DTs and LPMs, the greatest improvements for a dataset mostly occur at relatively smaller model sizes (see Table A1).

4.2.4. Summary

Summarizing our analysis above:

FIGURE 15 | Linear Probability Model: improvements and the distribution over depths of the density trees.
1. We see significant improvements in the F1 score across different combinations of model families, model sizes and datasets with different dimensionalities and label distributions.

2. Since in the DT experiments, we have multiple datasets for which we reached the optimal tree size, we were able to empirically validate the following related key hypotheses:
(a) With larger model sizes the optimal distribution tends toward the original distribution. This is conveniently indicated with $p_o \to 1$ as $\eta$ increases.
(b) There is model size $\eta'$, beyond which $\delta F_1 \approx 0%$.

3. For all the model families experimented with—DTs, LPMs, GBMs (results in Table A1)—the greatest improvements are seen for relatively smaller model sizes.

4. In the case of DTs, the improvements are, in general, higher with multiclass than binary datasets. We do not see this disparity for LPMs. We believe this happens because of our subjective notion of size: in the case of DTs there is a single tree to which the size constraint applies, making the baseline easier to beat for multiclass problems; while for LPMs it applies to each one-vs.-rest linear model.

Its harder to characterize the behavior of the GBMs in this regard, since while the base classifiers are DTs, each of which is a multiclass classifier, a GBM maybe comprised of multiple DTs.

5. The GBM experiments give us the opportunity to study the effect of using model families, $F$, of different strengths. We make the following observations:

(a) We see both these factors at work: (1) a weaker model family has an easier baseline to beat, which may lead to higher $\delta F_1$ scores relative to using a stronger model family (2) a stronger model family is likely to make better use of the optimal distribution, which may lead to higher $\delta F_1$ scores relative to using a weaker model family.
(b) For a stronger model family, the benefit of using our algorithm diminishes quickly as model size grows.
(c) While the improvement $\delta F_1$ for a weaker family may exceed one for a stronger family, the improved score $F_1_{\text{new}}$ may, at best, match it.

6. The depth distribution seems to favor either nodes near the root or the leaves, and this pattern is consistent across learning algorithms and datasets.

Given our observations, we would recommend using our approach as a pre-processing step for any size-limited learning, regardless of whether the size is appropriately small for our technique to be useful or not. If the size is large, then our method will return to the original sample anyways.

5. DISCUSSION

In addition to empirically validating our algorithm, the previous section also provided us with an idea of the kind of results we might expect of it. Using that as a foundation, we revisit our algorithm in this section, to consider some of our design choices and possible extensions.

5.1. Algorithm Design Choices

Conceptually, Algorithm 5 consists of quite a few building blocks. Although we have justified our implementation choices for them in section 3, it is instructive to look at some reasonable alternatives.

1. Since we use our depth distribution to identify the value of a depth $\in \mathbb{Z}_{\geq 0}$, a valid question is why not use a discrete distribution, e.g., a multinomial? Our reason for using a continuous distribution is that we can use a fixed number of optimization variables to characterize a density tree of any depth, with just an additional step of discretization. Also, recall that the depth distribution applies to all density trees in the forest $B$, each of which may have a different depth. A continuous distribution affords us the convenience of not having to deal with them individually.

2. A good candidate for the depth distribution is the Pitman-Yor process (Pitman and Yor, 1997)—a two-parameter generalization of the DP (recall, this has one parameter: $\alpha$). Considering our results in Figures 14–17, where most depth distributions seem to have up to two dominant modes, we did not see a strong reason to use a more flexible distribution at the cost of introducing an optimization variable.

3. We considered using the Kumaraswamy distribution (Kumaraswamy, 1980) instead of Beta for the mixture components. The advantage of the former is its cumulative distribution function maybe be expressed as a simple formula, which leads to fast sampling. However, our tests with a Python implementation of the function showed no significant benefit over the Beta in the SciPy package, for our use case: the depth distribution is in one dimension, and we draw samples in batches (all samples for a component are drawn simultaneously). Consequently, we decided to stick to the more conventional Beta distribution.

5.2. Extensions and Applications

Our algorithm is reasonably abstracted from low level details, which enables various extensions and applications. We list some of these below:

1. Smoothing: We had hinted at alternatives to Laplace smoothing in section 3.3. We discuss one possibility here. Assuming our density tree has $n$ nodes, we let $S \in \mathbb{R}^{n \times n}$ denote a pairwise similarity matrix for these nodes, i.e., $[S]_{ij}$ is the similarity score between nodes $i$ and $j$. Let $P \in \mathbb{R}^{1 \times n}$ denote the base (i.e., before smoothing) probability masses for the nodes. Normalizing $P \times S^t, k \in \mathbb{Z}_{\geq 0}$ gives us a smoothed pmf that is determined by our view of similarity between nodes. Analogous to transition matrices, the exponent $k$ determines how diffuse the similarity is; this can replace $\lambda$ as an optimization variable.

The ability to incorporate a node similarity matrix opens up a wide range of possibilities, e.g., $S$ might be based on the Wu-Palmer distance (Wu and Palmer, 1994), SimRank (Jeh and Widom, 2002), or Random Walk with Restart (RWR) (Pan et al., 2004).

2. Categorical variables: We have not explicitly discussed the case of categorical features. There are a couple of ways to handle data with such features:

---

Interestingly, another recent paper on interpretability does use the Kumaraswamy distribution (Bastings et al., 2019).
(a) The density tree may directly deal with categorical variables. When sampling uniformly from a node that is defined by conditions on both continuous and categorical variables, we need to combine the outputs of a continuous uniform sampler (which we use now) and a discrete uniform sampler (i.e., multinomial with equal masses) for the respective feature types.

(b) We could create a version of the data with one-hot encoded categorical features for constructing the density tree. For input to \(\text{train}_F()\) at each iteration, we transform back the sampled data by identifying values for the categorical features to be the maximums in their corresponding sub-vectors. Since the optimizer already assumes a black-box \(\text{train}_F()\) function, this transformation would be modeled as a part of it.

3. Model compression: An interesting possible use-case is model compression. Consider the column \(\text{boosting round} = 1\) for the \text{senset-set} dataset in Table 4. Assuming the base classifiers have grown to their max_depths, the memory footprint in terms of nodes for the GBMs with max_depth = 2 and max_depth = 5 are \(2^2 + 1 = 5\) and \(2^5 + 1 = 33\), respectively.

Replacing the second model (larger) with the first (small) in a memory constrained system reduces footprint by \((33 − 5)/33 = 85\%\) at the cost of changing the F1 score by \((0.60 − 0.62)/0.62 = −3.2\%\) only.

Such a proposition becomes particularly attractive if we look at the baseline scores, i.e., accuracies on the original distribution. For the larger model, \(F_1\)baseline = \(F_1\)new/(1 + \(\delta F_1/100\)) = 0.62/(1 + 1.8046) = 0.22. If we replace this model with the smaller model enhanced by our algorithm, we not only reduce the footprint but actually improve the F1 score by \((0.60 − 0.22)/0.22 = 173.7\%\).

We precisely state this application thus: our algorithm may be used to identify a model size \(\eta_e\) (subscript “e” for “equivalent”) in relation to a size \(\eta > \eta_e\) such that:

\[
\text{accuracy}(\text{train}_F(p^*_s, \eta_e), p) \approx \text{accuracy}(\text{train}_F(p, \eta), p) \quad (16)
\]

4. Segment analysis: Our sampling operates within the bounding box \(U \subset \mathbb{R}^d\); in previous sections, \(U\) was defined by the entire input data. However, this is not necessary: we may use our algorithm on a subset of the data \(V \subset U\), as long as \(V\) is a hyperrectangle in \(\mathbb{R}^{d'}\), \(d' \leq d\). This makes our algorithm useful for applications like cohort analysis, common in marketing studies, where the objective is to study the behavior of a segment—say, based on age and income—within a larger population. Our algorithm is especially appropriate since traditionally such analyses have emphasized interpretability.

5. Multidimensional size: The notion of size need not be a scalar. Our GBM experiments touch upon this possibility. The definition of size only influences how the call to \(\text{train}_F()\) internally executes; Algorithm 5 itself is agnostic to this detail. This makes our technique fairly flexible. For example, it is easy in our setup to vary both max_depth and number of boosting rounds for GBMs.

6. Different optimizers: As mentioned in section 3.1.2, the fact that our search space has no special structure implies the workings of the optimizer is decoupled from the larger sampling framework. This makes it easy to experiment with different optimizers. For example, an interesting exercise might be to study the effect of the hybrid optimizer \(\text{Bayesian Optimization with Hyperband (BOHB)}\) (Falkner et al., 2018) when \(\text{train}_F()\) is an iterative learner; BOHB uses an early stopping strategy in tandem with Bayesian Optimization.

7. Over/Under-sampling: As the range of the sample size parameter \(N_i\) is set by the user, the possibility of over/under-sampling is subsumed by our algorithm. For instance, if our dataset has 500 points, and we believe that sampling up to 4 times might help, we can simply set \(N_i \in [500, 2,000]\). Over/Under-sampling need not be explored as a separate strategy.

6. CONCLUSION

Our work addresses the trade-off between interpretability and accuracy. The approach we take is to identify an optimal training distribution that often dramatically improves model accuracy for an arbitrary model family, especially when the model size is small. We believe this is the first such technique proposed. We have framed the problem of identifying this distribution as an optimization problem, and have provided a technique that is empirically shown to be useful across multiple learning algorithms and datasets. In addition to its practical utility, we believe this work is valuable in that it challenges the conventional wisdom that the optimal training distribution is the test distribution.

A unique property of our technique is that beyond a pre-processing step of constructing a DT, which we refer to as a density tree, the number of variables in the core optimization step does not depend on the dimensionality of the data; it uses a fixed set of eight variables. The density tree is used to determine a feasible space of distributions to search through, making the optimization efficient. Our choice of using DTs is innovative since while all classifiers implicitly identify boundaries, only few classifiers like DTs, rules, etc., can explicitly indicate their locations in the feature space. We have also discussed how our algorithm may be extended in some useful ways.

We hope that the results presented here would motivate a larger discussion around the effect of training distributions on model accuracy.

DATA AVAILABILITY STATEMENT

The datasets analyzed for this study can be found on the LIBSVM website at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html and https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.
AUTHOR CONTRIBUTIONS
AG and BR have jointly formulated the problem, worked on certain aspects of the representation and designed experiments. AG has additionally worked on practical aspects of the representation, and executed the experiments.

REFERENCES
Alimoglu, F., and Alpaydin, E. (1996). "Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition," in Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAANN 96) (Istanbul).
Angelino, E., Larus-Stone, N., Alabi, D., Seltzer, M., and Rudin, C. (2017). "Learning certifiably optimal rule lists," in Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '17 (New York, NY), 35–44.
Bacchus, O., Lucie, M., and Krause, A. (2017). Practical Costless Constructions for Machine Learning, Available online at: https://ui.adsabs.harvard.edu/abs/2017arXiv170306476B
Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. Nat. Commun. 5:4308. doi: 10.1038/ncomms5308
Bastings, I., Aziz, W., and Titov, I. (2019). "Interpretatable neural predictions with differentiable binary variables," in Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (Florence: Association for Computational Linguistics), 2963–2977.
Bergera, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). "Algorithms for hyperparameter optimization," in Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11 (Stockholm: Curran Associates Inc.), 2546–2554.
Bergera, J., Yamins, D., and Cox, D. D. (2013). "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28, ICML'13, 1-115–1-123 (Atlanta, GA).
Blackwell, D., and MacQueen, J. B. (1973). Ferguson distributions via polya urn schemes. Ann. Stat. 1, 353–355.
Blaser, R., and Fryzlewicz, P. (2016). Random rotation ensembles. J. Mach. Learn. Res. 17, 1–26.
Blei, D. (2007). COS 597C Notes, Bayesian Nonparametrics. Available online at: https://www.cs.princeton.edu/courses/archive/fall07/cos597C/scribe/20070921.pdf
Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). Classification and Regression Trees. New York, NY: Chapman & Hall.
Brochu, E., Cora, V. M., and de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to user modeling and hierarchical reinforcement learning. CoRR abs/1012.2399. [Preprint].
Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., and Elhadad, N. (2015). "Intelligible models for healthcare: predicting pneumonia risk and hospital 30-day readmission," in Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15 (New York, NY), 1721–1730.
Chang, C.-C., and Lin, C.-J. (2001). "IJCNN 2001 challenge: generalization ability and text decoding," in Proceedings of IJCNN. IEEE (Washington, DC), 1031–1036.
Chang, C.-C., and Lin, C.-J. (2011). LIBSVM: A library for support vector machines. ACM Trans. Intell. Syst. Technol. 2, 27:1–27:27. doi: 10.1145/1961189.1961199
Collobert, R., Bengio, S., and Bengio, Y. (2002). "A parallel mixture of svms for very large scale problems," in Advances in Neural Information Processing Systems 14, eds T. G. Dietterich, S. Becker, and Z. Ghahramani (Montréal, QC: MIT Press), 633–640.
Dai, Z., Yu, H., Low, B. K. H., and Jaillet, P. (2019). "Bayesian optimization meets Bayesian optimal stopping," in Proceedings of the 36th International Conference on Machine Learning, Vol. 97 of Proceedings of Machine Learning Research, eds K. Chaudhuri and R. Salakhutdinov (Long Beach, CA), 1496–1506.
Dasgupta, S. (2011). Two faces of active learning. Theor. Comput. Sci. 412, 1767–1781. doi: 10.1016/j.tcs.2010.12.054
Dean, J. D., and Blackard, J. A. (1998). Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types, Ft Collins, CO: Colorado State University. doi: 10.5555/928509
Dua, D., and Graff, C. (2017). UCI machine learning repository. Available online at: https://archive.ics.uci.edu/ml/citation_policy.html
Duarte, M. F., and Hu, Y. H. (2004). Vehicle classification in distributed sensor networks. J. Parallel Distrib. Comput. 64, 826–838. doi: 10.1016/j.jpdc.2004.03.020
Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. Ann. Stat. 32, 407–499. doi: 10.1214/009053604000000067
Falkner, S., Klein, A., and Hutter, F. (2018). "Bohb: Robust and efficient hyperparameter optimization at scale," in IJCM (Stockholm), 1436–1445.
Galibert, M. A., Snoek, J., and Adams, R. P. (2014). "Bayesian optimization with unknown constraints," in Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI'14 (Arlington, VI: AUAI Press), 250–259.
Gelfand, S. B., and Mitter, S. K. (1989). Simulated annealing with noisy or imprecise energy measurements. J. Optim. Theory Appl. 62, 49–62.
Goodman, B., and Flaxman, S. (2017). European union regulations on algorithmic decision-making and a ‘right to explanation’. AI Mag. 38, 50–57. doi: 10.1016/j.aimag.2016.10.004
Grill, J.-B., Valko, M., Munos, R., and Munos, R. (2015). "Black-box optimization of noisy functions with unknown smoothness," in Advances in Neural Information Processing Systems 28, eds C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Montréal, QC: Curran Associates Inc.), 667–675.
Gutjahr, W. J., and Pflug, G. C. (1996). Simulated annealing for noisy or imprecise measurement functions. J. Glob. Optim. 8, 1–13.
Hansen, N., and Kern, S. (2004). "Evaluating the CMA evolution strategy on multimodal test functions," in Parallel Problem Solving From Nature PPSN VIII, Vol. 3242 of LNCS, eds X. Yao et al. (Berlin; Heidelberg: Springer), 282–291. Available online at: https://link.springer.com/chapter/10.1007/978-3-540-30217-9_29#citeas
Hanssen, N., and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. Evol. Comput. 9, 159–195. doi: 10.1162/106365601750190398
Hastie, T., Tibshirani, R., and Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2 Edn. Springer.
Hernández-Lobato, J. M., Gelbart, M. A., Adams, R. P., Hoffman, M. W., and Ghahramani, Z. (2016). A general framework for constrained bayesian models.

SUPPLEMENTARY MATERIAL
The Supplementary Material for this article can be found online at: https://www.frontiersin.org/articles/10.3389/frai.2020.00003/full#supplementary-material
Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). Anchors: high-precision model-agnostic explanations. Available online at: https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982
Rodriguez, J. J., Kuncheva, L. I., and Alonso, C. J. (2006). Rotation forest: a new classifier ensemble method. IEEE Trans. Pattern Anal. Mach. Intell. 28, 1619–1630.
Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Ratra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 618–626.
Settles, B. (2009). Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin, Madison, WI.
Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: a review of Bayesian optimization. Proc. IEEE 104, 148–175. doi: 10.1109/JPROC.2015.2494218
Snoek, J., Larochelle, H., and Adams, R. P. (2012). “Practical bayesian optimization of machine learning algorithms,” in Advances in Neural Information Processing Systems 25, eds F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Lake Tahoe, NV: Curran Associates, Inc.), 2951–2959.
Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., et al. (2015). “Scalable bayesian optimization using deep neural networks,” in Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37, ICML’15, 2171–2180.
Ustun, B., and Rudin, C. (2016). Supersparse linear integer models for optimized medical scoring systems. Mach. Learn. 102, 349–391. doi: 10.1007/s10994-015-5528-6
Uzilov, A. V., Keegan, J. M., and Mathews, D. H. (2006). Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. BMC Bioinformatics 7:173. doi: 10.1186/1471-2105-7-173
Wang, C.-C., Tan, K. L., Chen, C.-T., Lin, Y.-H., Keerthi, S. S., Mahajan, D., et al. (2014). Distributed newton methods for deep neural networks. Neural Comput. 30, 1673–1724. doi: 10.1162/neco_a_01088
Wang, Z., Zoghi, M., Hutter, F., Matheson, D., and De Freitas, N. (2013). “Bayesian optimization in high dimensions via random embeddings,” in Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI ’13 (Beijing: AAAI Press), 1778–1784.
Wu, Z., and Palmer, M. (1994). “Verbs semantics and lexical selection,” in Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics, ACL ’94 (Stroudsburg, PA: Association for Computational Linguistics), 133–138.

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Copyright © 2020 Ghose and Ravindran. This is an open-access article distributed under the terms of the Creative Commons Attribution License (CC BY). The use, distribution or reproduction in other forums is permitted, provided the original author(s) and the copyright owner(s) are credited and that the original publication in this journal is cited, in accordance with accepted academic practice. No use, distribution or reproduction is permitted which does not comply with these terms.