Interpretability using Compact Models

A Thesis

submitted by

ABHISHEK GHOSE

for the Award of the Degree

of

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

September 2022

THESIS CERTIFICATE

This is to certify that the thesis titled **Interpretability using Compact Models**, submitted by **Abhishek Ghose(CS15D004)**, to the Indian Institute of Technology, Madras, for the award of the degree of **Doctor of Philosophy**, is a bonafide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Balaraman Ravindran
Research Guide
Professor
Dept. of Computer Science
IIT-Madras, 600 036

Place: Chennai

Date: February 24, 2023

ACKNOWLEDGEMENTS

I have had the good fortune of being aided by multiple helping hands, or “latent variables” - as I would like to think of them, that have made the “generative process” of my PhD fruitful. I take this opportunity to thank them.

I would like to begin by thanking my advisor Dr. Balaraman Ravindran, who not only let me pick an uncommon problem and approach to investigate, but also was sympathetic to the many failures along the way. His critical evaluation of my ideas and the communication of them constituted significant learning for me, and his trust in me kept me motivated.

For all challenging undertakings in my life, I have always, albeit subconsciously at times, relied on the pillar of support that my parents and brother represent. This is true in this case as well: their faith in me led me to consider, and later join, this part-time PhD program, and their unwavering encouragement saw me through it.

I am indebted to Vijayalekshmi Kesavan, a good friend and then my wife, for her support: she was partner to many thoughtful conversations, and spent hours reviewing notes and drafts that I produced. Her motivation and support were especially crucial in the final stretch of the program.

I am grateful for the help I received from my friend Balasanjeevi B.. He readily took time out of his busy life to act as a sounding board for my ideas, often pointing out relevant literature to consult. This is humbling in light of the fact that he had played a similar role of being a friend and critique during my Masters as well!

I owe my gratitude to Dr. Shyam Rajagopalan, an ex-colleague and my friend, for inspiring me to consider a part-time PhD. His spirit of tinkering with ideas and projects convinced me of the feasibility of such an undertaking.

My team in [24]7.ai helped me in multiple small and large ways, but I would like to especially mention Dr. Ravi Vijayaraghavan, Dr. Prashant Joshi, Dr. Abir Chakraborty,

Dr. Paul Sauer, Dr. Cosimo Spera and Dr. Mandar Mutalikdesai for their support; it's challenging to work on a PhD along with a day-job, but their encouragement and support, both logistic and moral, eased the process.

Working on a program remotely often led to multiple administrative challenges; and I would like to thank Tarun Kumar and Shreyas Shetty for helping me tide over them. I am especially grateful to Shreyas for being available for multiple deeply technical discussions in the early stages of the program.

ABSTRACT

KEYWORDS: Machine Learning, Interpretability

The increased use of Machine Learning in various real world systems has led to a corresponding increased need for models to be understandable, either by being *interpretable* or *explainable*. We focus on interpretability in our work, noting that it is preferred for models to be small in size for them to be considered interpretable, e.g., a decision tree of depth 5 is easier to interpret than one of depth 50.

Smaller models also tend to have high bias. This suggests an inherent trade-off between interpretability and accuracy. Our work addresses this by proposing techniques to create compact models: small-sized models that minimize the difference in accuracy relative to their larger counterparts.

We propose two model agnostic techniques to construct such models. Both of them operate on the assumption that focusing on specific instances during model training potentially leads to increased model accuracy. The task of identifying such instances and their relative influence is formulated as that of learning a sampling distribution. The distribution is represented as an infinite Beta mixture model, and its parameters are learned using an optimization procedure that maximizes held-out accuracy.

The approaches differ in the representation of the optimization search space: in the first case, referred to as the density tree based approach, information about proximity of instances to class boundaries is used. This information is derived from the input space partitioning produced by decision trees. The second technique, referred to as the oracle based approach, utilizes the uncertainty in predictions of an oracle model.

Rigorous empirical validation of the proposed techniques is presented that uses multiple real world datasets and interpretable models with various notions of model size. We observe statistically significant relative improvements in the F1-score - occasionally greater than 100% - between a model of a given size trained in a standard manner, and

a compacted version of the same size created by our techniques. As a corollary, we challenge the conventional wisdom that train and test data need to be drawn from the same distribution for optimal learning, instead showing that this is not true when model sizes are small.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

BO	Bayesian Optimization
CART	Classification and Regression Trees
DP	Dirichlet Process
DT	Decision Tree
GBM	Gradient Boosting Machine
IBMM	Infinite Beta Mixture Model
IGMM	Infinite Gaussian Mixture Model
KDE	Kernel Density Estimator
LPM	Linear Probability Model
RF	Random Forest
SVM	Support Vector Machine
TPE	Tree-structured Parzen Estimator
pdf	Probability Density Function
pmf	Probability Mass Function

NOTATION

Common notations are listed below. Notations specific to techniques may be found listed in respective chapters, e.g., Sections 3.1.4 and 4.1.4.

$\mathbb{R}$, $\mathbb{Z}$, $\mathbb{N}$	Sets of *reals*, *integers* and *natural numbers* respectively.
$X \in \mathbb{R}^{N \times d}$	Ordered collection of N d-dimensional feature vectors.
$Y \in \mathbb{R}^{N}$	Ordered collection of N labels.
$[A]_{pq}$	Element at the p^{th} row and q^{th} column indices of a matrix A.
η	Model size.
$depth(T)$	Depth of decision tree T.
$\mathcal{F}$	Model family, e.g., Decision Tree.

CHAPTER 1

Introduction

1.1 Understandable Models

In recent years, Machine Learning (ML) models have become increasingly pervasive in various real world systems. In many of these applications, such as movie and product recommendations, it is sufficient that an ML model is accurate. However, there is a growing emphasis on models and their predictions to be *understandable* as well, especially in domains where the cost of being wrong is prohibitively high, e.g., medicine and healthcare (Caruana *et al.*, 2015; Ustun and Rudin, 2016; Yoon *et al.*, 2018; Alaa and van der Schaar, 2019*a*), defence applications (Gunning, 2016; Zhang *et al.*, 2020*a*), law enforcement (Perry *et al.*, 2013; Angwin *et al.*, 2016; Larson *et al.*, 2016; Rudin *et al.*, 2020), self-driving cars (Kim and Canny, 2017; Kim *et al.*, 2018) and banking (Alloway, 2015; Castellanos and Nash, 2018).

In general, the ability to understand a model leads to:

1. Increased trust: the user of a model can trace model decisions back to easily understandable data artefacts, to convince themselves that the prediction rationale is plausible and robust, i.e., not too narrowly defined.

2. Ability to audit: this is important when transparency is required. For example, for an application rejected by a loan risk assessment system, the applicant might want to know the specific reasons that influenced the decision.

3. Ability to debug models: it is important to ensure that a model has not learned unintended associations within the data to produce a deceptive accuracy score. A common example is classifiers inadvertently using metadata as features, e.g., a text classifier using document headers (Ribeiro *et al.*, 2016) or an image classifier using the text in the image source tag (Lapuschkin *et al.*, 2019). Either kind of

metadata - document headers or image source tags - is unlikely to be present at the point of deployment.

Such issues may be easily rectified if the decision process of a model is open to inspection.

4. Discovery: aside from the common objective of prediction accuracy, the goal of a modeling exercise might be to gain insight into a phenomena. This is only feasible if a model is understandable. An example of this is analyzing amino acid interactions learned by protein models (Vig *et al.*, 2021) that are based on the *Transformer* (Vaswani *et al.*, 2017) architecture. Another example is studying pathogen exposure by analyzing implicit relationships learned by an ML model (Fountain-Jones *et al.*, 2019).

Recent interest in understandable models is also evidenced by the following factors:

1. Laws like the European Union's *General Data Protection Regulation (GDPR)* (Goodman and Flaxman, 2017) and France's *Digital Republic Act* mandate the "right to explanation" for a user, which necessitates some form of model transparency. Newer laws are expected to emphasize this as well, e.g., *Algorithmic Accountability Act* (Clarke, 2019).

2. Multiple recent tools support various forms of model analysis. Some examples are:

   (a) InterpretML from Microsoft. (Nori *et al.*, 2019).

   (b) AllenNLP Interpret, AllenNLP (Wallace *et al.*, 2019).

   (c) Language Interpretability Tool, Google (Tenney *et al.*, 2020).

   (d) AI Explainability 360, IBM (Arya *et al.*, 2020).

Today, there exists a rich body of discourse around the specification and role of understandable models (Doshi-Velez and Kim, 2017; Lipton, 2018; Miller, 2019; Barredo Arrieta *et al.*, 2020; Krishnan, 2020; Molnar, 2022), along with multiple user studies that assess how they influence human decision-making (Freitas, 2014; Lage *et al.*, 2019; Kaur *et al.*, 2020; Bhatt *et al.*, 2020; Poursabzi-Sangdeh *et al.*, 2021).

In essence, the past few years has seen the construction of understandable models establish itself as an important area of research in the ML community. Our work investigates a specific approach for building such models. Before we detail our contributions, we first consider the broader trends in the area in the next section.

1.2 Types of Understandable Models

While today there exist multiple techniques to make models understandable, these may be broadly categorized as either producing *interpretable* models or producing model *explanations*[1]:

1. *Interpretability*: this area looks at building models that are *inherently* interpretable, i.e., are considered easy to understand as-is[2]. Some examples of interpretable models are:

   (a) Rules (Quinlan, 1990; Friedman and Popescu, 2008), rule lists (Letham *et al.*, 2015; Angelino *et al.*, 2017), rule sets (Wang, 2018).

   (b) Decision trees (DT) (Breiman *et al.*, 1984; Quinlan, 1993, 2004; Hu *et al.*, 2019), decision sets (Lakkaraju *et al.*, 2016).

   (c) Sparse linear models (Efron *et al.*, 2004; Ustun and Rudin, 2016).

   (d) Linear or generalized additive models that include pairwise interactions (Lim and Hastie, 2015; Lou *et al.*, 2013; Wang *et al.*, 2021).

   An interpretable model may be specific to a task as well, e.g., rules for negation scope detection in natural language (Pröllochs *et al.*, 2019). The size of a model also plays a role, and is discussed in Section 1.3.

2. *Explainability*: this area looks at techniques that may be applied to models that do not naturally lend themselves to convenient interpretation, i.e., "black-box

[1] While these categories are commonly understood as defined here, there is some variance in how these terms are used (Lipton, 2018; Tjoa and Guan, 2020; Kaur *et al.*, 2020; Krishnan, 2020). For instance, what we consider to be interpretable models here may be referred to as "intrinsically/inherently interpretable" models, while model explanations may be seen as producing "post-hoc interpretability".

[2] See Barceló *et al.* (2020) for an interesting attempt to mathematically characterize the notion of interpretability in terms of the computational complexity of answering post-hoc queries.

models". *Convolutional Neural Networks (CNN)* and *Random Forests (RF)* are examples of such models. These techniques *approximate* a trained model either at the scale of individual predictions, or globally, by creating an understandable approximation of the black box model. Examples of such techniques are:

(a) Locally interpretable explanations as provided by LIME and Anchors (Ribeiro *et al.*, 2016, 2018).

(b) Feature attribution based on Shapley values (Lundberg and Lee, 2017; Ancona *et al.*, 2019), mutual information approximated by variational inference (Chen *et al.*, 2018) or visualized as *Partial Dependence Plots (PDP)* (Friedman, 2001).

(c) Visual explanations for Convolutional Neural Networks such as Grad-CAM (Selvaraju *et al.*, 2017) and Ablation-CAM (Desai and Ramaswamy, 2020).

(d) Text *rationales* or short excerpts from a text document that justify its label (Lei *et al.*, 2016).

(e) Approximation of a black-box model with an inherently interpretable model such as a symbolic model composed of a finite number of familiar functions (Alaa and van der Schaar, 2019*b*), decision trees (Craven and Shavlik, 1995; Di Castro and Bertini, 2019) or rules where the level of detail may be interactively adjusted (Ming *et al.*, 2019).

Each of these approaches have certain strengths and limitations. These are listed below.

1. Interpretability:

   - Strengths: The model is easily understood by inspection. As a by-product, it is easy to infer the reasoning behind individual predictions.
   - Limitations: In addition to optimizing for prediction accuracy, the model is constrained to a specific representation, e.g., rules, decision tree. Considering the general principle that unconstrained optimization produces results that are equivalent or better than constrained optimization, this might result

in relatively less accurate models. Effectively, we trade-off accuracy for representation[3].

As an example, note that the best performing NLP models today on the GLUE (Wang *et al.*, 2018*a*) and SuperGLUE (Wang *et al.*, 2019) benchmarks[4], are black-box models.

2. Explainability:

   - Strengths: The modeler enjoys the convenience of training a model for high accuracy, irrespective of its representation. An appropriate explanation technique may be identified later, among the several options available today. Model-agnostic techniques like LIME (Ribeiro *et al.*, 2016), SHAP (Lundberg and Lee, 2017), L2X (Chen *et al.*, 2018), INVASE (Yoon *et al.*, 2019) and SIPA (Scholbeck *et al.*, 2020) may be applied to arbitrary models. Model-specific techniques exist for common black-box models, e.g., DeepLIFT (Shrikumar *et al.*, 2017) for Neural Networks, TreeSHAP (Lundberg *et al.*, 2018) and LeafInfluence (Sharchilev *et al.*, 2018) for decision tree ensembles, saliency maps for CNNs (Simonyan *et al.*, 2014).

   - Limitations: Being a model approximation, an explanation is a "model of a model"; thus, care must be taken to ensure high *fidelity*[5] with the underlying black-box model.

     As examples of approximation errors that arise in practice, consider perturbation based explanations as used by LIME and SHAP: here, explanations may be unstable: neighboring instances might generate disparate explanations (Alvarez-Melis and Jaakkola, 2018), or may be inconsistent: it is possible to generate different explanations for the same prediction (Lee *et al.*, 2019; Slack *et al.*, 2021), or use of potentially out-of-distribution data (generated via perturbation) might expose the explanation technique to adversarial attacks (Slack *et al.*, 2020).

     For a general discussion of advantages of interpretability over model expla-

[3] See Bertsimas *et al.* (2019) and Dziugaite *et al.* (2020) for formalizations of this idea.

[4] The respective leaderboards are available at `https://gluebenchmark.com/leaderboard` and `https://super.gluebenchmark.com/leaderboard/`.

[5] Yeh *et al.* (2019) provides some objective measures of this quantity.

nations, see Rudin (2019).

It is easy to see that these approaches are complementary and are each useful in specific contexts. Not surprisingly, research and applications in each category continue to flourish. Our research focuses on techniques to create certain types of interpretable models - we describe this in the next section.

1.3 Interpretability using Compact Models

In the previous section, we mentioned that interpretable models conform to representations that are considered easy to understand, e.g., rules, trees, sparse linear models. Often, a common desirable property across such representations is a relatively small *model size*. Some anecdotal examples of this are:

1. A decision tree with $depth = 50$ is significantly harder to understand than one with $depth = 5$.

2. An ensemble of decision trees, as in a Random Forest or a Gradient Boosted model, is harder to reason about than a single decision tree.

3. A sparse linear model with 50 non-zero coefficients is harder to understand than one with 10 non-zero coefficients.

The relationship between interpretabiltiy and model size has been scientifically studied as well :

- User studies indicate that small model size is one of a few important factors that makes a model interpretable: Lage *et al.* (2019) show in the context of decision sets that small model sizes aid interpretability (although its not the most important property do so); Poursabzi-Sangdeh *et al.* (2021) find that smaller model sizes aid in certain tasks that require a human subject to have understood how a model works; Feldman (2000) notes that longer Boolean formulae are harder to learn by humans.

While model size is important, Kulesza *et al.* (2013) caution against focusing on size in isolation, arguing smaller model sizes can be detrimental to understanding if they are too simplistic. Freitas (2014) highlights this aspect as well.

- This role of model size is variously acknowledged in the design and analysis of interpretable models: Herman (2017) refers to this as low *explanation complexity*, this is seen as important for *simulability* - ease of simulating the reasoning process of a model by a human (Lipton, 2018; Murdoch *et al.*, 2019) - and is often listed as a desirable property in interpretable model representations. Some examples of the latter are:

  - Decision Trees with a small number of leaves are preferred for explaining clusters (Moshkovitz *et al.*, 2020; Laber *et al.*, 2021).
  - Decision Sets are preferred to be concise both in terms of number of rules and number of predicates per rule (Lakkaraju *et al.*, 2016)
  - A low number of terms with non-zero coefficients in linear models are preferred for intelligibility. This was one of the original motivations behind the *LASSO* technique (Tibshirani, 1996), and is a design constraint for local linear models in LIME (Ribeiro *et al.*, 2016).
  - Rule learners like *LIBRE* (Mita *et al.*, 2020) and *Bayesian Rule Lists* (Letham *et al.*, 2015) emphasize learning of a small number of compact rules.

Based on existing literature, we note that it is desirable for interpretable models to be small in size. This also is the aspect of interpretability our research explores: we provide a model-agnostic technique to improve the accuracy of a *model for a given size*. This technique may be used to produce *accurate models of small sizes*; which are likely to be easier to interpret than larger models within the same model family.

We term the models our techniques produce **compact models**, to indicate they are as accurate as larger models of the same model family.

1.4 Objectives of Thesis

The key objectives of the thesis are to propose techniques that:

1. Produce compact models.
2. Are *model-agnostic*, i.e., the techniques may be applied to create models of an arbitrary model family such as Decision Tree, Logistic Regression, etc.

1.5 Contributions of Thesis

The main contributions of the thesis are as follows:

1. We propose two model-agnostic techniques to create compact models. The fundamental approach we adopt is to *learn* a distribution over the training data, such that a model trained on a sample drawn using it maximizes test accuracy. Figure 1.1 compares the generalizations learned by a model when training on the original distribution vs the distribution produced by our technique. We observe the latter strategy obtains a test accuracy of $F1 = 0.72$ compared to $F1 = 0.64$ of the former.

   The proposed techniques differ in their implementation of this principle:

   (a) In our first technique, we partition the input space into regions based on their proximity to class boundaries, and we propose an algorithm to efficiently sample from these partitions. The partitioning is realized by specialized decision trees we introduce, called *density trees*. We refer to this approach as the *density tree based approach*. The distribution in Figure 1.1(c) was created using this technique.

   (b) As an extension, we develop a second technique where the need for density trees is obviated by the use of a powerful probabilistic classifier, that we refer to as the *oracle*. The uncertainty in the oracle's predictions is used to learn the sampling distribution. We refer to this technique as the *oracle based approach*.

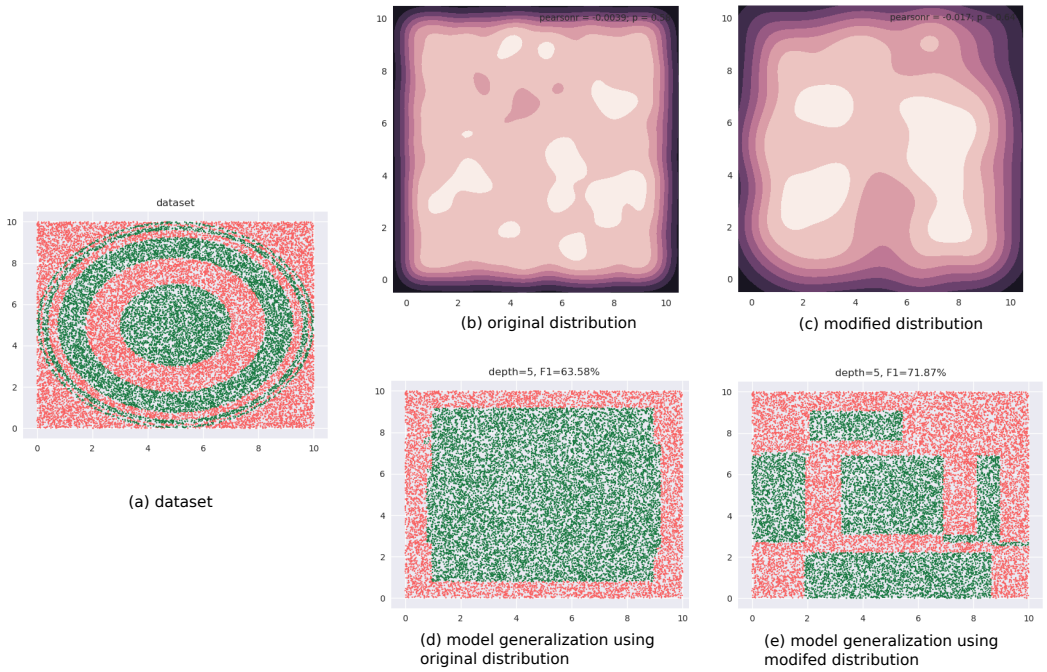


Figure 1.1: Changing the input distribution can significantly affect model accuracy. (a) shows a toy dataset. (b) shows the original distribution of the data as a *Kernel Density Estimation* plot and (d) visualizes the generalization learned by a Decision Tree of $depth = 5$ when using this distribution for training. (c) shows visualizes the sampling distribution learned by our technique and (e) shows the corresponding generalization learned by a Decision Tree with $depth = 5$. The latter achieves a higher accuracy of $F1 = 0.72$, compared to $F1 = 0.64$ of the former.

2. Both techniques cast the task of identifying the optimal training distribution into an optimization problem; however, in our formulation only a fixed number of optimization variables[6] are required irrespective of the dimensionality of the data.

3. We present the counter-intuitive result that for small models, the highest test accuracy is often seen with a training distribution that is *different* from the test distribution. This challenges the conventional wisdom that train and test distributions need to be identical for effective learning.

   We also demonstrate that as model size increases, the optimal training distribution progressively approximates the test distribution.

4. Rigorous empirical validation is provided to establish the effectiveness of our techniques.

[6]Eight and seven optimization variables are required by the density tree and oracle based approaches respectively.

1.6 Previous Work

While there is precedent to using different train and test distributions, such as when there is class imbalance in the data (Japkowicz and Stephen, 2002; Chawla *et al.*, 2002; He *et al.*, 2008; Santhiappan *et al.*, 2018), we are not aware of any work that applies the principle to learn size-constrained models. The following list compares and contrasts various techniques that possess some similarity with ours:

1. **Knowledge Distillation (KD)**: KD looks at using powerful "teacher" models (similar to our oracle) to learn a smaller "student" model (Gou *et al.*, 2021). The key differences with KD are:

   (a) Unlike KD our goal is not to approximate the oracle's performance. Instead, we evolve the smaller model towards a more accurate version. In fact, we ignore the oracle's label assignments entirely. This is in contrast to KD methods that may use teacher-assigned labels (Bucilă *et al.*, 2006) or distribution of label confidences (Hinton *et al.*, 2015)[7], or in general, focus on extracting "dark knowledge" from the oracle in some form.

   (b) A lot of KD research focuses on Neural Networks, e.g., *FitNets* (Romero *et al.*, 2015), *DistilBERT* (Sanh *et al.*, 2019), *ProfWeight* (Dhurandhar *et al.*, 2018). In contrast, our technique is model-agnostic.

   (c) Methodological differences aside, our observations suggest that the oracle might only be required to make our algorithm computationally efficient (discussed in Chapter 5).

2. **Explainable AI (XAI)**: Although we use an oracle model, the goal is not to explain its predictions. Consequently, we place much weaker demands of the information shared by the oracle than in XAI algorithms such as *LIME* (Ribeiro *et al.*, 2016), *SHAP* (Lundberg and Lee, 2017), *TREPAN* (Craven and Shavlik, 1995) or *GlocalX* (Setzu *et al.*, 2021):

   (a) A fundamental difference is we don't request for the predicted labels of the oracle.

[7] While we use the uncertainty in the oracle's prediction, note that we don't know which labels is the oracle more or less uncertain about, i.e., we ignore label *identity*.

(b) As an extension of the previous point, we ignore *fidelity* to the oracle's predictions, which is explicitly enforced and measured in XAI techniques. This precludes any deliberate alignment between the logical process the oracle follows to classify an instance vs that followed by the interpretable model.

(c) The information from the oracle is a one-time input to the algorithm, and is restricted to the training instances available to the interpretable model[8]. This is different from queries on synthetic instances performed by various XAI techniques to determine the neighborhood of a data instance, e.g., perturbed instances in case of *GLocalX*[9], *LIME*, *SHAP*, or generated random instances within constrained regions of the input space in *TREPAN*.

In short, we do not infer explanations, local or global, from the oracle. It is used to provide a one-dimensional view of the training data, as a one-time input to our algorithm; this is not equivalent to explaining the oracle. As mentioned earlier, this work suggests that the oracle might be only needed to provide reasonable computational complexity (discussed in Chapter 5).

3. **Active Learning**: In the case of active learning too, a predictive model maybe learned on a distribution that is different from the test distribution. However, some significant differences are:

   (a) Active learning works in the setting where only some or none of the labels of the training data are initially known, and there is an explicit label acquisition cost. We work within the traditional supervised setting where labels of all training instances are known.

   (b) The goal of an active learner is to minimize the total label acquisition cost, while being as accurate as a supervised learner that has access to complete label information. This is very different from our goal of performing *better* than a supervised learner, especially when the model size is small, assuming complete label information.

It must be noted that the term "oracle" in the active learning literature might refer

[8]The oracle only sees the training data available to the interpretable model, and is required to provide uncertainty values for these instances.

[9]GlocalX indirectly does this via the local learner it uses, *LORE* (Guidotti *et al.*, 2019).

to either a model or a human labeler; in our work, it exclusively refers to a model.

4. **Transfer Learning**: Transfer learning informs the training process of a "target" learner, given a "source" learner (Torrey and Shavlik, 2009; Pan and Yang, 2010; Weiss *et al.*, 2016). Our second technique is ostensibly similar as we have an oracle (our source learner) informing the interpretable model (our target learner). However, here are some key differences:

   (a) The typical application of transfer learning is in settings where the source learner has access to more data than the model it must transfer knowledge to; here transfer learning is seen as a way to overcome the data shortage by directly having the source learner convey knowledge, in some form, to the target model. This is different from our setting where the same data is available to both the oracle and the interpretable model.

   (b) Transfer learning techniques usually make some assumptions about the model family. Some examples are Boolean concepts (Thrun and Mitchell, 1994), Markov Logic Networks (Mihalkova and Mooney, 2006) or task-specific neural networks like *BERT* (Devlin *et al.*, 2019) or *ULMFiT* (Howard and Ruder, 2018) for Natural Language Processing, and *VGG networks* (Simonyan and Zisserman, 2015) for image recognition. In comparison, our technique is model agnostic, both w.r.t. the oracle and the interpretable model.

   (c) Although instance re-weighting techniques have been investigated as a means of transfer learning[10], their objective is to perform effective learning in situations where the data distribution available in the source task/domain is different from that in the target task/domain (Liao *et al.*, 2005; Dai *et al.*, 2007; Kamishima *et al.*, 2009). In our case, these two distributions, as provided, are identical; we *choose* to use a different training distribution in the interest of improving accuracy.

5. **Identifying Coresets**: Coreset construction techniques (Bachem *et al.*, 2017; Munteanu and Schwiegelshohn, 2018) seek to create a "summary" weighted sample of a dataset, known as a *coreset*, with the property that a model learned on

[10] We specifically mention this since instance re-weighting maybe seen as a form of sampling.

this dataset approximates one learned on the complete dataset. Some critical difference in our objective are:

(a) We want to reduce model size, not data size. In fact, the techniques we propose may *increase* training data size, as they sample with replacement (the maximum size can be controlled by the user).

(b) The primary motivation behind identifying a coreset is to avoid training on the entire dataset. But our goal is to allow models to make task-specific selection of training instances. They have unrestricted access to the entire dataset. They may train on it (or samples of similar size) possibly multiple times, based on the optimization trajectory, which contradicts the goal of coreset extraction.

This is not to say that the tools of analysis from the areas listed above cannot be adapted to our objective; but they do not directly solve for it today.

1.7 Outline of the Thesis

In this chapter, we provided an overview of the general problem of model interpretability (Sections 1.1, 1.2 and 1.3), the scope of the problem that is of interest to us (Section 1.4), the contributions of this work (Section 1.5) and how they relate to prior art (Section 1.6). The further chapters are organized thus:

- In Chapter 2, *Background*, we present an overview of some of the mathematical tools that are used in our techniques: *Bayesian Optimization* and mixture models based on the *Dirichlet Process*.

- In Chapter 3, *Compact Models using Density Trees*, we describe the density tree based approach, and provide empirical validation of its effectiveness.

- Chapter 4, *Compact using Probabilistic Oracles*, discusses the oracle based approach, also providing rigorous empirical validation of its effectiveness. We also compare the oracle and density tree based approaches in this chapter.

- Finally, Chapter 5, *Conclusions and Future Work*, concludes the thesis with a summary of our contributions, discussion on applications of our techniques beyond interpretability, and directions for future research.

CHAPTER 2

Background

In this chapter, we present an overview of some of the mathematical tools we utilize in our research. Of key relevance are the following:

1. *Bayesian Optimization*, discussed in Section 2.1.
2. *Dirichlet Process Mixture Models* in Section 2.2.

The final section in this chapter, Section 2.3, discusses the relevance of these techniques to our work.

2.1 Bayesian Optimization

2.1.1 Overview

We are interested in the following optimization problem:

$$\arg\min_{x \in \mathcal{X}} f(x) \tag{2.1}$$

where[1]:

1. The mathematical expression for $f(x)$ is unavailable to us.
2. We are allowed to evaluate $f(x)$ at any value x.
3. $f(x)$ might be noisy.
4. $f(x)$ might be expensive to evaluate.

[1] Of course, all discussion here applies to the analogous case of $\arg\max_{x \in \mathcal{X}} f(x)$

Such problems frequently arise in the area of *hyperparameter optimization (HO)* (Hutter *et al.*, 2019) and *Neural Architecture Search (NAS)* (Elsken *et al.*, 2019), where the search space $\mathcal{X}$ represents possible hyperparameter values or network configurations respectively, and $f(x)$ is a loss metric, like the RMSE, calculated on a validation dataset[2].

Let's briefly consider why HO is an example of the above problem. Since we want to find the best hyperparameters for an *arbitrary* classifier, f is not known, but for a given value x of the hyperparameters, we can obtain $f(x)$ by training the classifier. Training for different x is potentially a computationally expensive exercise. The performance of $f(x)$ may vary across multiple trials for the same x due to stochasticity in the learning algorithm, e.g., local search with different starting points in the case of neural networks or random sampling in the case of *Random Forests*.

The family of techniques referred to as *Bayesian Optimization (BO)* has emerged as an effective tool to solve such problems (Shahriari *et al.*, 2016; Springenberg *et al.*, 2016; Feurer and Hutter, 2019*a*; Ma *et al.*, 2019; White *et al.*, 2021). It circumvents the issue of an unknown f by learning a *surrogate* function that approximates it, which is constructed over multiple evaluations $f(x)$ at different x. The optimization is then performed over the surrogate. The two key ingredients in a BO algorithm are:

1. A surrogate function $\hat{f}$. An important property required is it should be able to provide a distribution over various possible predictions: $p(\hat{f}(x) = y|x), \forall y \in range(f), \forall x \in \mathcal{X}$. $\hat{f}$ becomes a better approximation of f as the latter gets evaluated over multiple iterations. A popular example of $\hat{f}$ is the *Gaussian Process* (Shahriari *et al.*, 2016; Hutter *et al.*, 2019), although other representations may be used, e.g., *Kernel Density Estimation* (Bergstra *et al.*, 2011, 2013), *Deep Neural Networks* (Snoek *et al.*, 2015).

2. An *acquisition* function A. This proposes the next most promising point to evaluate f at, given the current realization of $\hat{f}$. The proposal point is given by $\arg\max_{x \in \mathcal{X}} A(x, \hat{f})$. This choice is made based on the trade-off between the following strategies:

[2]We emphasize here that $\mathcal{X}$ does *not* denote the input data space. The dataset is fixed for an HO task: we seek the best hyperparameters *given* a problem or a dataset.

- *Exploration*: evaluate f at regions in $\mathcal{X}$ for which there doesn't exist enough information yet, since the minima might potentially exist in such a region.
- *Exploitation*: evaluate in the neighborhood of the minimum seen so far, since it might be seen as an indicator of the global minima existing in close proximity.

The acquisition function quantifies this trade-off. It relies on the probabilistic nature of the surrogate function to precisely account for the current quality of approximation. A popular example of A is the *Expected Improvement (EI)*:

$$EI(x) = E_y[\max(y^* - y, 0)] \tag{2.2}$$

$$= \int_y \max(y^* - y, 0) p(\hat{f}(x) = y|x) dy \tag{2.3}$$

Here y^* denotes the minimum thus far in the iterations. $EI(x)$ computes the expected value of the utility function $\max(y^* - y, 0)$, which measures the extent by which the current minimum might be improved upon at a given x (negative improvements are set to 0). Some other acquisition functions are *Probability of Improvement*, *Upper Confidence Bound*, *Entropy Search* and *Thompson Sampling* (see Shahriari *et al.* (2016) for a survey of common acquisition functions).

Algorithm 1 shows the execution steps for a BO algorithm, for an optimization budget of T iterations. Here, $\mathcal{U}(\mathcal{X})$ denotes an uniform distribution over the search space $\mathcal{X}$ and $update(\hat{f}_{t-1}, \mathcal{H})$ represents the mechanism of update for the surrogate $\hat{f}_t$, based on the past estimate and history of evaluations of f.

An important point to note is that finding the next best proposal point x also requires an optimization problem to be solved (line 7 in Algorithm 1). For this purpose, BO algorithms typically use an *auxiliary optimizer*, e.g., the *CMA-ES* (Hansen and Ostermeier, 2001) optimizer is used in Wang *et al.* (2013), while Eric *et al.* (2008) use *DIRECT* (Jones *et al.*, 1993). It is important for the auxiliary optimizer to be fast (since it is invoked per iteration) as well as accurate (the convergence of the BO depends on the quality of proposal points). The possibility of entirely eliminating this optimization step has also been explored (Wang *et al.*, 2014; Merrill *et al.*, 2021).

Algorithm 1: General BO framework.

Data: Function f, search space $\mathcal{X}$, optimization budget T
Result: Local minimizer $x^* \in \mathcal{X}$

$\mathcal{H} \leftarrow \{\}$
for $t \leftarrow 1$ **to** T **do**
  if $t = 1$ **then**
    $x_1 \sim \mathcal{U}(\mathcal{X}), y_1 \leftarrow f(x_1)$
    $x^* \leftarrow x_1, y^* \leftarrow y_1$
  else
    $x_t \leftarrow \arg\max_{x \in \mathcal{X}} A(x, \hat{f}_{t-1})$
    $y_t \leftarrow f(x_t)$
    if $y_t < y^*$ **then**
      $x^* \leftarrow x_t, y^* \leftarrow y_t$
  end
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(x_t, y_t)\}$
  $\hat{f}_t \leftarrow update(\hat{f}_{t-1}, \mathcal{H})$
end
return x^*

Before we conclude this section, we provide the following intuitive visualizations to further illustrate the operation of a typical BO algorithm:

1. Figure 2.1 visualizes multiple iterations of a BO, over a univariate search space[3]. The surrogate model and the acquisition function used are a Gaussian Process regressor and Expected Improvement, respectively. Each subplot here corresponds to the state of information at the end of an iteration in the loop in Algorithm 1.

   Note the choice of the acquisition function at (b) and (c). At (b), it seems to exploit current information, and the new proposal point is very close to the initial point. However, at (c), it seems to be exploring and it selects a point in a high uncertainty region. As expected, the uncertainty decreases around regions where f has been evaluated. Over the iterations, we observe that $\hat{f}$ improves in its approximation of f, especially near the minima. This is a property of BO algorithms - the function tends to be approximated better near its minima relative to other regions.

2. It is informative to compare BO against a gradient based algorithm like *Gradient*

[3]Library used: modAL (Danka and Horvath, 2018).

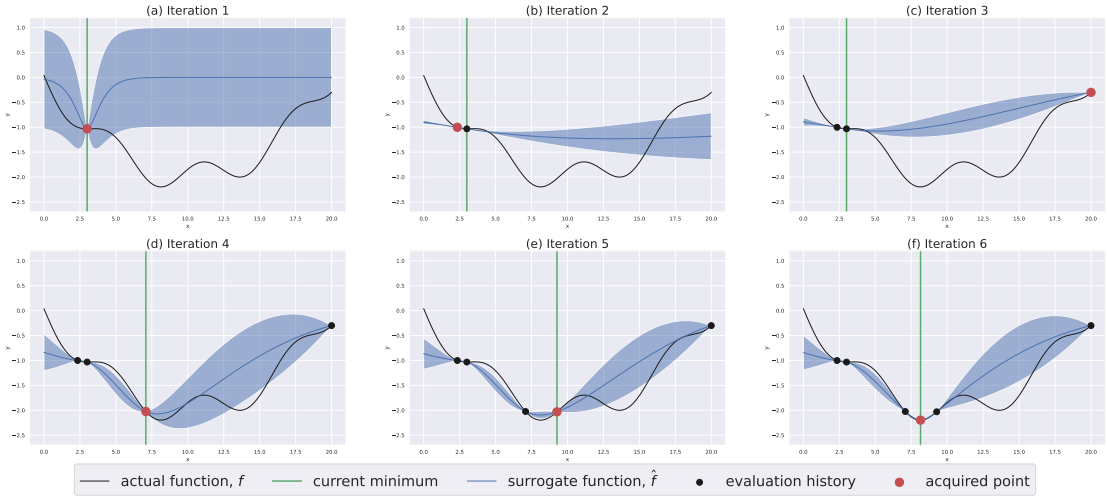


Figure 2.1: Multiple iterations of a BO are shown. The surrogate model's uncertainty is shown by an area bounded by one standard deviation of its estimate around its mean prediction. Wide areas indicate high uncertainty. Note how over multiple iterations: (a) the model uncertainty decreases and $\hat{f}$ becomes a better fit to f, especially near the global minima, and (b) the estimate of the minimizer (vertical green line) keeps improving.

Descent (GD)[4]. Figure 2.2 shows such a comparison[5]. Here, plots (a) and (b) show how BO and GD, respectively, minimize a function with multiple minima[6]. Only the actual function, f, is shown. The red points show where f was evaluated: later iterations are indicated by both the darker shades and larger sizes of the points. Both algorithms were provided the same starting point. The bottom row, (c) and (d), show a *Kernel Density Estimation (KDE)* plot constructed over points x for which a value of $f(x)$ was requested by the BO and GD algorithms respectively. This illustrates where these algorithms "focus their attention".

We see the following patterns: (a) The BO initially explores the entire space, as seen by the locations of the smaller/lightly-shaded points. Over time, the focus shifts to the neighborhood of the minima. This correlates with our observation in Figure 2.1 that the region near the minima is approximated better relative to other regions. The KDE shows a peak at the minima, but there is a non-trivial density in the other regions of the search space as well. (b) In contrast, the GD is narrowly focused on the minima closest to the starting point. Consequently, we see a sharp peak at this minima in the KDE, and no density outside of this neighborhood.

[4]While we use a simple batch GD here for purposes of illustration, in practice, robust versions, e.g., *momentum*-based GD, *stochastic* GD, *multi-start* GD, are typically used.

[5]Library used: Hyperopt (James Bergstra *et al.*, 2013). This is used in our research as well.

[6]This is the one-dimensional version of the *Ackley function* (Ackley, 1987).

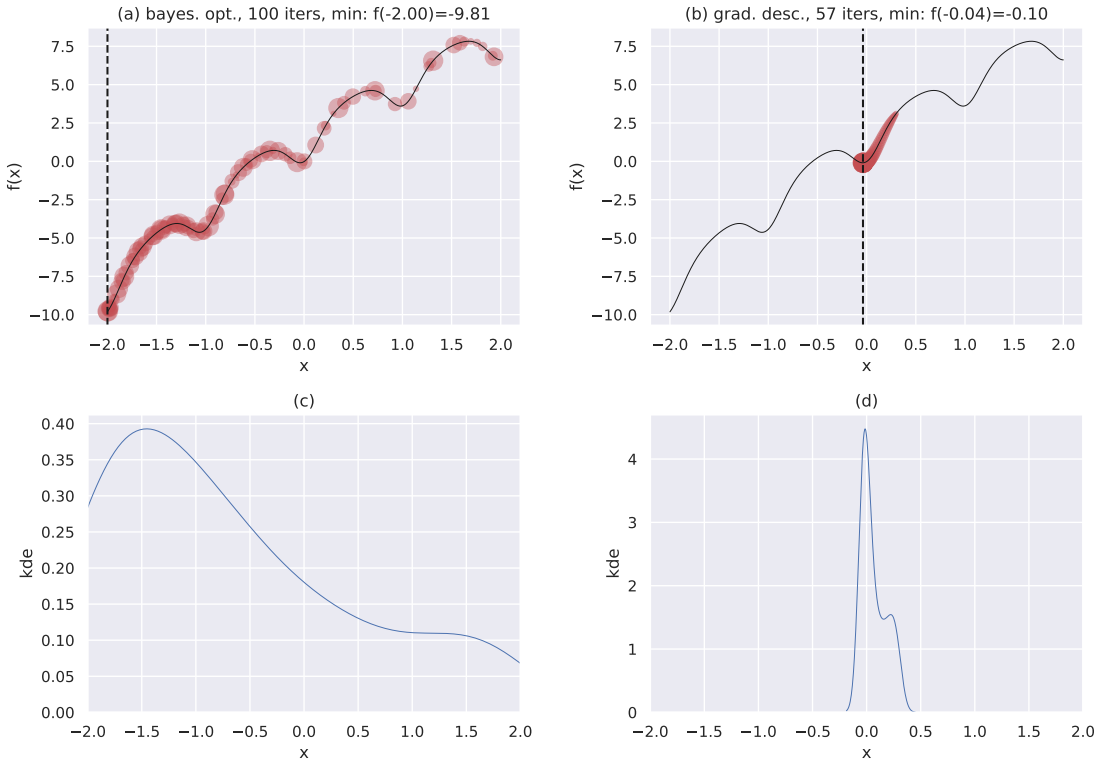


Figure 2.2: This figure shows how BO and GD differ in their strategies for finding the global minima. Plots (a) and (c) represent BO, while (b) and (d) represent GD. They are both provided the same initial point of $x = 0.3$. The minima found by the GD is strongly dependent on its starting point, and here it is shown stuck at a local minima. BO manages to find the global minima with exploration.

Having studied the general principles underlying BO algorithms, we now consider a specific BO algorithm: the *Tree-structured Parzen Estimator*. This is what we use in our research.

2.1.2 Tree-Structured Parzen Estimators

We briefly noted the importance of having a good auxiliary optimizer in the previous section. The *Tree-structured Parzen Estimator (TPE)* formulates the minimization problem in a way that makes fast computation of $EI(x)$ possible.

The primary difference between TPE and the standard BO approach (presented in the previous section) is that while the latter models $p(y|x)$ using a surrogate function[7],

[7]We use the shorthand $p(y|x)$ for $p(\hat{f}(x) = y|x)$. Also, we will assume $y \in (-\infty, \infty)$ in this section to simplify notation; the case of $y \in range(f)$ may be seen as a special case of this.

TPE models $p(x|y)$, and indirectly, $p(x)$. It stores the following densities:

$$p(x|y) = \begin{cases} l(x) \text{ if } y < y^* \\ g(x) \text{ if } y \geq y^* \end{cases} \tag{2.4}$$

These densities are learned over the observations from various evaluations of f, i.e. $y = f(x)$. We see the density $p(x|y)$ is divided up into a model of regions where the minima is likely to exist, $l(x)$, vs the rest, $g(x)$. Unlike Algorithm 1, we cannot have y^* to be the current minimum, since then there would be no points to construct $l(x)$. Instead, this is set to be a fixed quantile γ of observed values, i.e., $p(y < y^*) = \gamma$ where $\gamma \in (0, 1)$.

Let's consider the expression for $EI(x)$:

$$\begin{aligned} EI(x) &= \int_{-\infty}^{\infty} \max(y^* - y, 0) p(y|x) dy \\ &= \int_{-\infty}^{y^*} (y^* - y) p(y|x) dy + \int_{y^*}^{\infty} 0 p(y|x) dy \\ &= \int_{-\infty}^{y^*} (y^* - y) \frac{p(x|y)p(y)}{p(x)} dy \end{aligned} \tag{2.5}$$

Let's consider the denominator. By construction $p(y < y^*) = \gamma$, which leads to:

$$\begin{aligned} p(x) &= \int_{-\infty}^{\infty} p(x|y)p(y)dy = \int_{-\infty}^{y^*} p(x|y)p(y)dy + \int_{y^*}^{\infty} p(x|y)p(y)dy \\ &= \int_{-\infty}^{y^*} l(x)p(y)dy + \int_{y^*}^{\infty} g(x)p(y)dy \\ &= l(x) \int_{-\infty}^{y^*} p(y)dy + g(x) \int_{y^*}^{\infty} p(y)dy \\ &= \gamma l(x) + (1 - \gamma)g(x) \end{aligned} \tag{2.6}$$

Since the denominator in Equation 2.5 doesn't depend on y, the numerator may be

independently integrated:

$$\begin{aligned}\int_{-\infty}^{y^*}(y^*-y)p(x|y)p(y)dy &= \int_{-\infty}^{y^*}(y^*-y)l(x)p(y)dy \\ &= y^*l(x)\int_{-\infty}^{y^*}p(y)dy - l(x)\int_{-\infty}^{y^*}yp(y)dy \\ &= \gamma y^* l(x) - l(x)\int_{-\infty}^{y^*}yp(y)dy \end{aligned} \tag{2.7}$$

Substituting the expressions from Equation 2.7 and Equation 2.6 for the numerator and denominator respectively, into $EI(x)$ from Equation 2.5, we have:

$$EI(x) = \frac{\gamma y^* l(x) - l(x)\int_{-\infty}^{y^*}yp(y)dy}{\gamma l(x) + (1-\gamma)g(x)} = \frac{\gamma y^* - \int_{-\infty}^{y^*}yp(y)dy}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}} \tag{2.8}$$

Recall $\gamma \in (0,1)$ is a constant, and given that at an iteration, y^* is fixed, the only dependence of $EI(x)$ on x comes from the denominator:

$$EI(x) \propto \left(\gamma + (1-\gamma)\frac{g(x)}{l(x)}\right)^{-1} \tag{2.9}$$

The last expression suggests a strategy for determining the next proposal point:

1. Sample multiple $x \sim l(x)$.
2. Rank them by the score $\frac{g(x)}{l(x)}$, and pick the one with the lowest score.

Thus, the challenge of optimizing the acquisition function is replaced by the computationally much cheaper alternative of sampling and scoring x. The TPE algorithm maintains an ordered list of observed y, with the densities $l(x)$ and $g(x)$ being modelled as Gaussian Kernel Density Estimators (also known as *Parzen Estimators*), when the search space of x is $\mathcal{U}([a,b])$. The list makes it convenient to identify x for constructing $l(x)$ or $g(x)$. A side-effect of using kernel density estimation is the model update mechanism - $update(\hat{f}_{t-1}, \mathcal{H})$ in Algorithm 1 - is also cheap: a new Gaussian is introduced at x to modify either $l(x)$ or $g(x)$, depending on $f(x) < y^*$ or $f(x) \geq y^*$ respectively.

For further details, such as support for other kinds of search spaces, including those that are hierarchical, see Bergstra *et al.* (2011, 2013). TPE has also been extended in

various ways, e.g., multi-objective optimization (Ozaki *et al.*, 2020), to be used in conjunction with another optimizer like *Hyperband* (Falkner *et al.*, 2018) and for accepting prior information (Souza *et al.*, 2021). The relationship in Equation 2.8, between EI and the "density ratio" $g(x)/l(x)$, is also utilized by the optimizer proposed in Tiao *et al.* (2021). A criticism of using density ratios in the form presented here appears in Song *et al.* (2022).

2.2 Dirichlet Process Mixture Models

In this section we discuss mixture models based on the *Dirichlet Process (DP)*. This is our preferred density representation framework in subsequent chapters. We introduce the DP by first describing a way to intuitively represent densities, then highlight its limitations and finally show how the DP solves for them.

2.2.1 Overview

A common problem in Machine Learning is to mathematically represent the distribution of data $X \subseteq \mathbb{R}^d$. A quantity of interest produced by such a model is $p(x)$, which denotes the probability of $x \in \mathbb{R}^d$ being generated[8] by the same distribution that generated X. We consider the *Gaussian Mixture Model (GMM)*, a popular example[9] of such a representation: this assumes the data is generated by one of K Gaussian distributions, resulting in $p(x) = \sum_{i=i}^{K} \rho_i \mathcal{N}(x|\mu_i, \Sigma_i)$. Here, ρ_i is the *mixing coefficient* of $\mathcal{N}(\mu_i, \Sigma_i)$, where $\sum_{i=i}^{K} \rho_i = 1$. It denotes the contribution of particular Gaussian to the overall distribution. The parameters of the model $\mu_i, \Sigma_i, \rho_i, \forall i \in \{1, 2, ..., K\}$, given data X and an assumed value for K, are commonly learned using the *Expectation Maximization (EM)* algorithm (Dempster *et al.*, 1977). Figure 2.3 shows an example: Figure 2.3(a) shows the data whose distribution we wish to model and Figure 2.3(b) visualizes the likelihood function for a GMM learned on the data, under the *assumption* that $K = 2$.

[8] Note here that $p(x)$ may be applied to any $x \in \mathbb{R}^d$, although the provided dataset X is a subset of $\mathbb{R}^d$.

[9] The GMM is also a well-studied representation, with one of the earliest known instances being Pearson (1894).

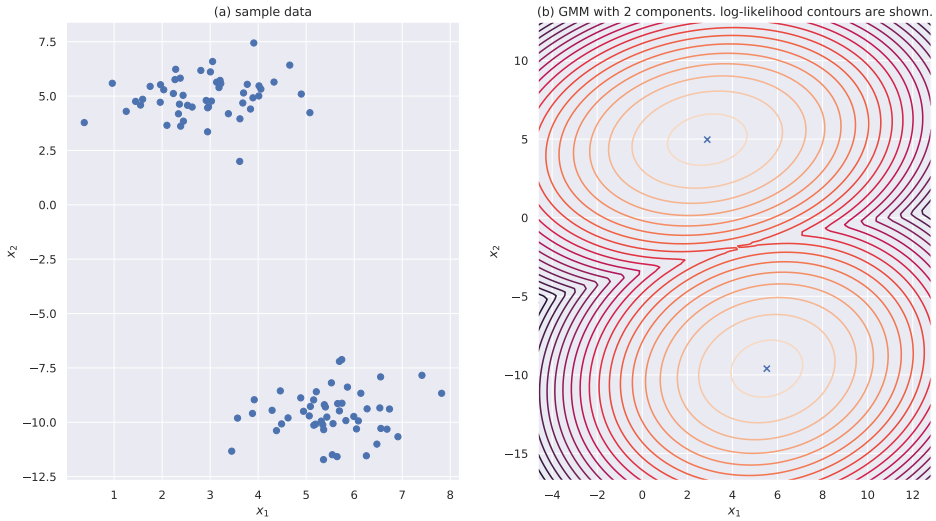


Figure 2.3: Generative modeling with GMMs: (a) shows sample data whose distribution we want to model (b) under the assumption $K = 2$, various model parameters are learned; we show the centers of the component Gaussians with blue crosses and contour lines for various values of the *log likelihood* in the input space.

The GMM illustrates an intuitive decomposition of the distribution modelling problem:

1. Assume the distribution is a result of multiple smaller data generation sources, e.g., the Gaussian components in the GMM.

2. The distribution is adequately described by: (a) the relative amount of data each of these sources generate, e.g., ρ_i, and (b) the properties of the components, e.g., μ_i and Σ_i above.

While the above decomposition is fairly general, a key challenge with the GMM is that K needs to be specified. In practice, GMMs for different values of K are learned, and the best model is picked based on metrics like the *Akaike* or *Bayesian Information Criteria*, or based on a task-specific metric. This still suffers from logistic challenges: (a) it requires an appropriate search space for K to be specified (b) this might be a computationally expensive approach depending on the size of data X and the search space for K.

An elegant approach that does not suffer from this limitation is the *Dirichlet Process (DP)*-based mixture model. The limitation of specifying K is obviated by allowing for an *infinite number of components* of which only a finite number, referred to as *clusters*, produce the observed data. Additionally, its a *Bayesian* approach: instead

of point estimates of model parameters, we reason about their distributions. We present an intuitive overview of this approach, by deconstructing it into the following logical steps:

1. Representing the probability masses of an infinite number of components using the **stick-breaking process** (Section 2.2.2).

2. Adding priors to define the properties of the infinite components (Section 2.2.3). This is the **Dirichlet Process**.

We elaborate these steps next.

2.2.2 The Stick-Breaking Process

Before we can express the ρ_k of an infinite number of components, let us consider the case of a *finite* number of components K in a Bayesian setup. A common prior used is the *Dirichlet* distribution, $Dir(\boldsymbol{\alpha})$. Here $\boldsymbol{\alpha}$, known as the *concentration parameter*, is vector with K components, with $0 < \alpha_k, \forall k \in \{1, 2, ..., K\}$. A sample from the distribution $\rho \sim Dir(\boldsymbol{\alpha})$ is also a vector with K components such that $0 \leq \rho_k \leq 1$ and $\sum_{k=1}^{K} \rho_k = 1$. Thus, a sample from the Dirichlet distribution may be seen as multinomial distribution. Figure 2.4 shows samples from Dirichlet distributions with $K = 3$, for different values of $\boldsymbol{\alpha}$.

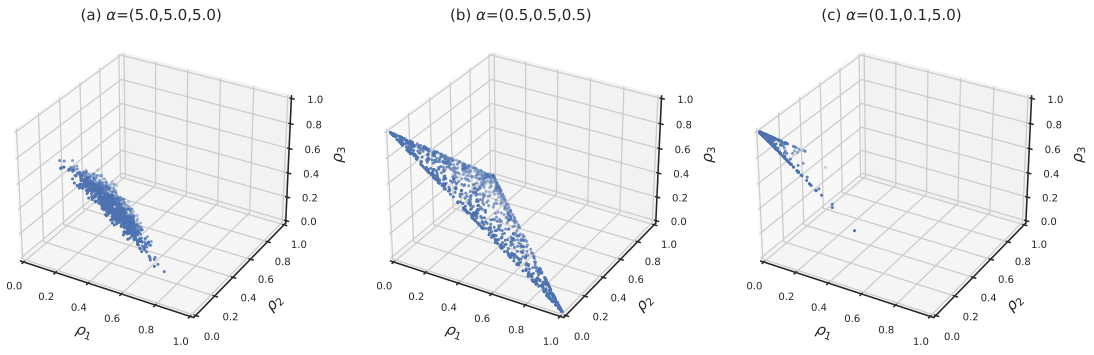


Figure 2.4: Samples from $Dir(\boldsymbol{\alpha})$ for different $\boldsymbol{\alpha}$ are shown as scatter plots. Note how the high equal values of α_i in (a) predominantly result in multinomial distributions that are close to the uniform distribution, while low equal values, as in (b), result in multinomials where the individual probabilities may take a wide range of values. In (c), we see that setting a relatively high α_3 leads to skewed mulitnomial distributions with most of the mass being concentrated in ρ_3.

What makes this distribution particularly useful here is $\rho_1, \rho_2, ..., \rho_K$ may be *sequentially* sampled. This utilizes its *neutrality* property:

$$\rho \sim Dir(\alpha) \implies \rho_1 \perp\!\!\!\perp (\rho_2, \rho_3, ..., \rho_K) \tag{2.10}$$

where,

$$\rho_1 \sim Beta(\alpha_1, \sum_{k=2}^{K} \alpha_k) \tag{2.11}$$

$$\Big(\frac{\rho_2}{1-\rho_1}, \frac{\rho_3}{1-\rho_1}, ..., \frac{\rho_K}{1-\rho_1}\Big) \sim Dir(\alpha_2, \alpha_3, ..., \alpha_K) \tag{2.12}$$

The above property suggests a recursive procedure to sample ρ:

1. Sample $V_1 \sim Beta(\alpha_1, \sum_{k=2}^{K} \alpha_k)$. Let $\rho_1 = V_1$.
2. Sample $V_2 \sim Beta(\alpha_2, \sum_{k=3}^{K} \alpha_k)$. From Equation 2.12, we know $V_2 = \frac{\rho_2}{1-V_1}$. This gives us $\rho_2 = (1-\rho_1)V_2 = (1-V_1)V_2$.
3. Sample $V_3 \sim Beta(\alpha_3, \sum_{k=4}^{K} \alpha_k)$. Extending Equation 2.12, $V_3 = \frac{\rho_3}{(1-V_1)(1-V_2)}$, and therefore, $\rho_3 = (1-V_1)(1-V_2)V_3$.

Note that at no point in the above process do we directly sample from the Dirichlet distribution. The above procedure may be simplified by formulating it as a *stick-breaking process*. We start with a stick of unit length, successively breaking off lengths ρ_k from one end, with V_k denoting the *fraction* of the remaining stick to be removed (recall the standard $Beta$ is supported on $[0, 1]$). For example, here are the first few steps corresponding to the procedure above:

1. We begin with a stick of unit length, of which the fraction $V_1 \sim Beta(\alpha_1, \sum_{k=2}^{K} \alpha_k)$ is to be removed. But since this is unit length, $\rho_1 = V_1$, where ρ_1 is the length removed.
2. We are now left with a stick of length $1 - V_1$ (since $\rho_1 = V_1$). The proportion to remove now is $V_2 \sim Beta(\alpha_2, \sum_{k=3}^{K} \alpha_k)$. The absolute length of the stick to remove is $\rho_2 = V_2(1 - V_1)$.
3. The length of the stick remaining is $1 - \rho_1 - \rho_2$, or equivalently, $1 - V_1 - (1 - V_1)V_2 = (1 - V_1)(1 - V_2)$. We now want to remove the fraction $V_3 \sim$

$Beta(\alpha_3, \sum_{k=4}^{K} \alpha_k)$. This is an absolute length of $\rho_3 = V_3(1 - V_1)(1 - V_2)$ for removal.

In general, for $k \neq K$,

$$\rho_k = V_k \prod_{i=1}^{k-1}(1 - V_i) \text{ where } V_k \sim Beta(\alpha_k, \sum_{i=k+1}^{K} \alpha_i) \tag{2.13}$$

We set $V_K = 1$ to ensure $\sum_{k=1}^{K} \rho_k = 1$ (Ishwaran and James, 2001), since:

$$1 - \sum_{k=1}^{K-1} \rho_k = \prod_{k=1}^{K-1}(1 - V_k) \tag{2.14}$$

$$\implies \sum_{k=1}^{K-1} \rho_k = 1 - \prod_{k=1}^{K-1}(1 - V_k) \tag{2.15}$$

$$\text{with } V_K = 1, \text{ from Equation 2.13, } \rho_K = \prod_{k=1}^{K-1}(1 - V_k) \tag{2.16}$$

$$\implies \sum_{k=1}^{K-1} \rho_k + \rho_K = 1 \tag{2.17}$$

This constructive sampling process hints at how we might treat infinite components; we sample sequentially as many components we require. The above process itself cannot be used since we still need to completely specify $\boldsymbol{\alpha}$ (in Equation 2.13) and the dimensionality of $\boldsymbol{\alpha}$ must match the number of components, i.e., be infinite. Essentially, we want the convenience of sequentially sampling ρ_k without having to specify an infinite-dimensional $\boldsymbol{\alpha}$.

The *Dirichlet* stick-breaking process (Sethuraman, 1994) offers a solution[10]. For the *scalar* parameter α, the sampling step is defined as:

$$\rho_k = V_k \prod_{i=1}^{k-1}(1 - V_i) \text{ where } V_k \sim Beta(1, \alpha) \tag{2.18}$$

This step may be repeated ad infinitum and is shown to result in multinomial distributions with *countably infinite* number of components, i.e., $\sum_{k=1}^{\infty} \rho_k = 1$ where $0 \leq \rho_k \leq$

[10]The "stick-breaking" metaphor refers to more than one sampling process (Ishwaran and James, 2001; Ren *et al.*, 2011), but this version (Sethuraman, 1994) is the one typically implied in the context of the Dirichlet Process.

1. Comparing to Equation 2.13, we see that the $Beta$ parameters are fixed at 1 and α, thus eliminating the challenges with using a vector $\boldsymbol{\alpha}$.

Samples ρ from this stick-breaking process are shown in Figure 2.5 for $\alpha = 0.3$ and $\alpha = 3$. Each row shows three samples of ρ for a given α. The x-axis shows $k \in \{1, 2, ..., 20\}$, while the y-axis denotes ρ_k.

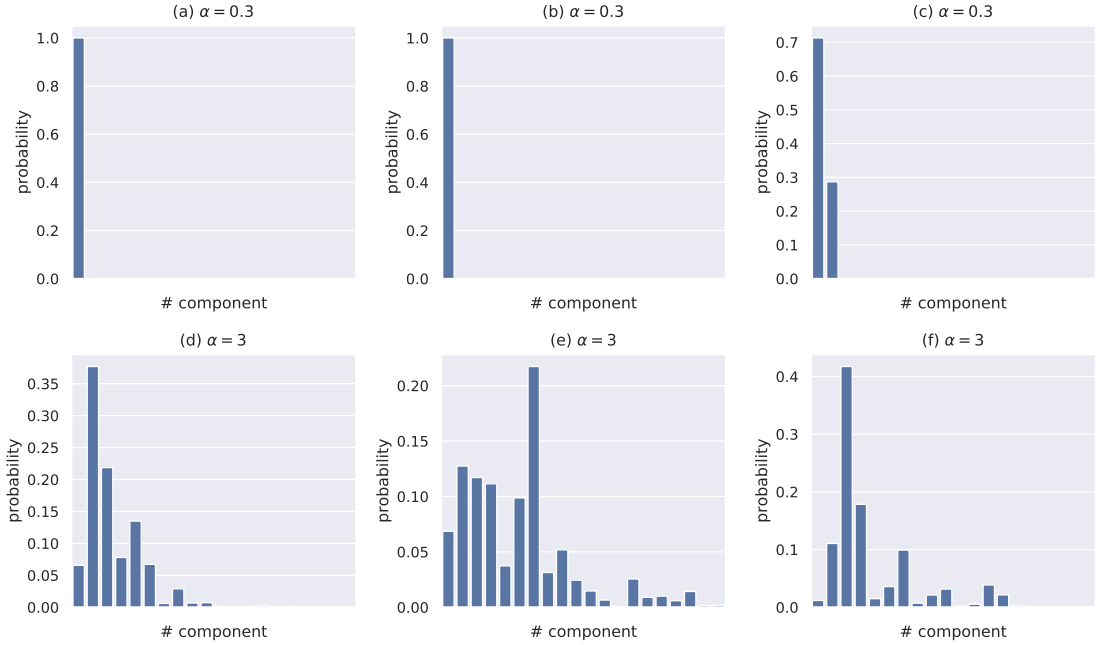


Figure 2.5: Sample probabilities obtained using the stick-breaking process are shown. (a), (b), (c) show sampled probabilities when $\alpha = 0.3$, and (d), (e), (f) show this for $\alpha = 3$. For larger values of α, relatively larger number of components get significant mass. The effect of decaying probabilities with components is universally observed for any α.

We notice as k increases, ρ_k diminishes. This is easily understood using the stick-breaking metaphor: as the stick gets smaller, the length of the stick that may now be removed, ρ_k, can only be small relative to the initial pieces removed. This probability masses are not perfectly ordered, as Figure 2.5 shows, but they inevitably approach 0. The rate of decline is controlled by α, where higher values result in slower decline.

Since the process is completely parameterized by α, this is often concisely denoted as sampling from the *Griffiths-Engen-McCloskey (GEM)* distribution (McCloskey, 1965; Engen, 1975; Ewens, 1990): $\rho \sim GEM(\alpha)$. Figure 2.6 shows the first two components, ρ_1 and ρ_2, in a scatter plot, for different values of α.

In Figure 2.6, at a relatively low $\alpha = 0.3$, we observe that most points are close to the line $\rho_1 + \rho_2 = 1$, since much of the probability mass is concentrated in the first few

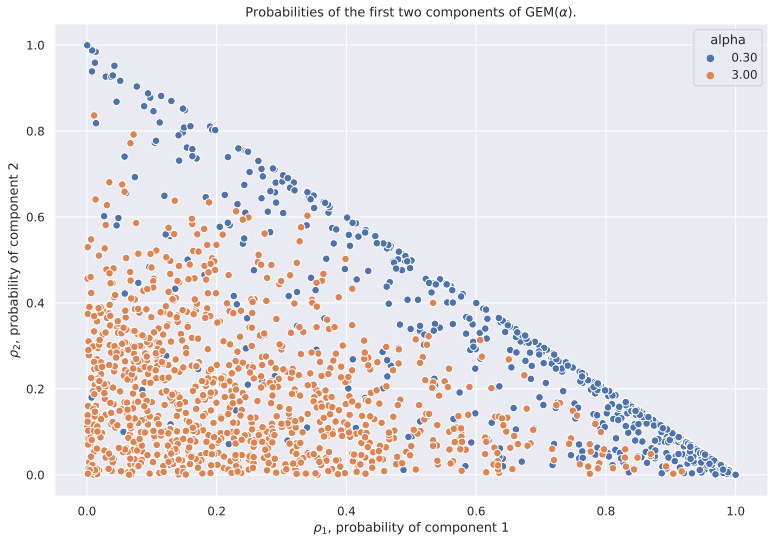


Figure 2.6: The probability masses, ρ_1 and ρ_2, of *only the first two* components of multiple samples ρ from the $GEM(\alpha)$ distribution are shown, for different values of α. At a low $\alpha = 0.3$, most of the mass is concentrated in the first few components, hence $\rho_1 + \rho_2 \approx 1$. At a higher $\alpha = 3$, the initial components receive lower masses, hence the points are located close to the origin. We observe a similar pattern in Figure 2.5.

components - this is what Figure 2.5 also shows. At a higher $\alpha = 3$, while ρ_1 and ρ_2 may take a range of values, we see them concentrated near the origin with low values - this is also something we might expect after seeing Figure 2.5. Points cannot be located in the region $\rho_1 + \rho_2 > 1$ since these are probability masses, and therefore $\sum_{k=1}^{\infty} \rho_k = 1$.

Now that we have a way to constructively determine the probability mass of an infinite number of components, we look at describing their properties.

2.2.3 The Dirichlet Process

To completely specify the data distribution, we also need to describe the distribution of the individual components. To take a familiar example, we might assume the components are Gaussian distributions[11], in which case we need to determine (μ_k, Σ_k) for each component k.

This might be done in the following manner:

1. We propose a prior G_0, referred to as the *base measure*, from which we may draw distribution parameters $\theta \sim G_0$ for the components.

[11] This is known as the *Infinite Gaussian Mixture Model* (Rasmussen, 1999).

2. For each component $k \in GEM(\alpha)$, obtained by the stick-breaking process, we assign it the randomly drawn parameter $\theta_k \sim G_0$.

This effectively produces the following distribution:

$$G := \sum_{k=1}^{\infty} \rho_k \delta_{\theta_k} \tag{2.19}$$

G represents a distribution with components that have distribution parameters θ_k and mixing coefficients ρ_k. The *Dirac delta function* δ_{θ_k} may be interpreted as a shorthand to signify that only components with distribution parameters θ_k, among all possible $\theta \sim G_0$, are valid components, and mixing coefficients ρ_k are respectively associated with these. This is the *Dirichlet Process (DP)* (Ferguson, 1973), and is expressed as $DP(\alpha, G_0)$ to indicate the dependence on α and G_0.

We illustrate this process in Figure 2.7 with an univariate example: $DP(3, \mathcal{N}(0, 1))$. We assume our components are Gaussians, with fixed $\Sigma_k = [0.5]$ and $\mu_k \sim \mathcal{N}(0, 1)$. This makes $G_0 := \mathcal{N}(0, 1)$ the base measure[12], and it acts as the prior for the only component distribution parameter μ_k.

With $\alpha = 3$, a $\rho \sim GEM(\alpha)$ is obtained using the stick-breaking process, and the probability masses of the first 20 components are shown in Figure 2.7(a). We then sample $\mu_k \sim G_0, k \in \{1, 2, ..., 20\}$, and visualize ρ_k placed at μ_k in Figure 2.7(b). This forms our G. The colors of the bars are consistent across plots (a) and (b) for easy comparison. We then generate points $x_i, i \in \{1, 2, ..., 500\}$ by following this procedure for each point:

1. For a given i, obtain $z_i \sim Cat(\rho)$. For illustration, we only show cases where $z_i \in \{1, 2, ..., 20\}$.

2. Generate $x_i \sim \mathcal{N}(\mu_{z_i}, [0.5])$.

The above steps are equivalent to $\mu_i \sim G$.

[12]Strictly speaking, (Σ_k, μ_k) should be *jointly* sampled from a base measure like the *Normal-inverse-Wishart* distribution. Here, we fix Σ_k and only consider μ_k to be drawn from the base measure in the interest of simplicity.

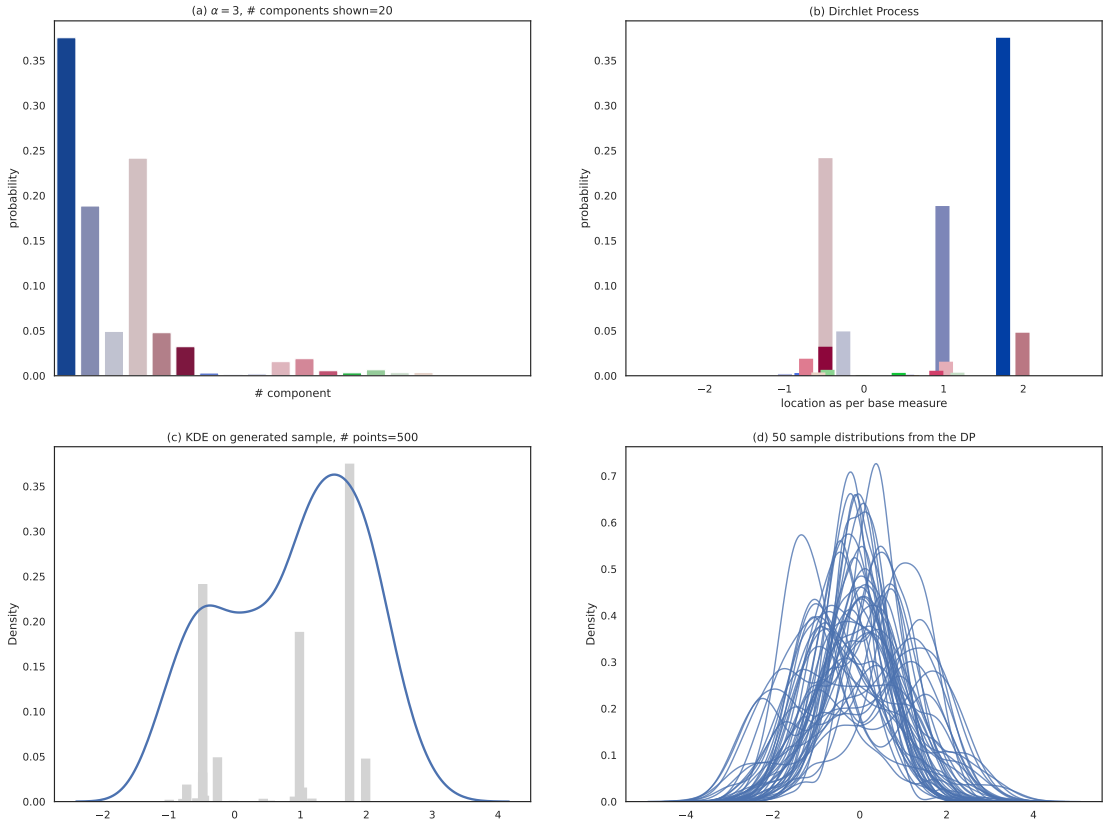


Figure 2.7: Sampling from $DP(3, \mathcal{N}(0,1))$: (a) The probability masses of the first few components are shown, obtained using the stick-breaking process. (b) These probabilities are assigned to positions on the real line using the base measure $G_0 = \mathcal{N}(0,1)$; these now form a distribution $G \sim DP(\alpha, G_0)$. (c) For $1 \leq i \leq 500$, we generate $\mu_i \sim G$, and then $x_i \sim \mathcal{N}(\mu_i, 0.5)$. A KDE is fit on this data and shown; this represents the distribution G. (d) Multiple $G \sim DP(\alpha, H)$ may be formed in this manner and visualized; some are shown here.

We fit a KDE over the instances thus generated - shown in Figure 2.7(c). This is the distribution G models. Of course, we may repeat this whole process, starting at a different $\rho \sim GEM(\alpha)$, leading us to a different G. A collection of KDEs fit on such multiple Gs are shown in Figure 2.7(d). Since the $DP(\alpha, G_0)$ can generate multiple distributions G of the form shown in Equation 2.19, for given values of α and G_0, it is considered to be a prior over *distributions*.

Let $F(\theta)$ describe our component distributions; in the example above, F is exemplified by the normal distribution $\mathcal{N}$, and it's parameter θ is the tuple (σ, μ). The steps in sampling N points using $DP(\alpha, G_0)$ may then be summarized as:

1. $\forall i \in \{1, 2, ..., N\}$, sample $z_i \sim Cat(\rho)$, where $\rho \sim GEM(\alpha)$. We only sample as many components $\rho_1, \rho_2, ..., \rho_K$ as is required to generate a random N instances. Note that this effectively uses a single instance of $\rho \sim GEM(\alpha)$, since

$\rho_1, ..., \rho_{i-1}, \rho_i$ are fixed when sampling ρ_{i+1}

2. $\forall k \in \{1, 2, ..., K\}$, sample $\theta_k \sim G_0$. Note the lack of dependence between ρ_k and θ_k.

3. $\forall i \in \{1, 2, ..., N\}$, sample $x_i \sim F(\theta_{z_i})$.

One way to accomplish Step 1 above is via *inverse transform sampling*: (a) $\forall i \in \{1, 2, ..., N\}$, generate $u_i \sim \mathcal{U}(0, 1)$ (b) sample $\rho_1, \rho_2, ...$ until ρ_K such that $\sum_{k=1}^{K} \rho_k \geq \max_{i=1}^{N}\{u_i\}$ (c) assign $z_i = k$ where k is the smallest value such that $u_i \leq \sum_{j=1}^{k} \rho_j$. This is a reasonable strategy (albeit approximate) since most of mass in a DP is concentrated in the initial components, but is computationally expensive when $u_i \approx 1$.

In practice, strategies that provably sample from a DP are used, e.g., the *Blackwell-MacQueen urn* process (Blackwell and MacQueen, 1973), the *Chinese Restaurant Process (CRP)* (Aldous, 1985). The key idea is that z_i might be directly assigned a component based on previous assignments $z_1, z_2, ...z_{i-1}$. For example, in CRP, the probability that z_i is assigned a new component k^* (equivalently, x_i is generated by a previously unseen component), as opposed to one of the K existing components is:

$$p(z_{i+1} = z | z_{1:i}, \alpha) = \frac{1}{\alpha + N}\Big(\alpha \mathbb{I}(z = k^*) + \sum_{k=1}^{K} N_k \mathbb{I}(z = k)\Big) \tag{2.20}$$

Here, N_k denotes the number of instances generated by k, i.e., $N_k = \sum_{j=1}^{i} \mathbb{I}(z_j = k)$. Equation 2.20 leads to the following constructive sampling process:

1. Let z_1 be assigned to component 1.

2. For $z_{i>1}$, assign a new component with probability $\frac{\alpha}{\alpha+N}$.

3. With probability $1 - \frac{\alpha}{\alpha+N}$, assign it to one of K existing components, with the probability of being assigned to component k being $\propto N_k$.

□

For exposition beyond this brief overview, we recommend the tutorials Teh (2010); Broderick (2015) and the texts Murphy (2012); Gelman *et al.* (2021). For models other than the DP, see Lijoi and Pruenster (2009).

2.3 Relevance

As mentioned in Chapter 1, the goal of our research is to construct compact models. The mechanism we use to do so is to learn a distribution over the training data that leads to high accuracies for a given model size.

Algorithm 2 shows the process at a high-level. Input dataset D is split into training, validation and test sets, denoted by $D_{train}, D_{val}, D_{test}$ respectively. The distribution of interest, denoted by $P_{D_{train}}(\theta)$, is learned on the training data in T iterations. The distribution parameter θ is learned to maximize model accuracy acc_{val} on the validation split D_{val}. The algorithm is general enough to admit different scoring metrics, e.g., F1, AUC, Precision, Recall. However, in our experiments we exclusively use the *F1 (macro)* score.

Algorithm 2: An abstracted view.

Data: Dataset D (N instances, d dimensions), training algorithm f, model size η, scoring function $score$, optimization budget T.

Result: θ^*, new model accuracy

$D_{train}, D_{val}, D_{test} \leftarrow create\_splits(D)$
$\mathcal{H} \leftarrow \{\}, \theta^* \leftarrow \varnothing, acc^* \leftarrow 0, M^* \leftarrow 0$
for $t \leftarrow 1$ **to** T **do**
  $\theta_t \leftarrow suggest(\mathcal{H})$
  $D_s = \{(x_j, y_j)\}_{j=1}^{N_s}$, where $x_j \sim P_{D_{train}}(\theta_t)$
  $M_t \leftarrow f(D_s, \eta)$
  $acc_{train} \leftarrow score(M_t, D_{train})$
  $acc_{val} \leftarrow score(M_t, D_{val})$
  if $acc_{val} > acc^*$ **then**
    $\theta^* \leftarrow \theta_t, acc^* \leftarrow acc_{val}, M^* \leftarrow M_t$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\theta_t, acc_{train})\}$
end
return θ^*, $score(M^*, D_{test})$

In our work, the iterative learning, via the function $suggest()$ in Algorithm 2, is realized by a BO algorithm, while the distribution $P_{D_{train}}(\theta)$ is a DP mixture model. The history $\mathcal{H}$ of past parameters and corresponding held-out model evaluations is used

to learn from; this step is denoted by $suggest(\mathcal{H})$. Much of our research is concerned with refining the algorithm to make it practically usable, investigating the conditions under which it produces useful results and validating the results themselves.

CHAPTER 3

Compact Models using Density Trees

In this chapter, we present our first technique to minimize the trade-off between size and accuracy. Our approach is *model agnostic*, i.e., it may be applied to arbitrary model families. The technique learns a distribution over training data such that an interpretable model (of a given size) trained on a sample drawn from it often achieves test accuracy that is significantly higher than when the training data is used as-is. This is especially true when the model size is small.

Figure 3.1 (reproduced from Chapter 1) shows the impact of our technique. A two-label balanced classification dataset is shown in (a). Its distribution density is shown using a kernel density plot in (b), and the generalization learned by a DT with $depth = 5$, trained using a sample of this data, is shown in (d). The corresponding plots, when our technique is used, are respectively shown in (c) and (e). The improved generalization is easily seen by comparing the class-regions learned in (d) and (e).

The key contributions from the analysis in this chapter are:

1. We propose an algorithm to find a sampling distribution over a training dataset that is optimal[1] in terms of achieving high test accuracy, for a provided model family and model size.

2. As a corollary, we make this possibly counter-intuitive, observation: *in general, the optimal training distribution is not the same as the test distribution, especially at small model sizes*. This is a "small model" effect: we are able to clearly show that as model size increases the optimal training distribution progressively approximates the test distribution.

3. The technique is *model agnostic*[2] and may be applied to arbitrary learners. Its effectiveness using different learning algorithms and datasets is demonstrated. It

[1] Equation 3.1 precisely states the nature of this optimality.

[2] We adopt the common usage of the term (Ribeiro *et al.*, 2016; Lundberg and Lee, 2017; Chen *et al.*, 2018) to imply our technique is agnostic to model *families*.

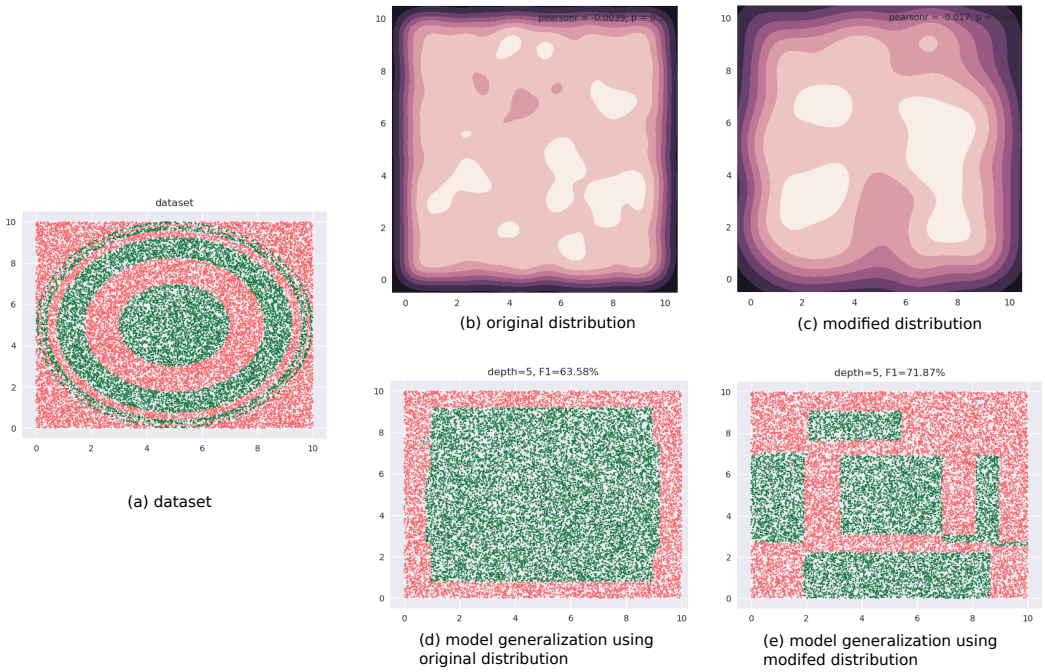


Figure 3.1: Changing the input distribution can significantly affect model accuracy. (a) shows a toy dataset. (b) shows the original distribution of the data as a *Kernel Density Estimation* plot and (d) visualizes the generalization learned by a Decision Tree of $depth = 5$ when using this distribution for training. (c) shows visualizes the sampling distribution learned by our technique and (e) shows the corresponding generalization learned by a Decision Tree with $depth = 5$. The latter achieves a higher accuracy of $F1 = 0.72$, compared to $F1 = 0.64$ of the former.

also admits a flexible notion of model size, e.g., depth of a decision tree, number of terms with non-zero coefficients in a linear model, number of trees in a *Gradient Boosted Model (GBM)* model.

4. We show that learning the optimal training distribution 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 note here that our technique may be used with any model family, and not just ones considered interpretable. But the fact that we typically see increased accuracy in the small model size regime 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 5.3.

The rest of the chapter is organized as follows: Section 3.1 provides on overview

of the technique. In Section 3.2 we describe in detail two formulations of the problem of learning the optimal density. Section 3.3 reports experiments we have conducted to evaluate our technique. It also presents our analysis of the results. In Section 3.4 we discuss some of the algorithm design choices, concluding with a summary in Section 3.5. A discussion of future work is deferred to Section 5.3.

3.1 Overview

This section provides an overview of various aspects of our work: we provide some intuition for why we expect the ideal train distribution to differ from test, mathematically formalize our contributions, describe where our technique fits into a model building workflow and establish our notation and terminology.

For a discussion on previous work, see Section 1.6, where we differentiate our technique with *Knowledge Distillation*, *Active Learning*, *Transfer Learning* and *Coreset* identification.

3.1.1 Intuition

Let's begin by considering why we might expect a learned distribution over the training data to result in greater accuracy.

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

- Decision Trees - Constructing optimal binary decision trees is an NP-complete problem (Hyafil and Rivest, 1976). Hence, heuristics like one step lookahead are used (as an artifact of this, note in our example that the CART tree has a significantly smaller number of leaves than the possible $2^5 = 32$).
- 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, hid-

den 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. This is what we observe in Figure 3.1, and it is extensively validated later in this chapter.

We make a note of the fact that in recent years multiple algorithms have been proposed for creating optimal decision trees, e.g., *GOSDT* (Lin *et al.*, 2020), *InferDT* (Avellaneda, 2020).One might argue that this line of research renders our work moot. However, in terms of number of instances, features, feature dichotomies and tree size, their computational complexity makes them suitable for specific datasets. As an example, consider the GOSDT algorithm applied to the 2D dataset from Figure 3.1. Here, both features are continuous and assume real number values that are uniformly distributed within the range $[0, 10]$. For 500 instances and a setting of 5 for the maximum depth, GOSDT unsuccessfully exits its process[3]. Problems with the scaling of GOSDT have also been noted in Agarwal *et al.* (2022) (see Section S4.2). In comparison, our experiments use 6000 training instances to construct trees with maximum depths up to 15. This is not meant to be a criticism of such techniques - on the contrary, we believe this is an exciting direction of research - but we do want to point out they may not be applicable to a wide variety of classification tasks yet.

Of course, a crucial difference is our techniques are model agnostic, and in our experiments we show it applied to DTs, linear models and GBMs.

3.1.2 Formal Statement

It is helpful to mathematically formalize the outcome we have described.

Let,

1. $accuracy(M, p)$ 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

[3]We used the official library available at `https://github.com/Jimmy-Lin/GeneralizedOptimalSparseDecisionTrees`. The process exits after having used $\sim$ 4 GB of swap memory on a machine with 32 GB of RAM, i7-7700HQ CPU and Ubuntu as its OS.

"accuracy" as a generic placeholder for a measure of model correctness. This may specifically measure *F1-score*, *AUC*, *lift*, etc., as needed.

2. $train_{\mathcal{F}}(p, \eta)$ 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 η, 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 η for data with distribution $p(X, Y)$, our technique produces the *optimal training distribution* $p^*_\eta(X, Y)$ such that:

$$p^*_\eta = \arg\max_q accuracy(train_{\mathcal{F}}(q, \eta), p) \tag{3.1}$$

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:

$$accuracy(train_{\mathcal{F}}(p, \eta), p) \lessapprox accuracy(train_{\mathcal{F}}(p^*_\eta, \eta), p) \tag{3.2}$$

The use of the "$\lessapprox$" symbol emphasizes these relationships are validated empirically using samples from the corresponding distributions p and q.

Furthermore, the relationship in Equation 3.2 may be separated into two regimes of operation. A model trained on p^*_η outperforms one trained on the original distribution p up to a model size η', with both models being comparably accurate beyond this point:

$$\text{For } \eta \leq \eta', accuracy(train_{\mathcal{F}}(p, \eta), p) < accuracy(train_{\mathcal{F}}(p^*_\eta, \eta), p) \tag{3.3}$$

$$\text{For } \eta > \eta', accuracy(train_{\mathcal{F}}(p, \eta), p) = accuracy(train_{\mathcal{F}}(p^*_\eta, \eta), p) \tag{3.4}$$

3.1.3 Workflow

Figure 4.5 shows how our sampling technique modifies the model building workflow. In the standard workflow, we feed the data into a learning algorithm, $train_{\mathcal{F}}()$, 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.

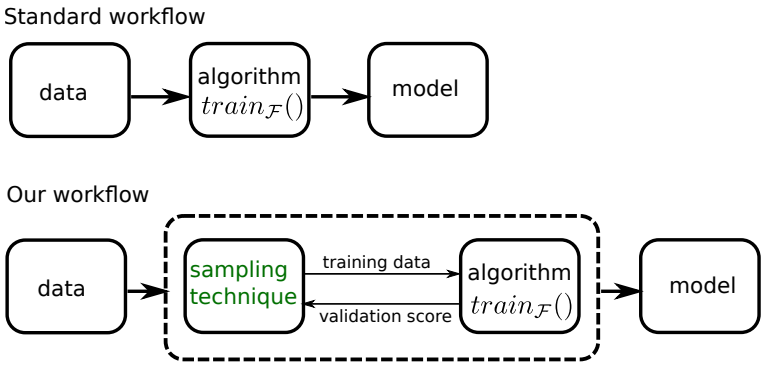


Figure 3.2: Our workflow compared with the standard workflow. Arrows denote flow of information. The key difference is the presence of a sampler, that interacts with the training algorithm over multiple iterations to produce a model.

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.

3.1.4 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 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 definition of size they might disagree on the *value* for the size up to which the model stays interpretable. For e.g., 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 chapter 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 *train*, *validation* and *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 θ, defined over the variable x, is denoted by $p(x; \theta)$.

5. We use the following terms introduced before:

   - $accuracy(M, p)$ 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 $accuracy(M, (X, Y))$ where (X, Y) is the dataset.

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

6. We denote the depth of a tree T by the function $depth(T)$.

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

3.2 Methodology

In this section we describe our sampling technique. We begin with a intuitive formulation of the problem in Section 3.2.1 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 3.2.5.

3.2.1 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 Ψ is a parameter vector. Our optimization algorithm runs for a budget of T time steps. Algorithm 3 lists the execution steps.

Algorithm 3: Naive formulation

Data: Learning algorithm $train_{\mathcal{F}}()$, size of model η, data (X, Y), iterations T
Result: Optimal density Ψ^*, accuracy on test set s_{test}

1 Create stratified subsets $(X_{train}, Y_{train}), (X_{val}, Y_{val}), (X_{test}, Y_{test})$ from (X, Y);
2 **for** $t \leftarrow 1$ **to** T **do**
3 $\quad \Psi_t \leftarrow suggest(s_{t-1}, ...s_1, \Psi_{t-1}, ..., \Psi_1)$ `// suggest() is described below`
4 $\quad (X_t, Y_t) \leftarrow$ sample N_s points from (X_{train}, Y_{train}) based on $p(X_{train}; \Psi_t)$;
5 $\quad M_t \leftarrow train_{\mathcal{F}}((X_t, Y_t), \eta)$;
6 $\quad s_t \leftarrow accuracy(M_t, (X_{val}, Y_{val}))$;
7 **end**
8 $t^* \leftarrow \arg\max_t \{s_1, s_2, ..., s_{T-1}, s_T\}$;
9 $\Psi^* \leftarrow \Psi_{t^*}, M^* \leftarrow M_{t^*}$;
10 $s_{test} \leftarrow accuracy(M^*, (X_{test}, Y_{test}))$;
11 **return** Ψ^*, s_{test}

In Algorithm 3:

1. $suggest()$ is a call to the optimizer at time t, that accepts past validation scores $s_{t-1}, ...s_1$ and values of the density parameter $\Psi_{t-1}, ..., \Psi_1$. 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 $x_i \in X_{train}$, and their corresponding labels $y_i \in Y_{train}$. We may alternatively think of the *sample weight* $w(x_i)$ of an instance x_i where $w(x_i) \propto p(x_i; \Psi_t), \forall x_i \in X_{train}$.

3. Although the training happens on a sample drawn based on Ψ_t, the validation dataset (X_{val}, Y_{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.

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_s to our improved formulation in Section 3.2.5.

Algorithm 3 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 Ψ_t. It is easy to see here what makes our technique model-agnostic: the arbitrary learner $train_{\mathcal{F}}()$ 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 precise representation of the pdf $p(x; \Psi)$.
2. The optimizer to use for $suggest()$.

We look at these next.

3.2.2 Density Representation

The characteristics the pdf, $p(x; \Psi)$, 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 3; 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* α, which determines both the number of components that have at least one instance associated with them[5], and the association of a specific data point to a specific component. The parameters describing the Gaussian for a component are not directly learned, but are instead themselves drawn from prior distributions; the parameters of these prior distributions comprises our fixed set of variables (Requirement 2). We make the parameter α part of our optimization search space, so that the appropriate number of components maybe discovered; this makes our pdf flexible (Requirement 1). The DP is discussed in detail in Section 2.2.

We make a few modifications to the IGMM for it to better fit our problem. This doesn't change its compatibility to our requirements. Our modifications are:

1. Since our data is limited to a "bounding box" within $\mathbb{R}^d$ (this region is easily found by determining the *min* and *max* values across instances in the provided dataset, for each dimension, ignoring outliers if needed), we replace the Gaussian mixture components with a multivariate generalization of the $Beta$ distribution. We pick $Beta$ since it naturally supports bounded intervals. In fact, we may treat the data as lying within the unit hypercube $[0, 1]^d$ without loss of generality, and with the understanding that the features of an instance are suitably scaled in the actual implementation.

   Using a bounded interval distribution provides the additional benefit that we don't need to worry about infeasible solution regions in our optimization.

2. Further, we assume *independence* across the d dimensions as a starting point.

[4]The optimizer we use, TPE, *can* handle conditional spaces. However, as mentioned, our goal is flexibility in implementation.

[5]In theory, the DP has an infinite number of components, of which only a finite number are associated with the observed data. The latter is often is referred to as *clusters*. Here we universally use the term "component" and rely on the context to make the connotation clear.

We do this to minimize the number of parameters, similar to using a diagonal covariance matrix in GMMs.

Thus, our d-dimensional generalization of the $Beta$ is essentially a set of d $Beta$ distributions, and every component in the mixture is associated with such a set. For k mixture components, we have $k \times d$ $Beta$ distributions in all, as against k d-dimensional Gaussians in an IGMM.

3. A $Beta$ distribution uses two positive valued *shape parameters*. Recall that we don't want to learn these parameters for each of the $k \times d$ $Beta$ distributions (which would defeat our objective of a fixed parameter space); instead we sample these 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 $A_j \sim Beta(a_j, b_j)$ and $B_j \sim Beta(a'_j, b'_j)$, 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)*[6]. For d dimensional data, we have $\Psi = \{\alpha, a_1, b_1, a'_1, b'_1, ..., a_d, b_d, a'_d, b'_d\}$. This is a total of $4d + 1$ parameters.

Algorithm 4 shows how we sample N_t points from (X, Y) using the IBMM.

We first determine the partitioning of the number N_s, 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_{ij}, B_{ij} - from the priors (lines 4-6), followed by constructing sampling weights $p(x_l|c_i), \forall x_l \in X$

[6]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)

Algorithm 4: Sampling using IBMM

Data: number of points to sample N_s, dataset $(X, Y), X \in \mathbb{R}^{N \times d}, Y \in \mathbb{R}^N$
Result: (X_t, Y_t), $X_t \in \mathbb{R}^{N_s \times d}, Y_t \in \mathbb{R}^{N_s}$

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

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.

3.2.3 Choice of Optimizer

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 $suggest()$ critically influences the utility of Algorithm 3.

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 $train_{\mathcal{F}}()$. 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 in which the instances are presented, etc.

3. **Requirement 3: minimizes calls to the objective function.** The acquisition cost of a fitness value s_t for a solution Ψ_t is high: this requires a call to $accuracy()$, which in turn calls $train_{\mathcal{F}}()$. 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 Optimistic Optimization* (Grill *et al.*, 2015) and *Hyperband* (Li *et al.*, 2017*b*).

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); Perrone *et al.* (2018); Dai *et al.* (2019), where the challenges are similar to ours. BO is discussed in detail in Section 2.1.

We briefly consider why BO techniques meet our requirements: they 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 $suggest()$ 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 (Hutter *et al.*, 2011; Bergstra *et al.*, 2011; Snoek *et al.*, 2012; Wang *et al.*, 2013; Gelbart *et al.*, 2014; Snoek *et al.*, 2015; Hernández-Lobato *et al.*, 2016; Rana *et al.*, 2017; Levesque *et al.*, 2017; Li *et al.*, 2017*a*; Letham *et al.*, 2017; Malkomes and Garnett, 2018; Perrone *et al.*, 2018; Nayebi *et al.*, 2019; Alvi *et al.*, 2019; Dai *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). TPE is decribed in detail in Section 2.1.2.

We note here that *TPE* supports conditional parameter spaces, which would have allowed us to use a finite mixture model such as GMMs, setting the number of mixture components as the top level optimization variable. However, our design choice of a fixed parameter space for $p(x; \Psi)$ effectively makes our technique a **framework**: any optimizer that satisfies the above criteria may be used. For example, any of the BO algorithms from the *black-box optimization challenge*, NeurIPS2020 (Turner *et al.*, 2021), may be used to implement $suggest()$ in Algorithm 9.

3.2.4 Challenges

The primary challenge with this formulation is the size of the search space. We have successfully tried out Algorithm 3 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 such as 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.

3.2.5 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 Ψ. 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 Ψ_t 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 Ψ_t 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 3.3 shows what this looks like when we learn a tree using CART on the dataset from Figure 3.1(a). Leaf regions are shown with the rectangles with the black edges.

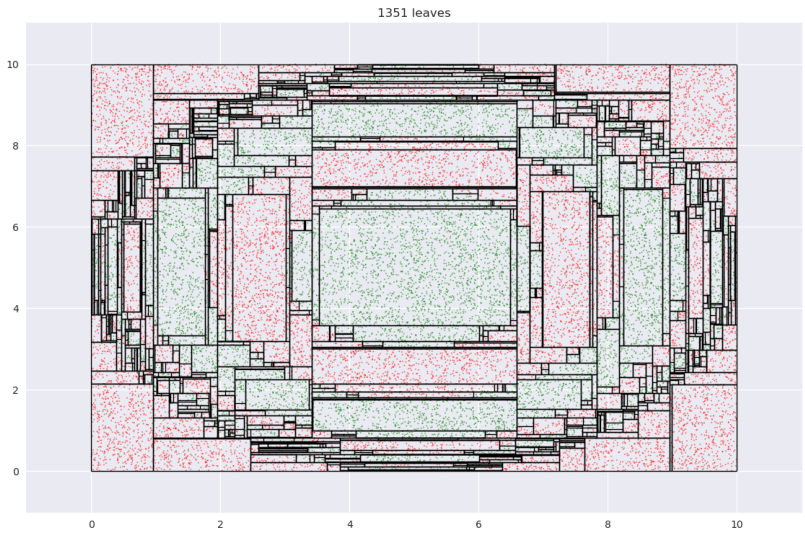


Figure 3.3: Tessellation of space produced by leaves of a decision tree.

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 3.4 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.

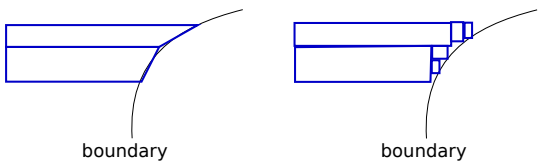


Figure 3.4: We see leaves of small areas because DTs are forced to approximate curvature with them.

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[7]. Instead of directly determining an optimal *pdf* on the input space, we now do the following:

1. Learn a DT, with no size restrictions, on the data (X_{train}, Y_{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 $L \in \{1, 2, ..., m\}$ be a random variable denoting a leaf. Our *pmf* is $P_L(i) = P(L = i) = f(R_i)$, where $f(R_i) \propto vol(R_i)^{-1}$.

   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, $x \sim \mathcal{U}(R_i)$.

   (c) Since leaves are characterized by low entropy of the label distribution, we assign the majority label of leaf i, denoted by $label(i)$, to the sampled point x.

   Assuming we have k unique labels, $label(i)$ is calculated as follows:

   Let $S_i = \{y_j : y_j \in Y_{train}, x_j \in X_{train}, x_j \in R_i\}$. Then,

$$label(i) = \arg\max_k \hat{p}_{ik} \tag{3.5}$$

$$\text{where, } \hat{p}_{ik} = \frac{1}{|S_i|} \sum_{S_i} I(y_j = k) \tag{3.6}$$

   Note here that because of using $\mathcal{U}(R_i)$ we may generate points $x \notin X_{train}$. Also, since a point $x \in R_i \cap X_{train}$ gets *assigned* $label(i)$, the conditional distribution

[7]While DTs have been often used to group similar instances into neighborhoods - such as for density estimation (Ram and Gray, 2011), anomaly detection (Liu *et al.*, 2008), local model construction (Bloniarz *et al.*, 2016), construction of local interpretable models (Plumb *et al.*, 2018), our work is the first we are aware of that utilizes DTs to identify regions adjacent to class boundaries.

of labels approximately equals the original distribution:

$$p(Y_t|X_t) \approx p(Y_{train}|X_{train}) \tag{3.7}$$

We call such a DT a *density tree*[8] which we formally define as follows.

Definition 3.2.1. We refer to a DT as a **density tree** if (a) it is learned on (X_{train}, Y_{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 $f(R_i) \propto vol(R_i)^{-1}$.

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 3.3 (reproduced below), q no longer ranges over all possible distributions; but over a restricted set relevant to the problem:

$$p^*_\eta = \arg\max_q accuracy(train_{\mathcal{F}}(q, \eta), p) \tag{3.8}$$

We visit the issue of efficiency towards 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 $f(R_i) = c \cdot vol(R_i)^{-1}$ where c is the normalization constant $c = 1/\sum_{i=1}^{m} vol(R_i)^{-1}$. However, this quantity changes rapidly with volume.

[8] 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 the thesis, and the term "density tree" is used as-is.

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, $f(R_i) = c \cdot diag(R_i)^{-1}$ where $c = 1/\sum_{i=1}^{m} diag(R_i)^{-1}$. Since DT leaves are axis-parallel hyperrectangles, $diag(R_i)$ 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 λ as the smoothing coefficient. This modifies our *pmf* thus:

$$f'(R_i) = c\Big(f(R_i) + \frac{\lambda}{m}\Big) \tag{3.9}$$

Here, c is the normalization constant. The optimizer discovers the ideal value for λ.

We pick Laplace smoothing because it is fast. Our framework, however, is general enough to admit a wide variety of options (discussed in Section 5.3).

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 3.5. 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 transforma-

tion may be noticed at the top left and bottom right regions).

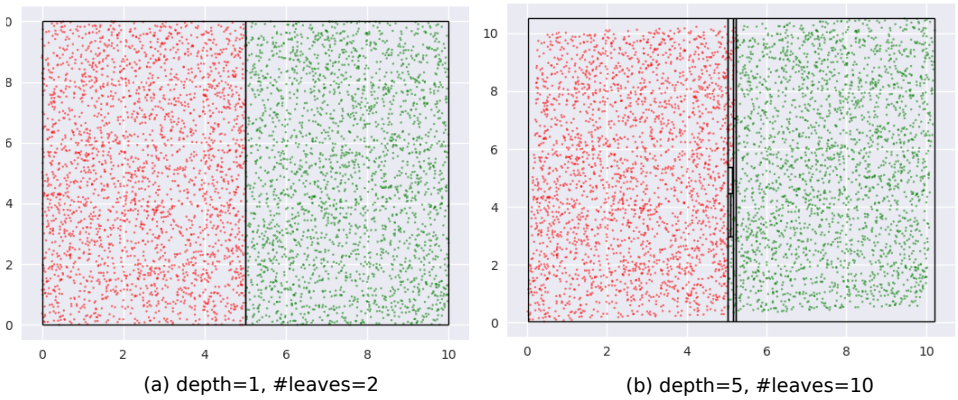


Figure 3.5: (a) Axis parallel boundaries don't create small regions. (b) This can be addressed by transforming the data. We see an increase in depth and the number of leaves of the density tree in the latter case.

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 3.6 illustrates this idea, using the same data as in Figure 3.5.

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

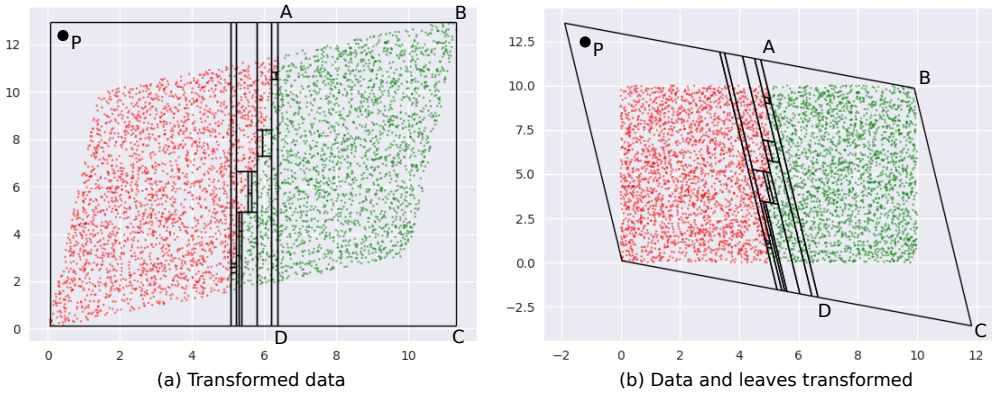


Figure 3.6: (a) Point P lies within a leaf in the transformed data. (b) In the inverse transformation, it is seen that the leaves contain regions outside the bounding box of the original dataset, and P is an outlier.

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:

- We lose the convenience and low runtime of uniform sampling $\mathcal{U}(R_i)$: the leaves are not simple hyperrectangles any more.
- For leaves that are not confined by the data bounding box in the original space, e.g., $\overline{ABCD}$, we cannot sample from the entire leaf region without risking obtaining outliers again, i.e., we might end up obtaining out-of-distribution points such as P anyway.

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 $A \in \mathbb{R}^{d \times d}$:

$$[A]_{pq} = 1, \text{ if } p = q \tag{3.10}$$

$$[A]_{pq} \sim \mathcal{U}([0, \epsilon]), \text{ if } p \neq q, \text{ where } \epsilon \in \mathbb{R}_{>0} \text{ is a small number.} \tag{3.11}$$

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 3.5 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 5 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.

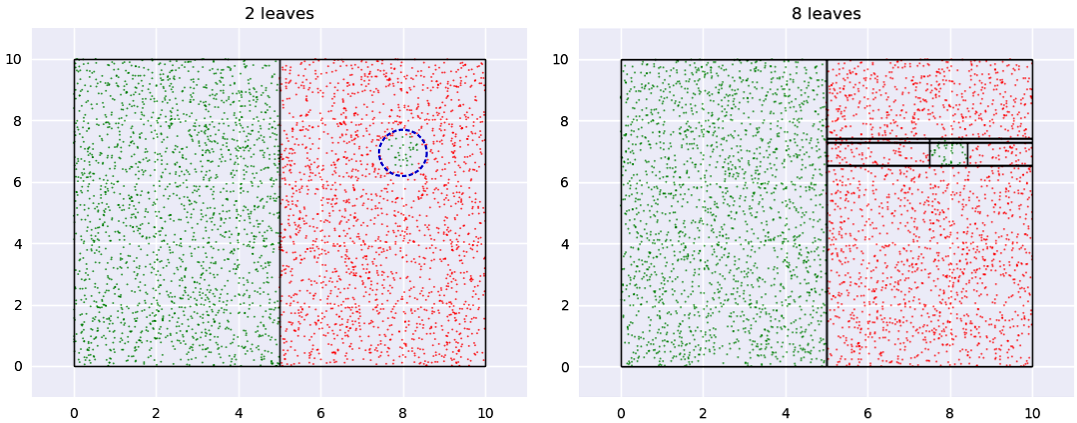
4. **Selective Generalization**. Since we rely on geometric properties alone to define our *pmf*, all boundary regions receive a high probability mass irrespective of their contribution to classification accuracy. This is not desirable when the classifier is small and must focus on a few high impact regions. In other words, we prioritize

Algorithm 5: Create bag of density trees, B

Data: (X_{train}, Y_{train}), size of bag n
Result: $B = \{(T_1, A_1), (T_2, A_2), ..., (T_n, A_n)\}$

$B = \{\}$;
for $i \leftarrow 1$ **to** n **do**
  Create matrix $A_i \in \mathbb{R}^{d \times d}$ s.t. $[A_i]_{pq} = 1$, if $p = q$ else $[A_i]_{pq} \sim \mathcal{U}([0, \epsilon])$;
  $X'_{train} \leftarrow A_i X_{train}$;
  $T_i \leftarrow$ learn tree on (X'_{train}, Y_{train});
  $B \leftarrow B \cup \{(T_i, A_i)\}$
end
return B

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*.



(a) Partitions using a shallow tree with 2 leaves.

(b) Partitions using a deeper tree with 8 leaves.

Figure 3.7: A region of low impact is shown in the first panel with a dashed blue circle. The first tree in (a) ignores this while a second, larger, tree in (b) creates a leaf for it.

Figure 3.7 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.

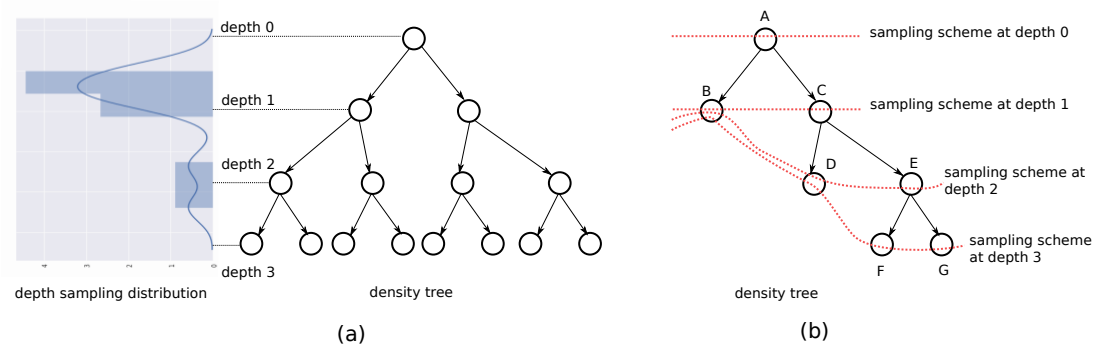


Figure 3.8: (a) The set of nodes at a depth have an associated *pmf* to sample from (not shown). A depth is picked based on the IBMM. (b) In case of an incomplete binary tree, we use the last available nodes closest to the depth being sampled from, so that the entire input space is represented. The red dotted lines show the nodes comprising the sampling scheme for different depths.

Figure 3.8(a) illustrates this idea. The depth distribution is visualized vertically and adjacent to a tree. We sample $r \in [0, 1]$ from the distribution, and scale and discretize it to reflect a valid value for the depth. Let $depth_T()$ 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 $depth_T(r) = 0$ i.e. at the *root*, and $r = 0.5$ implies we must sample from the nodes at $depth_T(r) = 1$. 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 3.2.2, the depth-distribution has a parameter α 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 6 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** $\boldsymbol{E}$. If the label distribution at a node has $entropy \leq E$, 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 $entropy > E$, we sample only among the *training* data instances that the node covers. Like ϵ, 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 3.8(b) shows this: at $depth = 0$ and $depth = 1$, 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 $depth = 2$ and $depth = 3$, 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 6 shows how sampling from B works.

□

Figure 3.9 illustrates some of the distributions we obtain using our mechanism. Panel (a) shows our data - note, we only have axis-aligned boundaries. In panels (b), (c), (d), we show the depth distribution at the top, going from favoring the root in (b), to nodes halfway along the height of the tree in (c), finally to the leaves in (d). The contour plot visualizes the distributions, where a lighter color indicates relatively higher sample density. We see that in (b), we sample everywhere in the data bounding box. In (c), the larger boundary is identified. In (d), the smaller boundary is also identified. A bag of size 5 was used and the smoothing coefficient λ 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. λ, the Laplace smoothing coefficient.

Algorithm 6: Sampling from a bag of density trees, B

Data: # points to sample N, bag of density trees B, depth distribution Ψ, smoothing parameter λ

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

$X = [\,], Y = [\,]$;
for $i \leftarrow 1$ **to** N **do**
  $r \sim \Psi$;
  $T, A \leftarrow$ randomly pick a tree and the corresponding transformation from B;
  $\Theta \leftarrow$ construct *pmf* over the nodes at $depth_T(r)$, smooth with λ `// back-up if depth is incomplete`
  $L \sim \Theta$ `//` L `is a node at` $depth_T(r)$
  $S_L \leftarrow \{(x_j, y_j) : x_j \in X_{train} \cap R_L$ and $y_j \in Y_{train}$ is its label$\}$;
  if $entropy(L) \leq E$ **then**
    $x \sim \mathcal{U}(L), y \leftarrow label(L)$ `//` $label()$ `defined in Equation 3`
  else
    $(x, y) \sim S_L$ `// notation: sample a point from` S_L
  end
  $X \leftarrow \begin{bmatrix} X \\ A^{-1}x \end{bmatrix}, Y \leftarrow \begin{bmatrix} Y \\ y \end{bmatrix}$;
end
return (X, Y)

2. α, 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 $A_i \sim Beta(a, b)$, $B_i \sim Beta(a', b')$.

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

We also introduced two additional parameters: ϵ 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. $N_s \in \mathbb{N}$, 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.

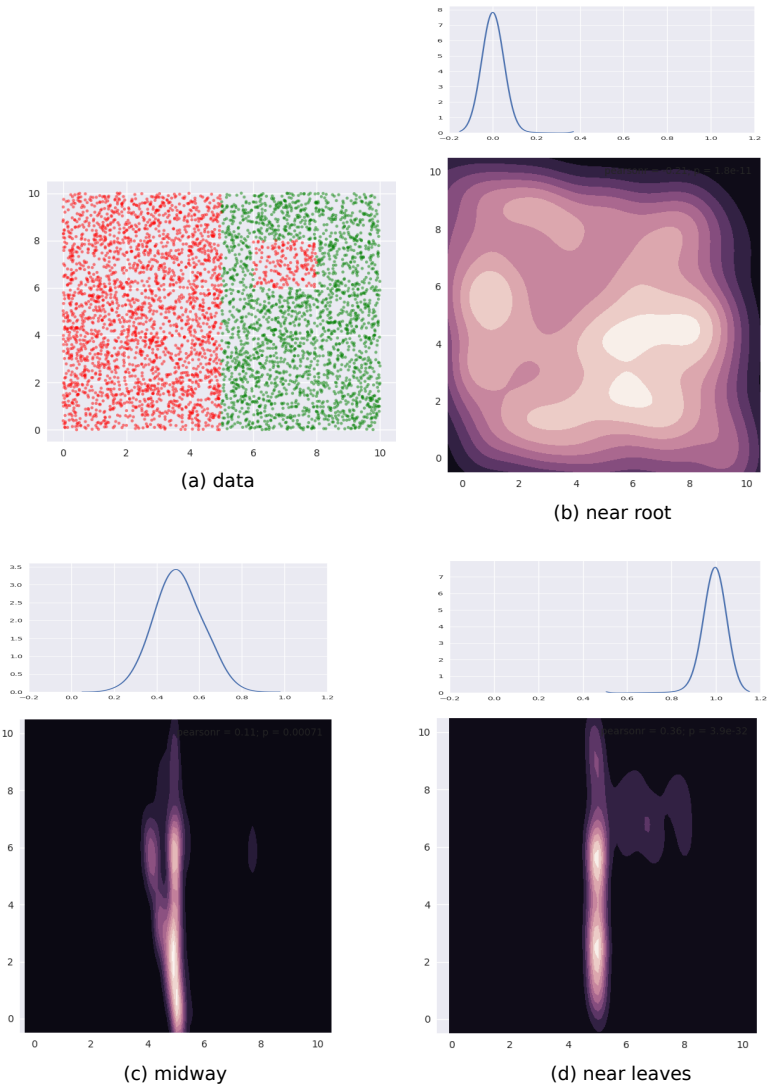


Figure 3.9: (a) shows our dataset, while (b), (c), (d) show how the sampling distribution varies with change of the depth distribution. At (b), where the depth distribution is concentrated at the root, there is no significant pattern. As we move away from the root towards the middle of the tree in (c), and then closer to leaves in (d), we observe the sampling distribution progressively discovers the multiple class boundaries.

We constrain N_s to be larger than the minimum number of points needed for statistically significant results.

Note that we can allow $N_s > |X_{train}|$. 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_s, we sample $(1 - p_o)N_s$ points from the density tree(s) and $p_o N_s$ points (stratified) from our training data (X_{train}, Y_{train}).

Recall that our hypothesis is that learning a distribution helps until a size η' (Equa-

tion 3.1.2). 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 Ψ_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 behaviour are collectively denoted by $\Psi = \{\alpha, a, b, a', b'\}$. The complete set of variables is denoted by $\Phi = \{\Psi, N_s, \lambda, p_o\}$.

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 eight 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 7 shows how we modify our naive solution to incorporate the new sampler.

As before, we discover the optimal Φ 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_{train}, Y_{train}), as described in Algorithm 5. At each iteration in the optimization, based on the current value p_{o_t}, we sample data from B and (X_{train}, Y_{train}), train our model on it, and evaluate it on (X_{val}, Y_{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 7 (also see Section A.1):

1. At $t = 1$, Φ is initialized as: $\alpha = 0.1, a = 1, b = 1, a' = 1, b' = 1, N_s = |D_{train}|, p_o = 1$. The values for α, a, b, a', b' carry no significance and are arbitrary, since setting $p_o \rightarrow 1$ forces sampling only from the original distribution. Combined with $N_s = |D_{train}|$, this setting mimics the baseline, i.e., training the

Algorithm 7: Adaptive sampling using density trees

Data: Learning algorithm $train_{\mathcal{F}}()$, size of model η, data (X, Y), number of density trees n, iterations T

Result: Φ^*, s_{test}

1 Create stratified samples $(X_{train}, Y_{train}), (X_{val}, Y_{val}), (X_{test}, Y_{test})$ from (X, Y);
2 Construct bag B of n density trees on (X_{train}, Y_{train});
3 **for** $t \leftarrow 1$ **to** T **do**
4 $\quad \Phi_t \leftarrow suggest(s_{t-1}, ...s_1, \Phi_{t-1}, ..., \Phi_1)$ // see text for initialization at $t = 1$
$\quad$ // Note: $\Phi_t = \{\Psi_t, N_{s\_t}, \lambda_t, p_{o\_t}\}$ where $\Psi_t = \{\alpha_t, a_t, b_t, a'_t, b'_t\}$.
5 $\quad N_o \leftarrow p_{o\_t} \times N_{s\_t}$;
6 $\quad N_B \leftarrow N_{s\_t} - N_o$;
7 $\quad (X_o, Y_o) \leftarrow$ sample N_o points from (X_{train}, Y_{train}) based on $p(X_{train}; \Psi_t)$;
8 $\quad (X_{dp}, Y_{dp}) \leftarrow$ sample N_B points from B, using Algorithm 6;
9 $\quad X_t \leftarrow \begin{bmatrix} X_o \\ X_{dp} \end{bmatrix}, Y_t \leftarrow \begin{bmatrix} Y_o \\ Y_{dp} \end{bmatrix}$ // combine the above samples
10 $\quad M_t \leftarrow train_{\mathcal{F}}((X_t, Y_t), \eta)$;
11 $\quad s_t \leftarrow accuracy(M_t, (X_{val}, Y_{val}))$;
12 **end**
13 $t^* \leftarrow \arg\max_t \{s_1, s_2, ..., s_{T-1}, s_T\}$;
14 $\Phi^* \leftarrow \Phi_{t^*}$;
15 $(X^*, Y^*) \leftarrow$ sample N_s^* points from (X_{train}, Y_{train}) and B based on p_o^*;
16 $M^* \leftarrow train_{\mathcal{F}}((X^*, Y^*), \eta)$;
17 $s_{test} \leftarrow accuracy(M^*, (X_{test}, Y_{test}))$;
18 **return** Φ^*, s_{test}

interpretable model without our algorithm, thus providing the optimizer with a good initial reference point in its search space[9].

Note, however, that the optimizer still needs to discover that any setting with $p_o \rightarrow 1$ and $N_s \approx |D_{train}|$, *independent* of values for α, a, b, a' and b', yields similar accuracy.

2. We use a $train : val : test$ split ratio of $60 : 20 : 20$.

3. 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_{val}, Y_{val}) and (X_{test}, Y_{test}) represent

[9]This is still not equivalent to the baseline because since models are trained in a resource constrained environment within the optimizer - as described in Section 3.3.2.

the original distribution, and thus indeed test the efficacy of our technique on data with varying degrees of class imbalance.

3.3 Experiments

3.3.1 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 3.1. The column "Label Entropy", quantifies the extent of class imbalance, and is computed for a dataset with C classes in the the following way:

$$\text{Label Entropy} = \sum_{j \in \{1,2,...,C\}} -p_j \log_C p_j \tag{3.12}$$

$$\text{Here, } p_j = \frac{|\{x_i | y_i = j\}|}{N}$$

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

3.3.2 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 $depth(T_{opt})$. We then try our algorithm for these settings of CART's max_depth parameter: $\{1, 2, ..., min(depth(T_{opt}), 15)\}$, 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

Table 3.1: Datasets: we use the dataset versions available on the LIBSVM website (Chang and Lin, 2011). However, we have mentioned the original source in the "Description" column. 10000 instances from each dataset are used. A $train : val : test$ split ratio of $60 : 20 : 20$ is used for D_{train}, D_{val} and D_{test} in Algorithm 7. The splits are stratified wrt labels.

S.No.	Dataset	Dimensions	# Classes	Label Entropy	Description
1	cod-rna	8	2	0.92	Predict presence of non-coding RNA common to a pair of RNA sequences, based on individual sequence properties and their similarity (Uzilov *et al.*, 2006).
2	ijcnn1	22	2	0.46	Time series data produced by an internal combustion engine is used to predict normal engine firings vs misfirings (Prokhorov, 2001). Transformations as in Chang and Lin (2001).
3	higgs	28	2	1.00	Predict if a particle collision produces Higgs bosons or not, based on collision properties (Baldi *et al.*, 2014).
4	covtype.binary	54	2	1.00	Modification of the *covtype* dataset (see row 12), where classes are divided into two groups (Collobert *et al.*, 2002).
5	phishing	68	2	0.99	Various website features are used to predict if the website is a *phishing* website (Mohammad *et al.*, 2012). Transformations used as in Juan *et al.* (2016)
6	a1a	123	2	0.80	Predict whether a person makes over 50K a year, based on census data variables (Dua and Graff, 2017). Transformations as in Platt (1998).
7	pendigits	16	10	1.00	Classify handwritten digit samples into the digits 0-9. (Alimoglu and Alpaydin, 1996; Dua and Graff, 2017).
8	letter	16	26	1.00	Images of the capital letters A-Z were produced by random distortion of these characters from 20 fonts. The task is to classify these character images as one of the original letters (Michie *et al.*, 1995). Transformations as in Hsu and Lin (2002).
9	Sensorless	48	11	1.00	Based on phase current measurements of an electric motor, predict different error conditions (Paschke *et al.*, 2013). We use the transformations from Wang *et al.* (2018*b*).
10	senseit_aco	50	3	0.95	Predict vehicle type using acoustic data gathered by a sensor network (Duarte and Hu, 2004).
11	senseit_sei	50	3	0.94	Predict vehicle type using seismic data gathered by a sensor network (Duarte and Hu, 2004).
12	covtype	54	7	0.62	Predicting forest cover type from cartographic variables (Blackard, 1998; Dua and Graff, 2017).
13	connect-4	126	3	0.77	Predict if the first player wins, loses or draws, based on board positions of the board game *Connect Four* (Dua and Graff, 2017).

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 $\{1, 2, ..., min(depth(T_{opt}), 15)\}$, 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, for a multi-class problem, 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: $\{1, 2, ..., min(d, 15)\}$. 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 3.1).

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).

Model selection within an optimizer iteration is performed by averaging over three

runs of holdout cross validation, with the ratio of the training to holdout data size being $80:20$. The splits are randomly determined and are stratified wrt class labels.

Baselines: For each model family, the baseline model is trained using standard (i.e., without using our algorithm and without recourse to an oracle) 3-fold cross-validation, stratified by class labels. In the case of DTs, the space of the parameter $min\_impurity\_decrease = \{0, 0.25, 0.5, 0.75, 1\}$ is also explored. This parameter enforces node splits to be executed only if the decrease in impurity is at least as large as the parameter's value[10].

It must be noted that the baseline model has access to more data for training than models trained within the optimizer: the former combines D_{train} and D_{val} for its 3-fold CV, while the latter has access to only D_{train}. With our splits of $D_{train} : D_{val} : D_{test} :: 60 : 20 : 20$, the baseline sees $80/60$ or $1.33x$ more data. In fact, often the difference is greater since at a particular iteration, the parameter N_s might be set to a value much lower than $|D_{train}|$. The reduced training data size is why the within-optimizer model selection doesn't use a 3-fold CV, since that would enforce a training split of 67%, as opposed to 80% that we use now.

3.3.3 Metrics

We measure the improvements in accuracy and the their statistical significance using the following metrics:

1. For each combination of dataset, learning algorithm and model size, the percentage *relative improvement* in the $F1$(macro) score is measured over (X_{test}, Y_{test}). The *baseline* score is provided by a model trained on the original distribution:

$$\delta F1 = \frac{100 \times (F1_{new} - F1_{baseline})}{F1_{baseline}} \tag{3.13}$$

   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

[10] See documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

poorly on a minority class.

Since the original distribution is part of the optimization search space, i.e., when $p_o = 1$, the lowest improvement we report is 0%, i.e., $\delta F1 \in [0, \infty)$.

All reported values of $\delta F1$ represent averaging over **five** runs of Algorithm 7, where we average the baseline and new scores *first*, and then calculate the improvement. In other words, if the runs are indexed by i, $F1_{new}$ and $F1_{baseline}$ are replaced by $\overline{F1}_{new} = \sum_{i=1}^{5} F1_{new,i}/5$ and $\overline{F1}_{baseline} = \sum_{i=1}^{5} F1_{baseline,i}/5$ respectively, in Equation 3.13. 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_t.

We take an average of the scores first since $F1_{baseline}$ can be a small value, especially at smaller model sizes, and being in the denominator, slight changes to it across runs can produce outsize differences in the per-run $\delta F1$ scores[11].

2. To measure statistical significance of our results we use the *Wilcoxon signed-rank test*, where the paired set of samples are $F1_{baseline}$ and $F1_{new}$ scores for a dataset. The *p-value* is reported. This test is separately performed for different model sizes.

3.3.4 Parameter Settings

Since TPE performs optimization with *box constraints*, we need to specify our search space for the various parameters in Algorithm 7:

1. λ: 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_{train}, Y_{train}). Hence, we set $p_o \in [0, 1]$.

3. N_s: We set $N_s \in [1000, 10000]$. The lower bound ensures that we have statistically significant results. The upper bound is set to a reasonably large value.

[11] This is different from the improvements reported in our paper, Ghose and Ravindran (2020), where the improvement scores from individual runs were averaged. We find the current approach more stable.

4. α: 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, α_{max}. Given α, for N points, the expected number of components k is given by:

$$E[k|\alpha] = O(\alpha H_N) \tag{3.14}$$

$$E[k|\alpha] \leq \alpha H_N \tag{3.15}$$

$$\alpha \geq \frac{E[k|\alpha]}{H_N} \tag{3.16}$$

   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, $k_{max} = 1 +$ depth of density tree. We use $N = 1000$, since this is the lower bound of N_s, and we are interested in the upper bound of α (note $H_N \propto N$ - see Section A.3). We set $k_{max} = 100$ (this is greater than any of the density tree depths in our experiments) to obtain a liberal upper bound, $\alpha_{max} = 100/H_{1000} = 13.4$. Rounding up, we set $\alpha \in [0.1, 14]$ [12].

   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.

Hyperparameters: The box constraints and the iteration budget required by the optimizer constitute task-specific hyperparameters. However, as we note above, we don't need to estimate a range for p_o and reasonable defaults may be applied to N_s, $\{a, b, a', b'\}$, λ and α. This results in the practical convenience of having to set the value for only a single hyperparameter: T, the iteration budget. This was set to $T = 1000$ for LPMs and $T = 3000$ for DTs based on limited search. Since the LPMs we use construct multiple *one-vs-rest* classifiers, higher iteration budgets are computationally expensive to use.

[12] 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 3.13, 3.14).

Table 3.2: Classification Results with DTs. Values indicate improvements $\delta F1$, averaged over five runs. Underlined entries denote the best improvement for a dataset.

depth = / datasets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cod-rna	3.89	0.82	0.24	2.18	0.29	0.12	0.22	0.22	0.17	0.00	-	-	-	-	-
ijcnn1	3.19	12.91	8.64	8.94	3.35	1.19	2.35	1.20	0.91	0.00	0.25	0.39	0.60	0.00	0.69
higgs	3.68	0.93	0.92	0.44	0.00	0.46	-	-	-	-	-	-	-	-	-
covtype.binary	0.41	0.34	0.78	0.78	0.76	0.75	0.41	0.57	0.63	0.81	0.00	1.64	-	-	-
phishing	0.00	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	5.05	5.58	4.75	4.07	2.39	3.40	0.82	1.83	-	0.65	0.00	2.35	-	0.92
pendigits	10.48	3.08	4.44	9.79	4.66	2.18	1.04	0.07	0.09	0.00	0.00	0.00	0.00	0.00	0.10
letter	1.06	10.49	32.31	46.65	45.91	20.68	9.32	5.04	4.21	2.13	0.53	0.00	0.00	0.00	0.00
Sensorless	0.00	41.98	71.18	80.30	37.82	16.74	7.85	4.29	2.19	1.09	0.63	0.16	0.75	0.28	0.00
senseit_aco	19.64	0.57	3.09	1.46	1.48	0.46	0.38	0.93	-	-	-	-	-	-	-
senseit_sei	2.18	1.02	2.15	0.53	0.96	1.01	0.42	-	-	-	-	-	-	-	-
covtype	27.80	99.72	18.36	6.74	6.23	2.42	2.11	2.43	1.65	2.03	2.61	1.69	0.00	0.67	0.53
connect-4	181.33	32.28	17.96	12.10	6.97	10.60	3.97	6.07	2.62	1.15	2.71	2.01	2.03	1.97	1.50

3.3.5 Improvements in Accuracy

The DT results are shown in Table 3.2. 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 ex in Table 3.2, *cod-rna* has an optimal size of 10.

For each dataset, the best improvement across different sizes is shown underlined. The horizontal line separates binary datasets from multiclass datasets.

This data is also visualized in Figure 3.10. The x-axis shows a scaled version of the actual tree depths for easy comparison: if the largest actual tree depth explored is η_{max} for a dataset, then a size η is represented by η/η_{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 0.71% 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 3.2. This is inline with Equations 3.3 and 3.4: beyond a model size η', $\delta F1 = 0\%$.

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 3.10. However, if we separate the binary and multiclass results, as in Figure 3.11, we note

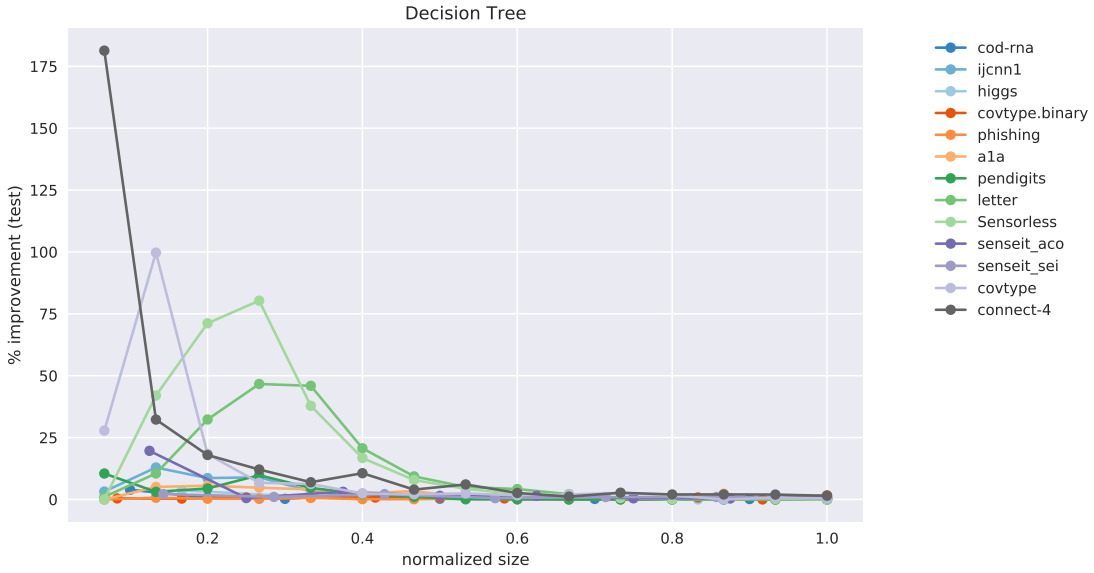
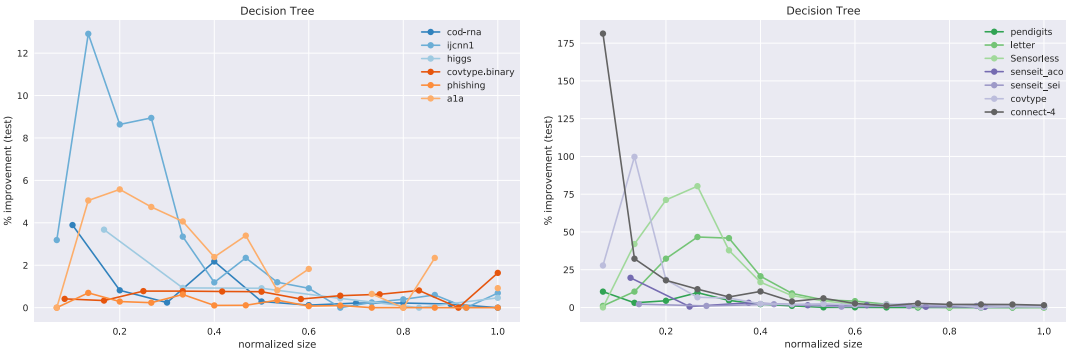


Figure 3.10: Improvement in F1 score on test with increasing size. Data in Table 3.2.

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 3.1, the datasets with most imbalance are *ijcnn1, covtype, connect-4*, for which we see best improvements of $12.91\%, 99.72\%, 181.33\%$ respectively.



(a) binary datasets

(b) multiclass datasets

Figure 3.11: Performance on (a) binary vs (b) multi-class classification problems using CART. This is an elaboration of Figure 3.10.

Figure 3.12 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* (the last column in Table 3.2 for these datasets are either empty or tending to 0). We observe that indeed $p_o \to 1$ as our model grows to the optimal size. This empirically

validates our hypothesis from Section 3.1.1, 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.

Figure 3.12: Variation of p_o with increasing model size.

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 $\delta F1$. For instance, if we have experimented with 3 model sizes, and $\delta F1$ are $7\%, 11\%$ and 2%, we sample $0.35N, 0.55N$ and $0.1N$ 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 $N = 10000$.

Figure 3.13 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.

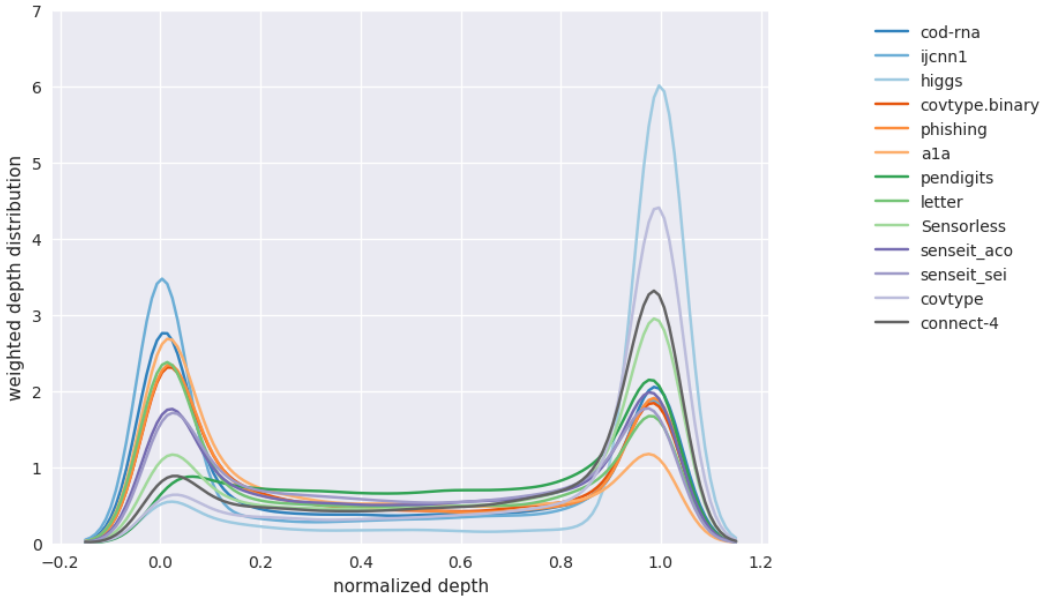


Figure 3.13: Distribution over levels in density tree(s). Aggregate of distribution over different model sizes.

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 3.13; hence using a distribution across the height of the density tree is still a useful strategy.

Table 3.3: Classification Results with LPMs. Values indicate improvements $\delta F1$, averaged over five runs. Underlined entries denote the best improvement for a dataset.

# terms = / datasets	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cod-rna	6.24	13.68	20.12	22.46	22.95	30.04	11.66	3.90	-	-	-	-	-	-	-
ijcnn1	6.40	3.54	1.39	2.06	3.18	2.08	2.15	3.04	3.22	3.25	2.93	3.22	2.73	3.28	3.10
higgs	13.49	10.36	7.94	4.52	5.29	4.10	4.26	2.47	2.82	2.99	3.61	2.62	2.70	2.31	1.99
covtype.binary	24.30	22.99	25.31	11.87	7.06	7.25	4.50	3.65	5.86	5.44	6.18	6.01	6.09	5.81	5.23
phishing	0.00	1.11	1.40	1.66	1.84	2.17	2.17	1.08	0.81	0.68	0.63	0.51	0.64	0.87	1.84
a1a	0.00	8.50	21.15	21.26	18.22	14.65	6.10	4.79	7.33	7.40	2.96	5.76	5.41	3.99	3.91
pendigits	8.40	9.31	9.87	6.91	8.74	4.00	4.15	0.91	1.12	0.47	0.49	0.28	1.00	0.50	0.59
letter	11.76	9.22	19.06	10.39	10.50	3.29	2.60	2.19	2.06	2.90	2.43	2.21	3.62	4.93	4.83
Sensorless	69.62	56.36	28.76	11.73	14.48	14.07	15.33	23.07	20.56	24.61	27.08	30.47	39.85	40.88	38.90
senseit_aco	4.51	59.23	29.82	19.38	16.72	11.30	11.79	8.18	5.57	4.39	3.50	2.79	2.84	1.39	1.76
senseit_sei	147.09	46.02	18.89	7.68	2.54	0.96	1.11	1.38	0.95	1.26	1.12	1.36	1.34	0.31	0.45
covtype	27.47	20.16	6.15	4.95	2.24	6.82	2.77	5.00	7.56	7.11	6.90	8.07	7.36	9.17	8.60
connect-4	33.47	18.24	18.19	11.41	6.14	3.78	4.79	3.85	4.60	3.49	3.17	1.87	1.92	0.66	2.34

The results for LPM are shown in Table 3.3. 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 (*senseit_sei* seems to be an exception). 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 3.14 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 3.13.

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 sizesHere too, class imbalance doesn't seem to play a role (datasets with most imbalance - *ijcnn1, covtype, connect-4* - show best improvements of $6.4\%, 27.47\%, 33.47\%$ respectively.

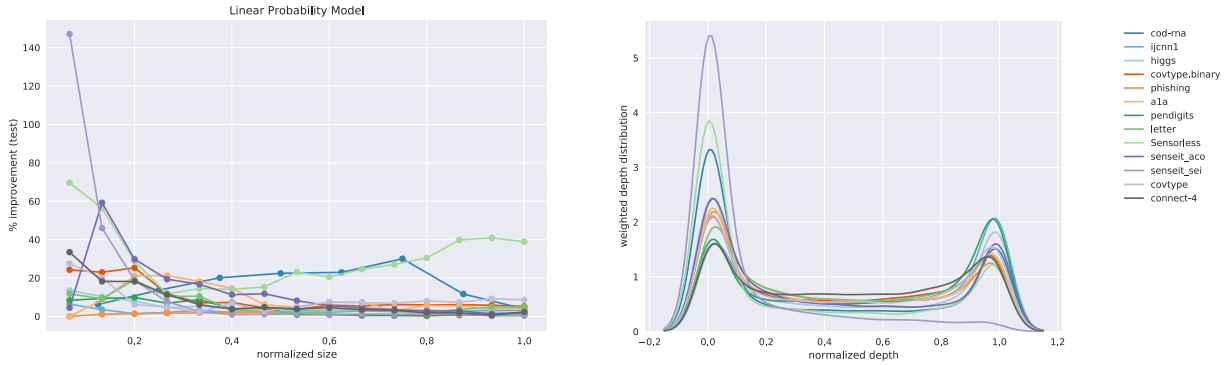


Figure 3.14: Linear Probability Model: improvements and the distribution over depths of the density trees.

3.3.6 Statistical Significance

We perform the *Wilcoxon signed-rank test* (Wilcoxon, 1945) to measure statistical significance of the $\delta F1$ scores presented in the previous section. These are shown in Figure 3.15. We use this test as it has been shown to be useful in comparing classifiers (Demšar, 2006; Benavoli *et al.*, 2016; Japkowicz and Shah, 2011). The test setup is as follows:

1. We compare the classifiers learned by our technique with the baseline, for a given range of model sizes. Separate tests are performed for different model size ranges since size strongly influences $\delta F1$.

2. Normalized model sizes are used for ease of comparison with Figure 3.10 and Figure 3.14. Binning of model sizes is done using *Sturges rule* (Sturges, 1926).

3. The *one-sided* version of the *paired* test is performed for each bin, where pairs of scores $F1_{baseline}$ and $F1_{new}$ for a dataset, for models with sizes assigned to the bin, are compared. In cases were where multiple model sizes for a dataset fall within the same bin, $F1_{baseline}$ and $F1_{new}$ are first averaged and then compared.

4. The following hypotheses are tested:

   - $\mathbf{H_0}$, null hypothesis: accuracies of models trained using density trees are not better.
   - $\mathbf{H_1}$, alternate hypothesis: accuracies of models trained using density trees are better.

p-values are shown for each bin. Small *p-values* favor $\mathbf{H_1}$, i.e., our algorithm.

5. Scores of $\delta F1 = 0$ are split equally between positive and negative ranks[13].

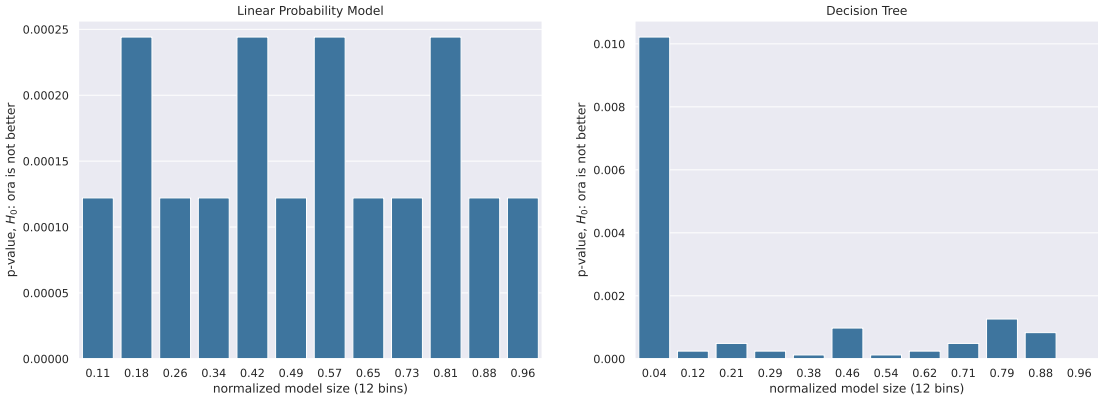


Figure 3.15: These plots show the *p-values* for the Wilcoxon signed-rank test, with the null hypothesis H_0: using density trees do *not* produce better F1 test scores. The bin boundaries are selected using the *Sturges* rule (Sturges, 1926). Low *p-values* favor our algorithm.

We observe that the improvements from using our technique are indeed significant for most model sizes for either model, as measured across multiple datasets.

3.3.7 Effect of Model Capacity

An interesting question to ask is how, if at all, the *bias* of the model family of $\mathcal{F}$ in Algorithm 7, 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_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 $max\_depth = 2$ and $max\_depth = 5$ as representing weak and strong model families respectively. We use the *LightGBM* library (Ke *et al.*, 2017) for our experiments.

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

[13]The `zplit` option in https://numpy.org/doc/stable/reference/generated/numpy.histogram_bin_edges.html is used.

Table 3.4: Classification Results with GBMs. Both $F1_{new}$ and $\delta F1$ are shown. Please see text for explanation of the highlighting scheme.

		boosting rounds =	1	2	3	4	5	6	7	8	9	10
datasets	max depth	score type										
Sensorless	2	$F1_{new}$	0.76	0.77	0.78	0.80	0.80	0.80	0.81	0.81	0.81	0.81
		$\delta F1$	3.19	3.35	3.10	5.05	4.12	1.75	3.21	1.96	1.90	2.43
	5	$F1_{new}$	0.91	0.92	0.93	0.94	0.94	0.94	0.94	0.95	0.95	0.95
		$\delta F1$	0.29	0.25	0.16	0.40	0.00	0.18	0.36	0.30	0.00	0.26
senseit_aco	2	$F1_{new}$	0.22	0.24	0.31	0.37	0.52	0.59	0.61	0.62	0.63	0.63
		$\delta F1$	0.00	6.81	41.41	67.44	69.29	9.39	6.83	4.70	2.33	1.10
	5	$F1_{new}$	0.22	0.30	0.42	0.51	0.58	0.62	0.65	0.66	0.67	0.68
		$\delta F1$	0.00	36.80	85.44	46.66	9.72	2.91	1.17	0.34	0.39	0.40
senseit_sei	2	$F1_{new}$	0.60	0.60	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.62
		$\delta F1$	171.08	171.28	173.05	174.66	173.47	165.00	78.73	48.52	24.61	17.63
	5	$F1_{new}$	0.62	0.64	0.64	0.64	0.64	0.65	0.64	0.64	0.65	0.66
		$\delta F1$	180.46	185.59	186.24	181.13	64.66	28.13	11.30	3.11	1.37	0.59

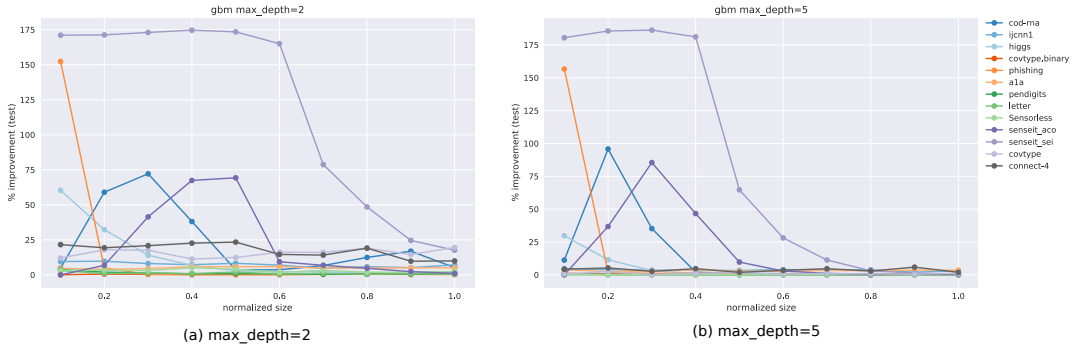
See Table A.1 for complete data, Fig 3.16 for plot.

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 3.4 in the interest of space. The complete results are made available in Table A.1 in the *Appendix*. We present both the improvement in the $F1$ score, $\delta F1$, and its new value, $F1_{new}$.

Figure 3.16 shows improvements for GBMs of different maximum depths for its trees: for (a) $max\_depth = 2$ and for (b), $max\_depth = 5$.



(a) max_depth=2 (b) max_depth=5

Figure 3.16: GBMs with (a) $max\_depth = 2$ and (a) $max\_depth = 5$. Size is the number of boosting rounds. Note the relatively faster drop in improvements in (b).

The cells highlighted in blue in Table 3.4 are where the GBM with $max\_depth = 2$ showed a larger improvement than a GBM with $max\_depth = 5$ 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 Figure 3.16, we see that improvements continue up to larger sizes when $max\_depth = 2$ (also evident from Table 3.4). This is not surprising: we expect a stronger model to extract patterns from data at relatively smaller sizes, compared to a weaker model.

Observe that in Table 3.4, for the same number of boosting rounds, the new scores $F1_{new}$ for the weaker GBMs are up to as large 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.

3.3.8 Summary

Summarizing our analysis above:

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 towards the original distribution. This is conveniently indicated with $p_o \to 1$ as η increases.

   (b) There is model size η', beyond which $\delta F1 \approx 0\%$.

3. For all the model families experimented with - DTs, LPMs, GBMs (results in Table A.1) - 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, $\mathcal{F}$, 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 $\delta F1$ 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 $\delta F1$ 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 $\delta F1$ for a weaker family may exceed one for a stronger family, the improved score $F1_{new}$ may, at best, match it.

6. The depth distribution seems to favour 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.

3.4 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 some of our design choices.

Conceptually, Algorithm 7 consists of quite a few building blocks. Although we have justified our implementation choices for them in Section 3.2, 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 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: α). Considering our results in Figures 3.13 and 3.14 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 us 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[14].

[14]Interestingly, another recent paper on interpretability does use the Kumaraswamy distribution (Bastings *et al.*, 2019).

3.5 Conclusion

The technique presented in this chapter 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.

This technique possesses multiple salient properties:

1. The optimization step uses a fixed set of eight variables, irrespective of the dimensionality of the data.
2. A reasonable choice of box constraints over the search space produces good results across datasets.
3. The technique is model-agnostic, allowing for use with an arbitrary interpretable model family.
4. Its a framework, which leaves open the possibility of conveniently improving upon it as better optimizers become available.

In addition to the above, we believe this work is innovative in the following ways:

1. It highlights of a "small model effect": the optimal training is different from the test distribution at small model sizes. This challenges the conventional wisdom that the training data must be drawn from the same distribution as the test data.
2. Use of the "depth distribution" over a DT to (a) control the extent of information about class boundaries (b) reduce the size of the optimization search space.

The results presented here suggest some novel applications and directions for future work - these are discussed in Section 5.3.

CHAPTER 4

Compact Models using Probabilistic Oracles

Since we use a forest of density trees to guide the training of an interpretable model, it is reasonable to assume that their classification accuracy limits the accuracy of the resultant interpretable model. This naturally leads us to ask: can we use an arbitrarily powerful guiding model? We propose a technique to do so in this chapter, answering the question in the affirmative.

At a high-level, our technique consists of the following steps:

1. We first learn an *oracle*: a highly accurate, possibly black-box, *probabilistic* model trained on the training data. It produces a probability distribution over labels for an instance x:

$$p(y_i|x), \forall y_i \in \{1, 2, ..., C\} \tag{4.1}$$

   Here, $\{1, 2, ..., C\}$ is the set of labels. The probabilities $p(y_i|x)$ may be informally construed as *confidences* of predicting labels y_i for instance x.

2. Next, we try to incorporate the oracle's implicit representation of class boundaries into our interpretable model. The mechanism used is to sample points from the training data based on a learned distribution over the *uncertainty* in the oracle's predictions[1].

3. The interpretable model is then trained on this sample.

We empirically show that this technique also leads to significant improvements in the classification accuracy, especially when the interpretable model size is small. Additionally, these improvements are greater than when using density trees.

[1] This is different from *Knowledge Distillation*; discussed in Section 1.6.

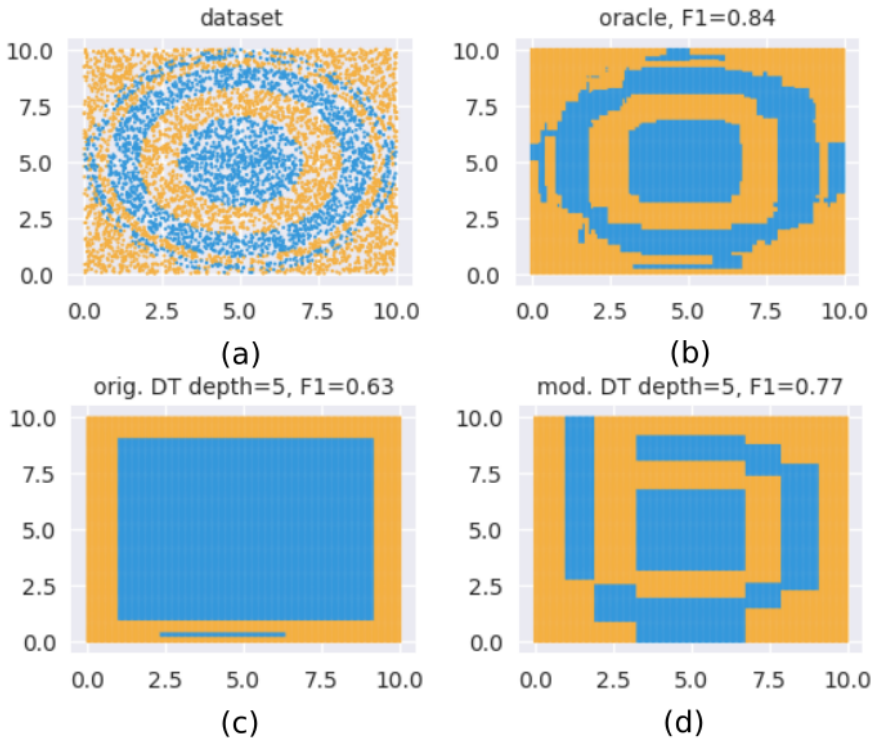


Figure 4.1: A demo of our technique using a GBM as an oracle. See text for explanation.

We visualize the use of such on oracle in Figure 4.1. The two-label balanced dataset we want to classify is shown in Figure 4.1(a). Figure 4.1(b) visualizes the generalization learned by a *Gradient Boosted Model (GBM)* using this dataset. This serves as our oracle with an $F1$ score of $= 0.84$. Figure 4.1(c) shows what a *CART* (Breiman *et al.*, 1984) decision tree of $depth = 5$ learns; here $F1 = 0.63$. Finally, Figure 4.1(d) shows what a CART decision tree of $depth = 5$ learns, when we supply the GBM as an oracle to our technique. There is a significant improvement with $F1 = 0.77$. Visually, we see the boundaries approximating the ones learned by the oracle in Figure 4.1(b).

The key contributions of the work presented in this chapter are:

1. The proposed algorithm identifies a sampling distribution over a training dataset that is optimal in terms of achieving high test accuracy, for a provided model family and model size.

2. We confirm the following "small model effect": *in general, the optimal training distribution is not the same as the test distribution, especially at small model sizes. As model size increases the optimal training distribution progressively approximates the test distribution.*

3. This algorithm is *model-agnostic* in that both the interpretable model and the oracle may belong to arbitrary model families, e.g., these can be *Linear Probability Model* and a *Random Forest*, or even a decision tree and a *Gated Recurrent Network (GRU)* respectively. It also admits a flexible notion of model size, e.g., depth of a decision tree, number of terms with non-zero coefficients in a linear model, number of trees *and* maximum depth per tree in a GBM model.

4. The sampling algorithm internally solves an optimization problem to identify the optimal distribution; however, in our formulation only a fixed number of *seven optimization variables* are required irrespective of the dimensionality of the data.

5. The proposed technique may be used as a tool to identify and study the optimal training data for a given data size, for a model.

Contrasted with the density tree approach, our contributions are:

1. In general, improvements in model accuracy are greater.

2. The ability to use an oracle from an arbitrary model family results in the practical benefit that it need not be learned from scratch. If there is already a pre-trained probabilistic model like a *deep neural network* available for a dataset, it may be conveniently plugged into our algorithm as-is.

3. The density trees and the interpretable model had to be constructed on the same (or very similar) feature space. Here, this is not required, and the oracle might be a sequence model that classifies text, while the interpretable model may be an n-gram based classifier. This considerably broadens the scope of our technique. We look at an example in Section 4.3.3.1.

The remainder of the chapter is structured as follows: in Section 4.1, we present an overview of the work. Section 4.2 discusses the algorithm in detail while Section 4.3 presents extensive experimental validation using real-world datasets. In Section 4.4 we discuss the results and their implications. Section 4.5 concludes the chapter with a summary of our contributions. A discussion on future work is deferred to Section 5.3.

4.1 Overview

In this section we present the intuition behind the proposed technique, a formal statement of the problem, and then a discussion of where our technique fits in within the standard model-building workflow. Finally, we establish the terminology relevant to the remaining chapter.

For a discussion on previous work, see Section 1.6, where we differentiate our technique with *Knowledge Distillation*, *Active Learning*, *Transfer Learning* and *Coreset* identification.

4.1.1 Intuition

Our intuition here builds upon certain observations from density tree based technique discussed in the previous chapter. There, to find an optimal training distribution, we learned two kinds of distributions for a density tree:

1. A pdf along the height of the tree.

2. A pmf across nodes at a particular depth. Although every depth has its own pmf, they are parameterized by a common parameter λ.

Figure 4.2 visualizes these.

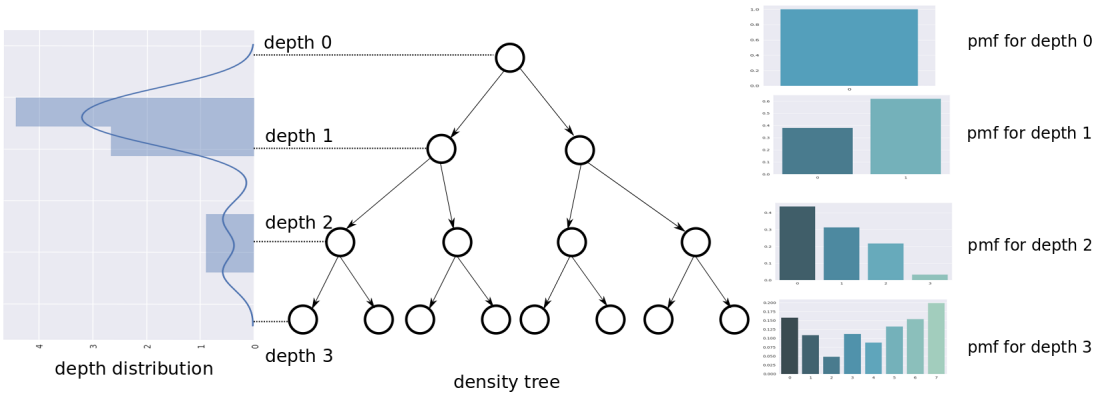


Figure 4.2: A schematic showing the two distributions we learn for a density tree - a pdf along the height and a pmf across the nodes at a particular depth.

To extend this technique, we need an arbitrary probabilistic model to provide information equivalent to that encoded by these distributions. We deconstruct this requirement as follows:

1. The equivalence that's probably easiest to see is for the information available at the leaf level. Regions with small volumes that indicate proximity to a class boundary may be substituted by regions of high prediction uncertainty.

   Figure 4.3 compares (a) the leaf-level tessellation of a density tree with (b) high uncertainty regions (darker in shade) obtained using a probabilistic SVM.

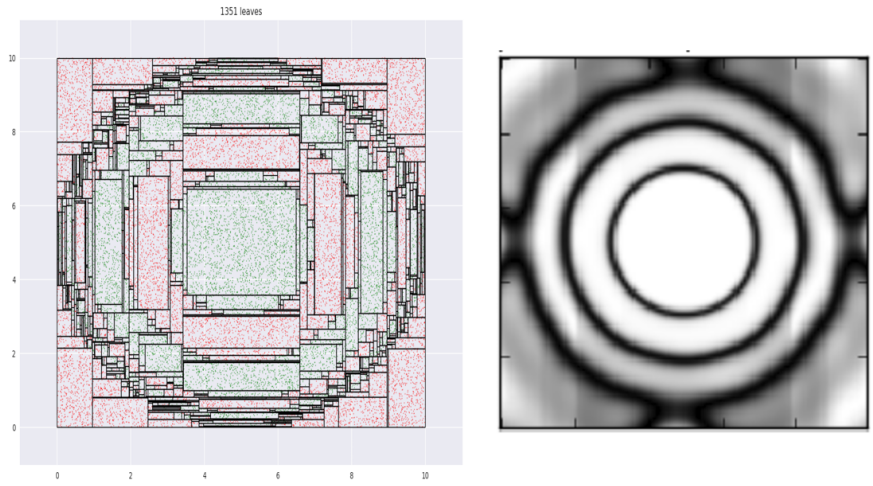


Figure 4.3: (a) Tessellation found by a density tree - this is a reproduction of Figure 3.3. (b) Uncertainty scores from a SVM are shown, where darker shades imply greater uncertainty. Note how low volume regions in (a) correspond to darker regions in (b).

2. At the root level there is no class boundary information available and we have just one node to sample from; this is trivially approximated by the original distribution.

3. There is probably no general model artifact that directly corresponds to the intermediate depths. However, recall our observation that *in general*, their contribution is *relatively less significant* than the root and leaf levels. Figure 4.4 reproduces a plot from the previous chapter that shows some learned depth distributions - note the low contributions from the intermediate depths.

   This suggests the possibility that we might *ignore* modeling this specific aspect of density trees. It might seem that this is a cost we must pay for generalizing

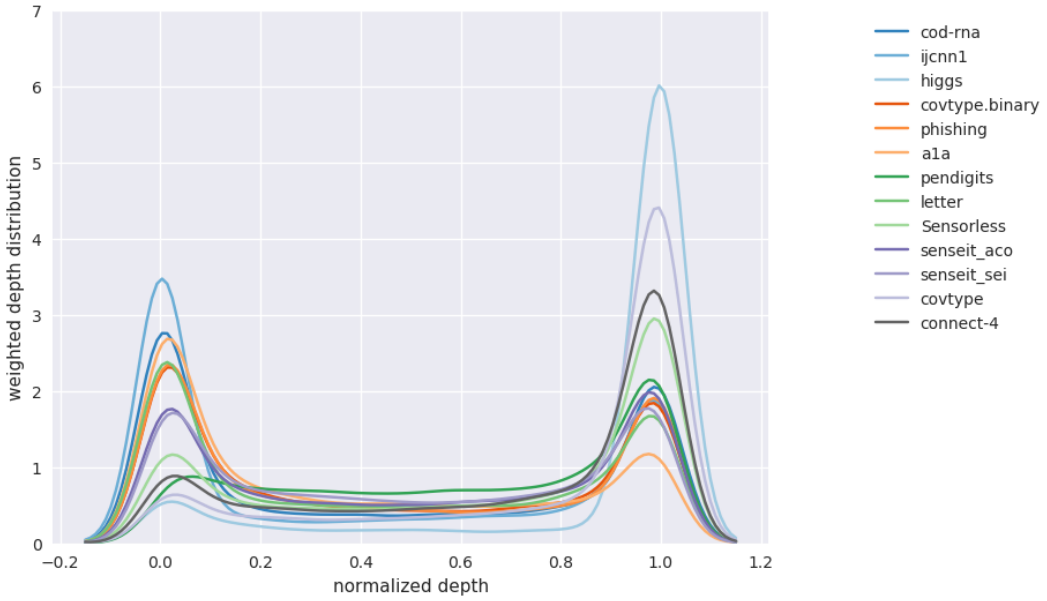


Figure 4.4: Distribution over levels in density tree(s) - this is a reproduction of Figure 3.13. This plot suggests that in general, the intermediate depths are not as important as the root and leaf level distributions.

our technique, but our empirical validation (Section 4.3.2) demonstrates no negative impact. On the contrary, the technique proposed in this chapter outperforms density trees.

4. The *pmf* distribution for nodes may be replaced by a continuous distribution over uncertainty values.

5. The following parameters are not specific to density trees and therefore, may be retained as-is:

   - N_s: number of instances to sample from the training dataset.
   - p_o: fraction of samples from the original distribution.

These equivalences (summarized in Table 4.1) inform the key intuition behind this work: the problem of learning density trees may be extrapolated to learning a distribution over uncertainty scores. Of course, we validate the resulting technique rigorously in Section 4.3.2.

Table 4.1: Information sources compared across density tree and oracle based approaches.

	Density Tree	**Oracle**
1	Original distribution, controlled by p_o	Original distribution, controlled by p_o
2	Root node	Approximated by original distribution, controlled by p_o
3	Intermediate depths	Ignored, owing to low impact
4	Leaf nodes	Uncertainty information

4.1.2 Formal Statement

We extend the terminology of the previous chapter (Section 3.1.4) to formalize the outcomes here. Let:

1. $accuracy(M, p)$ be the classification accuracy of model M on data represented by the joint distribution $p(X, Y)$ of instances X and labels Y. The term "accuracy" is used in a generic sense to represent prediction accuracy; depending on the application, this might be *F1-score*, *AUC*, *lift*, etc.

2. $train_{\mathcal{F},f}(p, \eta)$ produce a model obtained using a specific training algorithm f, e.g., CART (Breiman *et al.*, 1984), for a given model family $\mathcal{F}$, e.g., DTs, where the model size is fixed at η, e.g., trees with $depth = 5$. p represents the joint distribution $p(X, Y)$ of instances X and labels Y. $train_{\mathcal{F},f}(p, *)$ denotes there are no constraints imposed on the model size.

Then, we claim, and empirically demonstrate, that *the interpretable model trained on the sample generated by our learned distribution is at least as accurate as one learned on the original training data, and is up to as accurate as the oracle*:

$$accuracy(M_{\mathcal{I}p\eta}, p) \lessapprox accuracy(M_{\mathcal{I}q\eta}, p) \lessapprox accuracy(M_{\mathcal{O}p*}, p) \tag{4.2}$$

where,

$$M_{\mathcal{I}p\eta} = train_{\mathcal{I},g}(p, \eta)$$

$$M_{\mathcal{I}q\eta} = train_{\mathcal{I},g}(q, \eta)$$

$$M_{\mathcal{O}p*} = train_{\mathcal{O},h}(p, *)$$

Here,

- For a model named M_{ABC}, this is what the subscripts denote:

  1. A signifies if the model is an oracle or an interpretable model, with symbols $\mathcal{O}$ and $\mathcal{I}$ respectively.
  2. B denotes the training distribution.
  3. C is the model size.

- g and h represent specific training algorithms, e.g., *CART* for DTs, *rmsprop* (Graves, 2013) for neural networks. These are omitted in model names for brevity, and are made clear by context.

- We refer to $M_{\mathcal{I}p\eta}$ as the "baseline model", since this is the standard way of training a model against which we evaluate our approach.

- p and q both denote joint distributions of X and Y. $p(X, Y)$ is the distribution we are provided, and *all* our models use this as the *test* distribution. $q(X, Y)$ is the distribution we learn using the uncertainty scores provided by the oracle $M_{\mathcal{O}p*}$.

  Note that, typically, the train and test distributions are identical for a model, as in the terms $accuracy(M_{\mathcal{I}p\eta}, p)$ and $accuracy(M_{\mathcal{O}p*}, p)$. However, for the middle term in Equation 4.2 - $accuracy(M_{\mathcal{I}q\eta}, p)$ - the train and test distributions, q and p respectively, are different.

- The use of the "$\lessapprox$" symbol emphasizes these relationships are validated empirically using samples from the corresponding distributions p and q.

We also show that Equation 4.2 can be further refined into two size-regimes: the interpretable model trained on the new sample is more accurate than the baseline model only until a model size η^*. At sizes greater than η^* the model performances are equal:

$$accuracy(M_{\mathcal{I}p\eta}, p) < accuracy(M_{\mathcal{I}q\eta}, p) \lessapprox accuracy(M_{\mathcal{O}p*}, p), \text{ when } \eta \leq \eta^* \quad (4.3)$$

$$accuracy(M_{\mathcal{I}p\eta}, p) = accuracy(M_{\mathcal{I}q\eta}, p) \lessapprox accuracy(M_{\mathcal{O}p*}, p), \text{ when } \eta > \eta^* \quad (4.4)$$

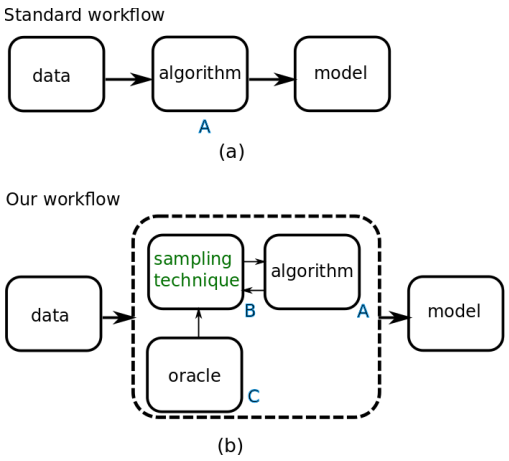


Figure 4.5: Modified workflow. Arrows denote flow of information. Our sampler B receives uncertainty information from the oracle C, which it then uses to iteratively learn a distribution, using the performance of A as its objective function.

4.1.3 Workflow

Figure 4.5 compares (a) a standard workflow to our (b) model building workflow. The arrows represent flow of information. In the standard setup, a model training algorithm, A, accepts training data and produces a model that maximizes some pre-defined prediction accuracy metric. Our workflow adds two new components - the adaptive sampling technique, B, and an oracle, C. The oracle provides information to the sampling technique, that enables it to identify a potentially "better" sample from the training data for input to algorithm A. Here, a "better" sample is the one that leads A to produce a model with the higher accuracy (measured on a held-out dataset), compared to training on the provided data as-is. Determining this sample is an iterative process; at each iteration, B modifies the sample based on the current accuracy of the model from A. The information from the oracle is conveyed to the sampling technique only once, before the beginning of the iterative interaction between A and B.

4.1.4 Terminology and Notation

We first define the notion of *model size* since it is critical for subsequent discussions. Model size is a model parameter with the following properties:

1. $model\_size \propto bias^{-1}$

2. The interpretability of a model decreases with increasing model size.

Only the first criteria above is required for using our technique. The second criteria reflects the usefulness of the technique for interpretability.

It must be noted that the notion of model size is subjective. Consider a GBM with DTs as base classifiers: here, the depth of the individual trees, or the number of trees, or both collectively may be seen as representing size. Even for a given notion of size, the value up to which a model is considered interpretable may be a matter of opinion. For example, some might consider a DT with $depth = 15$ to be interpretable, while some might decide $depth = 10$ to be the limit for interpretability. However, as long as the notion of size satisfies the criteria above, the discussion in this chapter applies.

We now introduce the notations used:

1. We denote a dataset, D, by a set of instance-label pairs, i.e., $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where x_i is the feature vector representing an instance and y_i is its corresponding label.

   Sometimes, we use *multisets*, when instance-label pairs may be repeated. Such usage is explicitly called out.

2. While we have referred earlier to the joint distribution of instances and labels, e.g., $p(X, Y)$ in Equation 4.2, this is understood to represent the dataset that we are actually given, in the form of a finite number of instance-label pairs.

3. We use the term *original*, as in *original distribution* or *original data* to denote the data that we are given. This is in contrast with samples we generate. The distribution of test datasets or held-out datasets is the original distribution for *all models discussed in this chapter*.

4. The terms $accuracy()$ and $train_{\mathcal{F},f}()$ are overloaded to accept a dataset as input in lieu of a distribution:

   - $accuracy(M, D)$ denotes the accuracy of model M with the dataset D as the test set.
   - $train_{\mathcal{F},f}(D, \eta)$ denotes a specific training algorithm f, for a model family $\mathcal{F}$, that accepts as input a dataset D, and trains a model of size η.

5. 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.

□

Next, we look at our methodology.

4.2 Methodology

We describe our methodology in this section. We begin with the intuition, and then look at the algorithm and various implementation details.

4.2.1 Measuring Uncertainty

We begin by discussing the measurement of uncertainty, since our technique critically depends on this quantity. We denote the uncertainty of prediction by a model M on an instance x by $u_M(x)$, where $u_M(x) \in [0, 1]$. A good uncertainty metric for our application (a) should not exclusively consider the confidence of the predicted label (b) should result in a high value even if the model is uncertain between two labels in a multi-class problem. We consider the following popular uncertainty metrics:

1. **Margin Uncertainty** (Scheffer *et al.*, 2001): This is computed as:

$$u_M(x) \leftarrow 1 - (p_{C_1} - p_{C_2}) \tag{4.5}$$

   Here, p_{C_1} and p_{C_2} denote the probabilities of the most confident and next most confident classes, provided by model M for instance x. Lower differences between the top two probabilities lead to higher scores for this metric.

2. **Least confident**: we calculate the extent of uncertainty w.r.t. the class we are most confident about:

$$u_M(x) = 1 - \max_{y_i \in \{1,2,\ldots,C\}} M(y_i|x) \tag{4.6}$$

Here, we have C classes, and $M(y_i|x)$ is the probability score produced by the model[2].

3. **Entropy**: this is the standard Shannon entropy measure calculated over class prediction confidences:

$$u_M(x) = \sum_{y_i \in \{1,2,\ldots,C\}} -M(y_i|x) \log M(y_i|x) \tag{4.7}$$

We do not use the *least confident* metric since it completely ignores confidence distribution across labels. While *entropy* is quite popular, and does take into account the confidence distribution, we do not use it since it reaches its maximum for only points for which the classifier must be equally ambiguous about *all* labels; for datasets with many labels (one of our experiments uses a dataset with 26 labels - see Table 4.2) we may never reach this maximum. The *margin uncertainty* metric satisfies all our criteria, and this is what we use.

We calibrate (Platt, 1999) the oracles for reliable probability estimates.

Fig 4.6 visually shows what uncertainty values look like for the different metrics. Panel (a) displays a dataset with 4 labels. A probabilistic linear SVM is learned on this, and uncertainty scores corresponding to the metrics "margin", "least confident" and "entropy" are visualized in panels (b), (c) and (d) respectively. Darker shades of gray correspond to high uncertainty. Observe that only the "margin" metric in panel (b) achieves scores close to 1 at the two-label boundaries.

There is no best uncertainty metric in general, and the choice is usually application specific (Settles, 2009).

4.2.2 Density Representation for Uncertainty

Since we want to learn a distribution over uncertainties, $p(u_M(x))$ needs to have a flexible representation. A desiderata for such a distribution is:

[2]The possibly confusing name "least confident" for this idea originated within the context of uncertainty sampling, where we are interested in sampling the most uncertain point, $x^* = \arg\min_x [\max_{y_i \in \{1,2,\ldots,C\}} M(y_i|x)]$, which may be considered to be the instance with the "least most confident label".

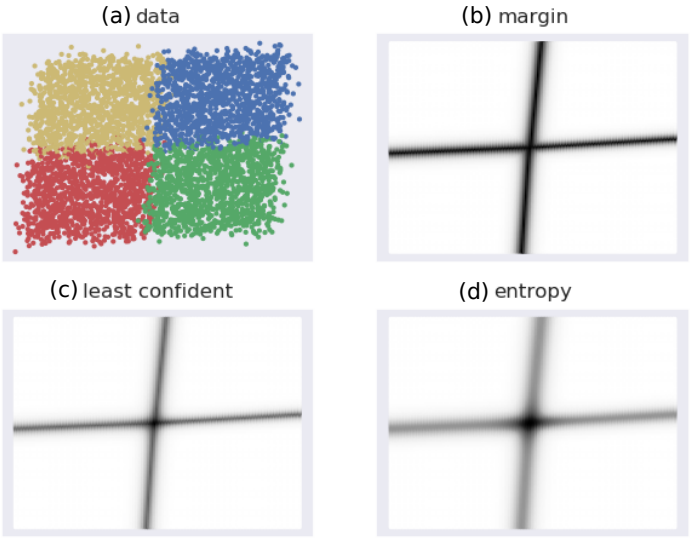


Figure 4.6: Visualizations of different uncertainty metrics. (a) shows a 4-label dataset on which linear SVM is learned. (b), (c), (d) visualize uncertainty scores based on different metrics, as per the linear SVM, where darker shades imply higher scores.

1. We want the distribution to be able to assume an arbitrary "shape", unlike, say using a normal distribution that is unimodal, and the mode is centered.

2. It should be defined over the bounded interval $[0, 1]$ since $u_M(x) \in [0, 1]$.

3. A *fixed set of parameters* is preferred over a conditional parameter space.

   We list this requirement since the parameters of this distribution are to be learned via optimization, and there are many more optimizers that can handle fixed than conditional parameter spaces. This affords us the flexibility of exploring a much wider variety of optimizers. Further discussed in Section 4.2.4.

As we have seen in the previous chapter, the *Infinite Beta Mixture Model (IBMM)* satisfies the above requirements.

We briefly review it here: the IBMM is a *Dirichlet Process (DP) mixture model* with $Beta$ components. A mixture model allows us to model an arbitrary distribution, satisfying our first requirement. Using $Beta$ components enables support for a bounded interval - this satisfies our second requirement. The DP is described by the *concentration parameter* $\alpha \in \mathbb{R}_{>0}$, which identifies the components that have at least one point assigned to them.The shape parameters of all the $Beta$ components are drawn from

shared prior distributions, which themselves are $Beta$ distributions. Use of a DP, with shared priors, gives us a fixed parameter space; this satisfies our third requirement.

This is how we sample N_s points, from a dataset D, using an oracle M_O:

1. Determine partitioning over the N_s points induced by the DP. We use *Blackwell-MacQueen* sampling (Blackwell and MacQueen, 1973) for this. Let's assume this step produces k partitions $\{c_1, c_2, ..., c_k\}$ and quantities $n_i \in \mathbb{N}$ where $\sum_{i=1}^{k} n_i = N$. Here, n_i denotes the number of points that belong to partition c_i.

2. We determine the $Beta(A_i, B_i)$ component for each c_i. We assume the priors for the $Beta$ parameters are also represented by $Beta$ distributions, i.e., $A_i \sim scale \times Beta(a, b)$ and $B_i \sim scale \times Beta(a', b')$. Since samples from the standard $Beta$ are within $[0, 1]$, we use a parameter $scale$ as a common multiplier to obtain a wide range of A_i, B_i.

   Thus we have exactly two prior $Beta$ distributions associated with our IBMM. Here, a, b, a', b' are positive reals.

3. Repeat for each c_i: for each instance-label pair (x_j, y_j) in our training dataset, we calculate the oracle uncertainty score, $u_{M_O}(x_j)$. We then calculate $p_j = Beta(u_{M_O}(x_j)|A_i, B_i)$. We scale the probabilities across instances to sum to 1. These quantities are used as sampling probabilities for various (x_j, y_j), and n_i points are sampled with replacement based on them.

The parameters for the IBMM are collectively denoted by $\Psi = \{\alpha, a, b, a', b'\}$. The best values for Ψ are learned via an optimization process detailed in Section 4.2.3.

The above procedure is summarized in Algorithm 8. Note that $temp$ and D' are multisets in the algorithm, since we sample with replacement. Accordingly, line 13 uses the **multiset sum**, $\uplus$: if (x_i, y_i) occurs m times in D' and n times within $temp$, then $D' \leftarrow D' \uplus temp$ has $m + n$ occurrences of (x_i, y_i).

Algorithm 8: Sample based on uncertainties and Ψ

Data: Sample size N_s, oracle M_O, dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, IBMM parameters $\Psi = \{\alpha, a, b, a', b'\}$

Result: Sample D', where $|D'| = N_s$

$D' = \{\}$ // assumed to be a multiset

$\{(c_1, n_1), (c_2, n_2), ..., (c_k, n_k)\} \leftarrow$ partition N_s using the DP // Here $\sum_{i=1}^{k} n_i = N_s$.

for $i \leftarrow 1$ **to** k **do**

  $A_i \sim scale \times Beta(a, b)$

  $B_i \sim scale \times Beta(a', b')$

  for $j \leftarrow 1$ **to** N **do**

    $p_j \leftarrow Beta(u_{M_O}(x_j); A_i, B_i)$

  end

  for $j \leftarrow 1$ **to** N **do**

    $p'_j \leftarrow c \cdot p_j$, where $c = 1/\sum_{j=1}^{N} p_j$ // normalize the probabilities

  end

  $temp \leftarrow$ sample with replacement n_i instance-label pairs based on p'_j // assumed to be a multiset

  $D' \leftarrow D' \uplus temp$ // $\uplus$ is the multiset sum

end

return D'

4.2.3 Learning Interpretable Models using an Oracle

We tie together the various individual pieces in this section. We have already discussed the parameters Ψ for the IBMM. Our technique uses two additional parameters:

1. $p_o \in [0, 1]$, proportion of instance-label pairs from the original training data. This parameter serves two purposes: (1) it acts as a "shortcut" for the optimizer to sample from the original distribution, as opposed to determining the right Ψ to do so (2) the relationship of p_o and model size enables us to study the correlation between model size and effectiveness of the original distribution during training.

2. $N_s \in \mathbb{N}$, sample size. Since the sample size can have a significant effect on model performance, we allow the optimizer to determine its best value. N_s is constrained to be at least as large as what is needed for statistically significant results.

The complete set of parameters is denoted by $\Phi = \{\Psi, N_s, p_o\}$, where the IBMM

parameters are denoted by $\Psi = \{\alpha, a, b, a', b'\}$.

Our technique randomly initializes Φ, creates a sample based on Algorithm 8 and the original training data (based on p_o), learns an interpretable model of size η on this sample, and evaluates it on a validation set. Based on the validation score, an optimizer modifies the parameters Φ, and repeats the process. Our stopping criteria is an iteration budget T. Algorithm 9 lists these steps.

Algorithm 9: Learning interpretable model using oracle

Data: Dataset D, model size η, $train_{\mathcal{O},h}()$, $train_{\mathcal{I},g}()$, iterations T
Result: Optimal parameters Φ^*, test set accuracy s_{test} at Φ^*, and interpretable model M^* at Φ^*

Create splits $D_{train}, D_{val}, D_{test}$ from D, stratified wrt labels
$M_O \leftarrow train_{\mathcal{O},h}(D_{train}, *)$
for $t \leftarrow 1$ **to** T **do**
    $\Phi_t \leftarrow suggest(s_1, ...s_{t-1}, \Phi_1, ..., \Phi_{t-1})$ `// see text for initialization at` $t = 1$
    `// Note:` $\Phi_t = \{\Psi_t, N_{s,t}, p_{o,t}\}$ `where` $\Psi_t = \{\alpha_t, a_t, b_t, a'_t, b'_t\}$.
    $N_o \leftarrow p_{o,t} \times N_{s,t}$
    $N_u \leftarrow N_{s\_t} - N_o$
    $D_o \leftarrow$ uniformly sample, with replacement, N_o points from D_{train}
    $D_u \leftarrow$ sample N_u points from D_{train} using **Algorithm 8** with input $(N_u, M_O, D_{train}, \Psi_t)$.
    $D_s \leftarrow D_o \uplus D_u$ `//` D_o, D_u `are assumed to be multisets`
    $M_t \leftarrow train_{\mathcal{I},g}(D_s, \eta)$
    $s_t \leftarrow accuracy(M_t, D_{val})$
end
$t^* \leftarrow \arg\max_t \{s_1, s_2, ..., s_{T-1}, s_T\}$
$\Phi^* \leftarrow \Phi_{t^*}$
$M^* \leftarrow M_{t^*}$
$s_{test} \leftarrow accuracy(M^*, D_{test})$
return Φ^*, s_{test}, M^*

Some details to note in Algorithm 9:

1. Values at $t = 1$ are identical to the case of using density trees (Algorithm 7). Φ is initialized as: $\alpha = 0.1, a = 1, b = 1, a' = 1, b' = 1, N_s = |D_{train}|, p_o = 1$. The values for α, a, b, a', b' are arbitrary, since setting $p_o \to 1$ implies only the original distribution is used for sampling. Given $N_s = |D_{train}|$, this setting mimics the baseline, i.e., training the interpretable model without our algorithm, thus providing the optimizer with a good initial reference point in its search space. This

setting is not equivalent to baseline training because of differences in resources, data and computational - detailed in Section 4.3.1.2.

2. The optimizer is represented by the function call $suggest()$ which takes as input all past parameter values and validation scores. $suggest()$ denotes a generic optimizer; not all optimizers require this extent of historical information.

3. While the training algorithm for the oracle, $train_{\mathcal{O},h}()$ is taken as input, a pre-constructed oracle M_O may also be used. This would eliminate the oracle training step in line 2.

4. $accuracy()$ on the validation data, D_{val}, serves as both the objective and fitness function.

5. Evaluation on the test set, D_{test} is done only once, in line 16, with the model that produces the best validation score.

6. Since we sample with replacement, both temporary datasets D_o and D_u, procured from uniformly sampling the original training data and sampling based on uncertainties respectively, are multisets. Accordingly, line 9 uses the multiset sum operator $\uplus$ to combine them.

7. Since the validation score s_t (line 11) needs to be reliable, in our implementation we repeat lines 7-10 *thrice* and use the averaged validation score as s_t.

8. Class imbalance is accounted for in our implementation when training model M_t in line 10. We either balance the data by sampling (this is the case with a *Linear Probability Model*), or an appropriate cost function is used to simulate balanced classes (this is the case with DTs and GBMs).

It is important to note here that D_{val} *and* D_{test} *are not modified by our algorithm in any way*, and therefore s_t and s_{test} measure the accuracy on the original distribution.

Algorithm 9 presents the core contribution of the chapter. Quite significantly, the optimization loop has a fixed set of seven variables, *irrespective* of the dimensionality of the data; this makes our technique practical for use on real-world datasets.

Clearly, the choice of the optimizer $suggest()$ is crucial - we discuss this next.

4.2.4 Choice of Optimizer

We begin by listing below the challenges faced by our optimizer:

1. **Black-box objective function**: Our objective function is $accuracy()$, which depends on the interpretable model produced by $train_{\mathcal{I},g}()$ in Algorithm 9. Since we want our technique to be model agnostic, nothing is assumed about the form of $train_{\mathcal{I},g}()$. This effectively makes our objective a black-box function.

2. **Noisy objective function**: The interpretable model is trained on a *sample* based on the current parameters Φ_t. This implies two models constructed for the same Φ_t may not be identical. There might be other sources of noise intrinsic to the learning algorithm too, e.g., local search used for training.

3. **Expensive objective function**: Every evaluation of the objective function requires an interpretable model to be trained, which is expensive. We want our optimizer to be conservative in its calls to the objective function.

We use *Bayesian Optimization (BO)* to implement $suggest()$. This is probably not surprising considering that we used BO for the density tree based approach where we had similar challenges. We don't review the choice of BO in detail here, and instead refer the reader to Section 3.2.3. BO is reviewed in Section 2.1.

We briefly justify its compatibility to our problem. Since BOs construct their own model of the response surface and optimize the corresponding *surrogate* objective, they can optimize black-box functions (Requirement 1). They explicitly model uncertainty[3] in observations, leading to robustness to noise (Requirement 2). Finally, BOs balance *exploitation and exploration* to make well-informed decisions about the best point to next evaluate - making it conservative in its calls to the objective function. We use the *Tree Structured Parzen Estimator (TPE)* algorithm (Bergstra *et al.*, 2011) because of its low runtime complexity and popularity within the hyperparameter optimization community. We use the implementation provided in the *Hyperopt* library (Bergstra *et al.*, 2013).

[3]The connotation of this term here is different from that in Section 4.2.1. Here, it denotes variance in the response surface model. In Section 4.2.1, it quantified uncertainty in the predicted label of an instance.

We again make a note of the fact that having a fixed parameter space for optimization makes our technique a **framework**: any optimizer that satisfies the above criteria may be used. For example, any of the BO algorithms from the *black-box optimization challenge*, NeurIPS2020 (Turner *et al.*, 2021), may be used to implement $suggest()$ in Algorithm 9. *This enables us to make Algorithm 9 faster and better as newer optimizers become available*.

4.2.5 Smoothing the Optimization Landscape

A final but key consideration in our optimization is to make it easier to discover the global maximum: Φ^* in Algorithm 9. Since BOs model the response surface of the actual objective function using a finite number of evaluations (s_t in Algorithm 9), a certain degree of smoothness is assumed (Shahriari *et al.*, 2016; Brochu *et al.*, 2010).

Here, the optimization variables Φ_t influence the sampling in Algorithm 8, which directly affects the score s_t that the BO consumes. Empirically, we have observed that the distribution of uncertainty scores produced by an oracle do not always form a smooth distribution. Consequently, neighboring values of Φ may pick drastically different samples leading to large differences in s_t.

To address this, we "flatten" the distribution[4] within $[0, 1]$. Our transformation is simple: we divide the interval $[0, 1]$ into B bins, and map approximately $|D_{train}|/B$ uncertainty scores to each bin, while maintaining order between the original and mapped scores. Within a bin, the mapped scores are linearly spread across its range. This distributes the mapped scores approximately uniformly in the range $[0, 1]$. The algorithm is detailed in Section A.7.

The alternative to flattening is to identify a suitable parameter for the BO algorithm, e.g., a suitable *kernel* for Gaussian Process based BO. However, this introduces additional hyperparameters; hence we prefer flattening.

Figure 4.7 visualizes the process of flattening. The original and modified uncer-

[4]Distribution transformations have a long history in statistics, e.g., *power transforms* like the *Box-Cox* (Box and Cox, 1964) and *Yeo-Johnson* (Yeo and Johnson, 2000) transforms. Within ML, *Batch Normalization* (Ioffe and Szegedy, 2015) is a popular example of a distribution transformation applied to a loss landscape (Santurkar *et al.*, 2018).

tainty distributions for the datasets `Sensorless` and `covtype.binary` are shown in Figure 4.7(a) and 4.7(b) respectively.

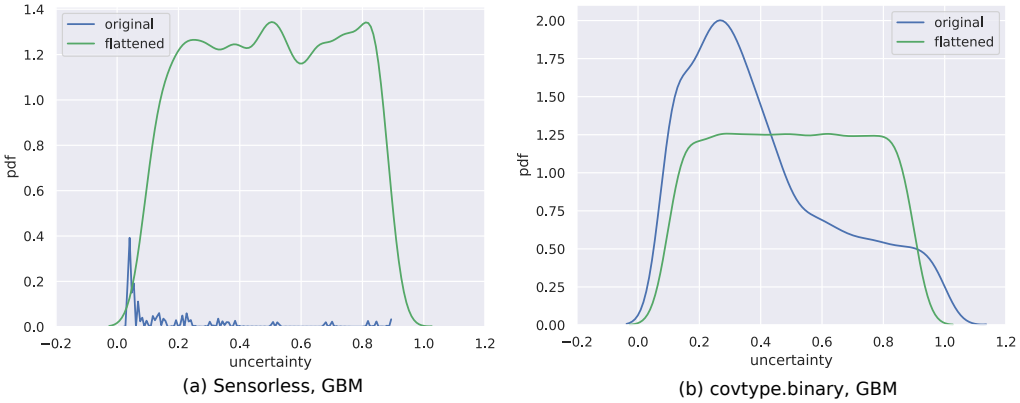


Figure 4.7: Example of curve-flattening, for datasets (a) `Sensorless` and (b) `covtype.binary`. The uncertainty scores shown are obtained using the GBM oracle.

While `Sensorless` appears to have a non-smooth distribution, and flattening here might help, this seems redundant for `covtype.binary`. *However, since this step is computationally inexpensive, we perform this for all our experiments, saving us the effort of assessing its need.* The effect of flattening in our experiments is discussed in Section 4.4.

Our transformation is invertible, which is useful in analyzing the observations from our experiments. Note however, it is not differentiable because of the discontinuities at the bin-boundaries; we also don't require this property.

The transformation affects line 7 in Algorithm 8. Instead of sampling based on the actual oracle uncertainty scores:

$$p_j \leftarrow Beta(u_{M_O}(x_j); A_i, B_i) \tag{4.8}$$

we sample based on the transformed uncertainty scores, $u'_{M_O}(x_j)$:

$$p_j \leftarrow Beta(u'_{M_O}(x_j); A_i, B_i) \tag{4.9}$$

The use of the transformation is optional, since Algorithm 9 does not critically depend upon it, but makes it robust (discussed in Section 4.4).

□

This concludes our discussion of algorithmic details (see Section A.1 for additional implementation details). In summary, we require seven parameters $\Phi = \{\Psi, N_s, p_o\}$, where $\Psi = \{\alpha, a, b, a', b'\}$. Hyperparameters are discussed in Section 4.3.1.5. Our experimental validation of the technique is discussed next.

4.3 Experiments

We now look at extensive evaluation of our technique. Our experiments maybe categorized into the following types:

1. **Validation**, Section 4.3.1: this set of experiments exhibit statistically significant improvements across multiple datasets, using different models and oracles (Section 4.3.1.6). Various properties of the learned distributions are analyzed (Section 4.3.1.8). The relationship between model capacity and the efficacy of our technique is also discussed (Section 4.3.1.9).

2. **Comparisons**, Section 4.3.2: here we compare the improvements produced by our technique with (a) a supervised version of uncertainty sampling and (b) using density trees.

3. **Additional applications**, Section 4.3.3: fundamentally, our technique learns a sampling distribution that leads to effective training. This can be used as a tool for the following interesting applications - (a) different feature representations may be used across the interpretable model and the oracle, e.g., a DT as the interpretable model with n-grams as input, and a Gated Recurrent Unit the oracle, that operates on a sequence of tokens, (b) a minimal sample for effective learning maybe identified using our technique and (c) a multivariate notion of model size may be used.

The section on validation experiments is the most comprehensive, establishing various aspects of our technique.

4.3.1 Validation

Since the goal of validation is the same as in the case of density trees, much of the experimental setup is similar. The two setups overlap in the following aspects:

1. Datasets used (Section 3.3.1).
2. Interpretable models used (Section 3.3.2): *Linear Probability Model (LPM)* and *Decision Trees (DT)*. The notions of the model sizes are identical as well.
3. Validation metric and tests of statistical significance (Section 3.3.3).
4. Analyzing the effect of model capacity (Section 3.3.7).

The key difference is the presence of oracle models; *Gradient Boosted Models (GBM)* and *Random Forests (RF)* are used, and metrics are reported for all combinations of the interpretable and oracle models, i.e., $\{LPM, DT\} \times \{GBM, RF\}$. In the interest of brevity the descriptions in this section are kept concise - please refer to the respective sections within Section 3.3 for details.

4.3.1.1 Data

Table 4.2 lists the *13* datasets used for evaluation. 10000 instances from each dataset are used. We use a $train : val : test$ split ratio of $60 : 20 : 20$ to create D_{train}, D_{val} and D_{test} in all our experiments (line 1, Algorithm 9). The data splits are stratified wrt class labels.

The skew of the label distribution is quantified using "Label Entropy", which is computed for a dataset with N instances and C labels in the the following manner:

$$\text{Label Entropy} = \sum_{j \in \{1,2,\ldots,C\}} -p_j \log_C p_j \tag{4.10}$$

$$\text{Here, } p_j = \frac{|\{x_i | y_i = j\}|}{N}$$

This value lies in the range $[0, 1]$, where high values indicate relatively balanced labels.

Table 4.2: We use the following datasets available on the LIBSVM website (Chang and Lin, 2011). Their original source is mentioned in the "Description" column. 10000 instances from each dataset are used. A $train : val : test$ split ratio of $60 : 20 : 20$ is used for D_{train}, D_{val} and D_{test} in Algorithm 9. The splits are stratified wrt labels.

S.No.	Dataset	Dimensions	# Classes	Label Entropy	Description
1	cod-rna	8	2	0.92	Predict presence of non-coding RNA common to a pair of RNA sequences, based on individual sequence properties and their similarity (Uzilov *et al.*, 2006).
2	ijcnn1	22	2	0.46	Time series data produced by an internal combustion engine is used to predict normal engine firings vs misfirings (Prokhorov, 2001). Transformations as in (Chang and Lin, 2001).
3	higgs	28	2	1.00	Predict if a particle collision produces Higgs bosons or not, based on collision properties (Baldi *et al.*, 2014).
4	covtype.binary	54	2	1.00	Modification of the *covtype* dataset (see row 12), where classes are divided into two groups (Collobert *et al.*, 2002).
5	phishing	68	2	0.99	Various website features are used to predict if the website is a *phishing* website Mohammad *et al.* (2012). Transformations used as in Juan *et al.* (2016)
6	a1a	123	2	0.80	Predict whether a person makes over 50K a year, based on census data variables (Dua and Graff, 2017). Transformations as in (Platt, 1998).
7	pendigits	16	10	1.00	Classify handwritten digit samples into the digits 0-9. (Alimoglu and Alpaydin, 1996; Dua and Graff, 2017).
8	letter	16	26	1.00	Images of the capital letters A-Z were produced by random distortion of these characters from 20 fonts. The task is to classify these character images as one of the original letters (Michie *et al.*, 1995). Transformations as in (Hsu and Lin, 2002).
9	Sensorless	48	11	1.00	Based on phase current measurements of an electric motor, predict different error conditions (Paschke *et al.*, 2013). We use the transformations from (Wang *et al.*, 2018*b*).
10	senseit_aco	50	3	0.95	Predict vehicle type using acoustic data gathered by a sensor network (Duarte and Hu, 2004).
11	senseit_sei	50	3	0.94	Predict vehicle type using seismic data gathered by a sensor network (Duarte and Hu, 2004).
12	covtype	54	7	0.62	Predict forest cover type from cartographic variables (Blackard, 1998; Dua and Graff, 2017).
13	connect-4	126	3	0.77	Predict if the first player wins, loses or draws, based on board positions of the board game *Connect Four* (Dua and Graff, 2017).

4.3.1.2 Models

For interpretable models $\mathcal{I}$, we consider the following model families (see Section 3.3.2 for additional details):

1. *Linear Probability Model (LPM)* (Mood, 2010): This is a linear classifier, where the notion of model size used is the number of terms in the model, i.e., features from the original data, with non-zero coefficients. The *Least Angle Regression* (Efron *et al.*, 2004) algorithm is used to construct the model up to a specified size.

   Since LPMs inherently handle only binary class data, for a multiclass problem, we construct a *one-vs-rest* model. The given size is enforced for *each* binary classifier.

   Sizes: For a dataset with dimensionality d, we construct models of sizes: $\{1, 2, ..., min(d, 15)\}$. We end up with sizes less than 15 only for the dataset *cod-rna*, which has $d = 8$. All other datasets have $d > 15$ (see Table 4.2).

2. *Decision Trees (DT)*: We use the implementation of CART in the *scikit-learn* library. Our notion of size here is the depth of the tree.

   Sizes: For a dataset, we first learn a tree T_{opt} (no size constraints are imposed) using standard $5-$fold cross-validation. We refer to this as the optimal tree, and its depth is denoted by $depth(T_{opt})$. We then experiment up to a model size of $min(depth(T_{opt}), 15)$. This is controlled by setting the values of CART's max_depth parameter to: $\{1, 2, ..., min(depth(T_{opt}), 15)\}$.

   We don't exceed $depth(T_{opt})$ since at this depth the model is saturated in its learning from the data, and we don't expect changes in the input distribution to influence accuracy.

   We control the depth using the parameter max_depth - the *maximum* depth to which a DT is grown - since decision tree libraries do not allow specification of an exact tree depth. However, we report improvements at *actual* depths. Being only able to specify max_depth also means that for different values it might lead

to trees with the same actual depth. For example, a DT with $depth = 5$ might be produced by setting $max\_depth = 5$ or $max\_depth = 6$.

Model selection within an optimizer iteration is performed by averaging over three runs of holdout cross validation, with the ratio of the training to holdout data size being $80 : 20$. The splits are randomly determined and are stratified wrt class labels.

Baselines: Similar to case for density trees (Section 3.3.2), baseline models are trained using standard 3-fold cross-validation, stratified by class labels. In the case of DTs, the space of the parameter $min\_impurity\_decrease = \{0, 0.25, 0.5, 0.75, 1\}$ is also explored. This parameter enforces node splits to be executed only if the decrease in impurity is at least as large as the parameter's value[5].

The baseline model has access to more data for training than models trained within the optimizer: the former combines D_{train} and D_{val} for its 3-fold CV, while the latter has access to only D_{train}. With our splits of $D_{train} : D_{val} : D_{test} :: 60 : 20 : 20$, the baseline sees $80/60$ or $1.33x$ more data. Often the difference is greater since at a particular iteration, the parameter N_s might be set to a value much lower than $|D_{train}|$. The reduced training data size is why the within-optimizer model selection doesn't use a 3-fold CV, since that would enforce a training split of 67%, as opposed to 80% that is available to it now.

4.3.1.3 Oracles

We want our oracle models $\mathcal{O}$ to be fairly accurate, so that the derived uncertainty information is reliable. Hence we pick the following model families:

1. *Gradient Boosted Models (GBM)*: We used a gradient boosting model with DTs as our base classifiers. The *LightGBM* library Ke *et al.* (2017) is used in our experiments. Effective parameters were determined using a validation set. **NOTE:** This is *not* D_{val} from Algorithm 9, since that would constitute *data leakage*. A sample, stratified by labels, from within D_{train} was held out for learning good GBM parameters.

[5] See documentation here: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

2. *Random Forests (RF)*: We used the implementation available in *scikit-learn*. Parameters were learned using 5-fold cross-validation over D_{train}.

The above oracles were calibrated (Platt, 1999) for reliable probability estimates.

4.3.1.4 Metrics

We measure two quantities - improvements in model accuracy and their statistical significance. These are identical to the ones in Section 3.3.3 for ease of comparison:

1. To measure $accuracy()$ as in Equation 4.2 or Algorithm 9, our metric of choice is the $F1$ (macro) score, evaluated on D_{test}. We use this since it accounts for class imbalance, e.g., it doesn't allow good results for a majority class to eclipse poor results for a minority class.

   To measure the *improvements* obtained using our technique, we record the percentage *relative improvement* in the $F1$ score compared to the *baseline* of training the model on the original distribution:

   $$\delta F1 = \frac{100 \times (F1_{new} - F1_{baseline})}{F1_{baseline}} \tag{4.11}$$

   Since the original distribution is part of the optimization search space, i.e., when $p_o = 1$, the lowest improvement we report is 0%, i.e., $\delta F1 \in [0, \infty)$.

   All reported values of $\delta F1$ represent averaging over **five** runs of Algorithm 9, where we average the baseline and new scores *first*, and then calculate the improvement. If the runs are indexed by i, $F1_{new}$ and $F1_{baseline}$ are replaced by $\overline{F1}_{new} = \sum_{i=1}^{3} F1_{new,i}/3$ and $\overline{F1}_{baseline} = \sum_{i=1}^{3} F1_{baseline,i}/3$ respectively, in Equation 4.11.

   Scores are averaged first to avoid large variations in per-run $\delta F1$, which can happen due to $F1_{baseline}$ potentially being a small value, especially at smaller model sizes.

2. The *Wilcoxon signed-rank test* is used to measure statistical significance, where the paired set of samples are $F1_{baseline}$ and $F1_{new}$ scores (from Equation 4.11)

for a dataset. The *p-value* is reported for all combinations of interpretable models and oracles, i.e., $\{LPM, DT\} \times \{GBM, RF\}$. This test is separately performed for different model sizes.

4.3.1.5 Parameter Settings

The optimizer we use, TPE, requires *box constraints*. Here we specify our search space for the optimization variables, Φ in Algorithm 9:

1. p_o: We want to allow the algorithm to pick an arbitrary fraction of samples from the original data; we set $p_o \in [0, 1]$.

2. N_s: We set $N_s \in [400, 10000]$. The lower bound ensures we have statistically significant results. The upper bound is set to a reasonably large value.

3. $\{a, b, a', b'\}$: Each of these parameters are allowed a range $[0.1, 10]$ to allow for a wide range of shapes for the component $Beta$ distributions.

4. $scale$: We fix $scale = 10000$ for our experiments, to allow for A_i and B_i to model skewed distributions where shape parameter large values might be required. For small values, the algorithm adapts by learning the appropriate $\{a, b, a', b'\}$.

5. α: For a DP, $\alpha \in \mathbb{R}_{>0}$. We use a lower bound of 0.1.

   To determine the upper bound, we rely on the following empirical relationship (Ohlssen *et al.*, 2007) between the number of components k and α:

   $$E[k|\alpha] \approx 5\alpha + 2 \tag{4.12}$$

   We empirically estimated a fairly inclusive upper bound on the number of components to be 500, which provides us the α upper bound of 99.6. Thus, we use $\alpha \in [0.1, 99.6]$.

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

We use a flattening transformation (discussed in Section 4.2.5) on the original uncertainty distributions, with a fixed number of 20 bins. *However, all visualizations of distributions in the following sections were prepared after performing an inverse transformation*; hence, in studying them, it might be convenient to assume that no transformation was applied.

Hyperparameters: In theory, the box constraints and the iteration budget required by the optimizer constitute our hyperparameters, which may be tuned for a specific task. However, as detailed above, we don't need to estimate a range for p_o and reasonable defaults may be applied to N_s, $\{a, b, a', b'\}$, $scale$ and α. This is practically valuable since we are left with T as our only hyperparameter. This was set to $T = 1000$ for LPMs and $T = 3000$ for DTs based on limited search. Since the LPMs we use construct multiple *one-vs-rest* classifiers, higher iteration budgets are computationally expensive to use.

□

This completes our discussion of the experimental setup; we present our observations next.

4.3.1.6 Improvements in Accuracy

Figure 4.8 shows the improvements for different combinations of interpretable and oracle models, $\{LPM, DT\} \times \{GBM, RF\}$. The model size is on the x-axis, and is normalized to be in $[0, 1]$, so that performance across datasets may be conveniently compared in the same plot.

For LPMs, the model sizes for a dataset, i.e., number of non-zero terms, are multiplied by $1/min(d, 15)$, where d is the dimensionality of the data. For DTs, the model sizes are multiplied by $1/min(depth(T_{opt}), 15)$. All $\delta F1$ values are *averaged over five runs*, in the manner described in Section 4.3.1.4.

Table 4.3 enumerates the observations corresponding to the plots in Figure 4.8. The column *model_ora* represents the model and oracle combination used. For example, *dt_gbm* implies DT was used as the model and GBM as an oracle.

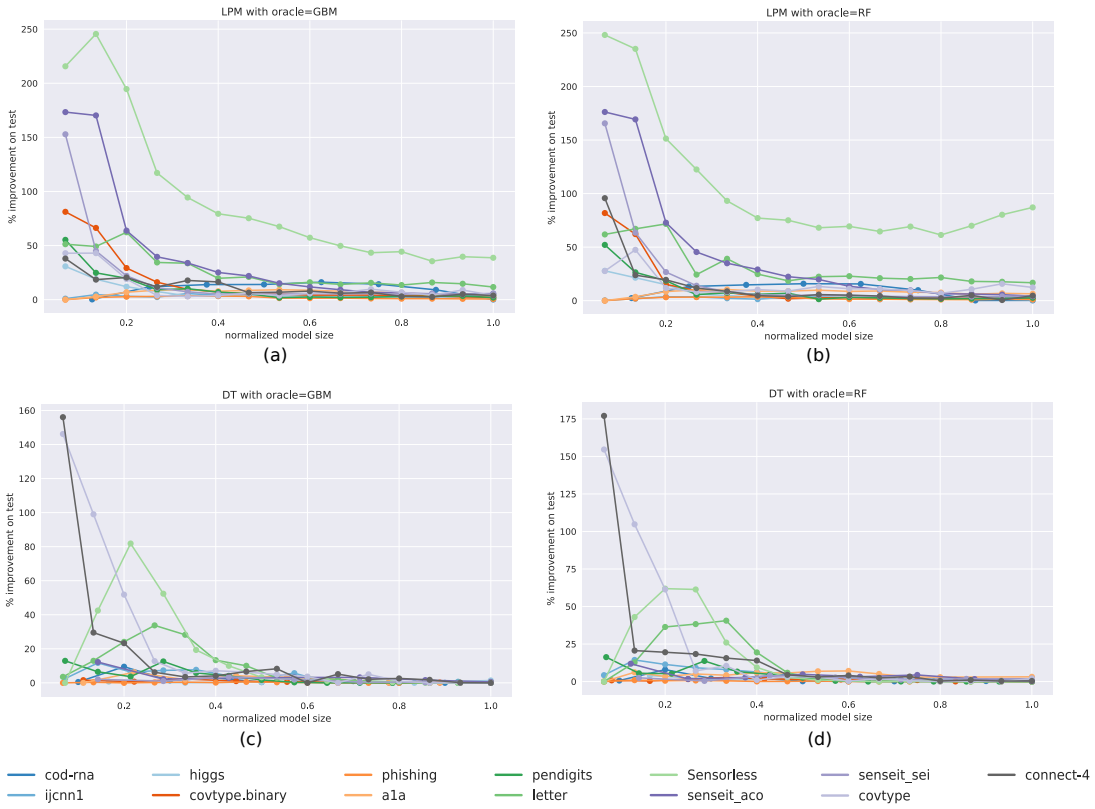


Figure 4.8: For different combinations of models and oracles: $\{LPM, DT\} \times \{GBM, RF\}$, these plots show improvements, $\delta F1$, seen for different model sizes and data. Table 4.3 shows the corresponding improvement scores.

We observe that the oracle based approach indeed works on a variety of datasets, across different combinations of interpretable and oracle models. In some cases, such as the dataset `Sensorless`, for the LPM and RF combination, improvements are as high as $\delta F1 = 248.12\%$. The general trend seems to be that $\delta F1$ decreases as model sizes increase, with eventually $\delta F1 \approx 0$. This decrease seems to be faster for DTs, which makes intuitive sense given that a unit increase in size for a DT adds more representational power (a layer of nodes) than for an LPM (another term), making it harder to beat the baseline performance of DTs.

This decrease empirically verifies the property expressed by Equations 4.3 and 4.4.

We note that $\delta F1$ does not strictly monotonically decrease for all datasets, possibly due to the optimization terminating at a local maxima, e.g., in Table 4.3 see the entry for `letter`, *lpm_rf*, $size = 2$ ($improvement = 67.06\%$) and $size = 3$ ($improvement = 71.08\%$). But it largely appears to follow the general trend of decrease even in these cases.

Table 4.3: This table shows the average improvements, $\delta F1$, over **five** runs for different combinations of models and oracles: $\{LPM, DT\} \times \{GBM, RF\}$. The best improvement for a model size and oracle is indicated in bold.

dataset	model_ora	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cod-rna	lpm_gbm	1.11	12.65	**14.80**	15.36	**16.18**	11.60	**8.41**	**2.61**	-	-	-	-	-	-	-
	lpm_rf	**2.95**	**13.37**	14.31	**15.49**	15.78	**12.53**	5.89	0.18	-	-	-	-	-	-	-
	dt_gbm	**1.09**	**9.80**	1.09	1.84	**0.44**	**0.75**	**1.08**	0.42	0.00	**0.28**	-	-	-	-	-
	dt_rf	0.57	2.78	**1.81**	**2.58**	0.13	0.46	0.40	**0.70**	**0.31**	0.00	-	-	-	-	-
ijcnn1	lpm_gbm	**0.59**	2.84	**3.71**	2.39	**4.97**	**4.61**	**3.85**	**3.97**	**3.46**	2.34	**3.00**	2.74	2.85	3.46	2.62
	lpm_rf	0.50	**3.26**	3.22	**3.88**	3.43	2.17	3.50	3.21	3.43	**4.05**	2.84	**4.37**	**3.99**	**3.48**	**4.23**
	dt_gbm	2.10	11.75	7.07	8.95	8.47	3.89	2.67	**2.36**	0.60	0.39	0.02	0.29	0.67	0.74	0.39
	dt_rf	**4.24**	**14.43**	**10.77**	**11.00**	**10.38**	**5.46**	**4.74**	2.32	**2.76**	**1.25**	**1.72**	**1.31**	**1.70**	**1.24**	**1.70**
higgs	lpm_gbm	**29.54**	17.59	10.79	6.81	2.88	2.69	**3.12**	**2.91**	**2.86**	**2.39**	**2.59**	1.59	2.17	**1.96**	0.76
	lpm_rf	23.18	**18.59**	**15.03**	**7.96**	**4.33**	**3.61**	2.29	2.55	1.78	1.34	2.02	**2.22**	**2.79**	1.63	**1.51**
	dt_gbm	1.62	0.59	1.22	0.75	0.01	**1.41**	-	-	-	-	-	-	-	-	-
	dt_rf	**3.98**	**0.85**	**1.90**	**1.63**	**1.69**	0.79	-	-	-	-	-	-	-	-	-
covtype.binary	lpm_gbm	86.19	**66.39**	**27.19**	**13.19**	**7.26**	**5.47**	**4.01**	**3.95**	**3.26**	**3.36**	**3.20**	**3.06**	**2.49**	**2.39**	1.46
	lpm_rf	**87.10**	63.38	12.76	8.33	6.25	3.75	2.49	2.41	2.76	2.77	2.41	2.67	2.42	2.34	**2.19**
	dt_gbm	**1.24**	**0.62**	**2.09**	0.99	0.52	1.02	0.15	0.50	0.06	-	-	-	-	-	-
	dt_rf	0.68	0.40	1.61	**2.01**	**1.70**	**1.45**	**1.04**	**1.30**	**0.97**	0.50	-	0.00	-	-	-
phishing	lpm_gbm	**0.00**	1.88	2.88	3.05	3.22	3.37	2.86	1.61	1.37	1.44	**1.21**	1.03	1.07	0.84	0.86
	lpm_rf	0.00	**2.14**	**3.29**	**3.22**	**3.59**	**3.79**	**3.29**	**1.85**	**1.46**	**1.46**	1.18	**1.18**	**1.22**	**1.27**	**1.08**
	dt_gbm	**0.00**	0.57	0.33	0.13	**0.44**	**0.11**	**0.48**	0.33	**0.13**	0.00	0.01	0.00	0.00	0.00	**0.05**
	dt_rf	0.00	**0.72**	**0.61**	**0.44**	0.44	0.08	0.12	**0.42**	0.13	**0.07**	**0.10**	**0.06**	**0.04**	**0.01**	0.00
a1a	lpm_gbm	0.00	2.55	7.58	9.11	9.03	7.87	8.72	8.86	8.56	7.90	7.38	7.14	5.78	6.15	5.56
	lpm_rf	**0.02**	**4.17**	**8.81**	**10.24**	**10.46**	**9.11**	**9.18**	**9.52**	**8.97**	**9.70**	**8.98**	**8.50**	**7.34**	**7.67**	**6.77**
	dt_gbm	**0.01**	5.67	2.10	4.33	3.53	2.91	0.40	0.64	0.27	-	-	-	-	-	-
	dt_rf	0.00	**6.62**	**3.44**	**5.14**	**4.36**	**5.70**	**4.99**	**5.14**	**5.92**	4.43	3.02	2.94	-	-	3.16
pendigits	lpm_gbm	**52.66**	**22.62**	16.88	**8.34**	**9.73**	**6.90**	5.12	**2.03**	2.44	2.21	2.31	2.13	**3.26**	**3.03**	2.39
	lpm_rf	50.10	21.68	**20.70**	7.77	8.16	6.15	**7.04**	1.61	**3.38**	**2.97**	**2.45**	**2.48**	2.75	2.65	**2.97**
	dt_gbm	13.70	**6.98**	**4.92**	12.72	6.21	**4.68**	**2.40**	**0.87**	**0.48**	**0.04**	**0.00**	**0.19**	**0.00**	**0.00**	**0.01**
	dt_rf	**18.93**	4.26	4.36	**13.70**	**6.67**	4.58	2.38	0.32	0.00	0.02	0.00	0.00	0.00	0.00	0.00
letter	lpm_gbm	59.54	44.83	58.49	**29.47**	35.49	17.43	**21.06**	16.47	17.48	15.02	16.88	15.98	**18.10**	17.24	15.30
	lpm_rf	**62.61**	**64.36**	**67.06**	24.71	**36.95**	**24.14**	19.88	**21.70**	**20.63**	**19.64**	**18.42**	**20.65**	17.93	**17.71**	**17.67**
	dt_gbm	**2.10**	10.91	25.55	33.11	30.04	15.54	**11.32**	**4.47**	**3.59**	**3.06**	**1.85**	**1.26**	**1.08**	**0.62**	**0.35**
	dt_rf	0.12	**12.53**	**34.24**	**37.39**	**35.43**	**16.14**	5.82	1.22	2.30	0.71	0.49	0.16	0.00	0.00	0.00
Sensorless	lpm_gbm	221.53	**259.30**	**195.99**	**121.89**	**94.41**	**83.82**	**75.07**	**67.90**	59.42	51.98	56.11	55.86	58.70	65.27	60.88
	lpm_rf	**238.94**	238.83	143.82	103.65	85.69	71.32	74.25	65.22	**66.78**	**61.20**	**59.88**	**56.84**	**60.67**	**69.39**	**72.12**
	dt_gbm	**0.04**	46.99	**66.38**	**54.79**	**22.65**	**10.67**	**2.28**	**1.69**	**0.84**	**0.85**	**0.41**	**0.14**	**0.16**	0.16	0.07
	dt_rf	0.01	**52.54**	57.27	44.33	16.26	6.49	2.23	0.69	0.27	0.06	0.29	0.08	0.00	**0.23**	**0.16**
senseit_aco	lpm_gbm	173.63	**175.44**	68.21	44.09	35.41	24.18	20.83	15.80	11.39	8.09	5.83	5.21	4.77	4.21	3.95
	lpm_rf	**177.67**	175.20	**91.96**	**47.67**	**37.03**	**30.28**	**25.19**	**22.54**	**14.74**	**10.46**	**9.28**	**6.31**	**5.91**	**5.46**	**4.23**
	dt_gbm	14.92	1.83	**4.89**	3.35	3.03	0.97	0.57	-	-	-	-	-	-	-	-
	dt_rf	**15.41**	**2.22**	3.88	**4.82**	**3.81**	**3.54**	**1.66**	0.25	-	-	-	-	-	-	-
senseit_sei	lpm_gbm	160.59	57.00	23.42	10.47	6.70	4.49	4.49	4.12	4.55	4.14	**4.40**	**4.91**	3.83	3.97	**4.29**
	lpm_rf	**165.98**	**67.35**	**29.13**	**14.35**	**8.80**	**5.53**	**4.72**	**4.90**	**5.11**	**4.47**	4.39	3.58	**4.16**	**4.26**	4.15
	dt_gbm	2.42	0.75	**3.26**	1.23	0.39	0.42	0.27	0.27	-	-	-	-	-	-	-
	dt_rf	**2.54**	**1.28**	3.23	**2.26**	**1.18**	**1.49**	**1.91**	-	-	-	-	-	-	-	-
covtype	lpm_gbm	**36.69**	46.55	**14.35**	**6.51**	4.64	6.49	6.73	7.72	8.42	8.39	**11.47**	**8.39**	4.73	9.37	6.17
	lpm_rf	32.52	**47.56**	12.23	5.46	**6.46**	**9.65**	**10.31**	**12.33**	**12.62**	**10.56**	9.86	7.85	**11.52**	**17.49**	**16.24**
	dt_gbm	146.22	**107.32**	43.31	**15.53**	4.30	**5.64**	3.83	3.22	**3.15**	0.93	**2.68**	**1.56**	0.63	0.34	0.00
	dt_rf	**152.12**	102.28	**53.72**	7.86	**9.41**	4.97	**4.67**	**4.42**	1.68	**2.86**	1.08	0.00	**0.85**	**0.90**	**2.79**
connect-4	lpm_gbm	27.54	**25.89**	14.36	**9.34**	**12.56**	**10.29**	5.47	4.48	5.45	3.79	**4.72**	2.99	1.92	**3.31**	**3.46**
	lpm_rf	**59.19**	14.24	**17.20**	9.00	9.22	6.98	**7.87**	**7.65**	**5.58**	**5.32**	2.40	**3.04**	**4.94**	1.94	3.23
	dt_gbm	136.35	**27.02**	**23.83**	7.80	4.13	3.46	**5.09**	**5.31**	0.39	**4.73**	2.02	**2.31**	**1.90**	**0.62**	**1.14**
	dt_rf	**147.32**	20.08	19.51	**18.57**	**13.17**	**12.70**	4.82	3.23	**3.90**	3.41	**3.72**	0.53	0.92	0.19	0.80

Before we conclude this section, we present an additional way to visualize improvements: create a correspondence of model sizes, without and with our technique, for the same accuracy. See Figure 4.9 as an example. The point $(12, 2)$ for `senseit_aco` implies that the accuracy of a LPM with 2 non-zero terms produced by our technique equals, or is greater than, the accuracy of a baseline LPM with 12 non-zero terms. The model size on the y-axis is the median of five runs. We refer to such a plot as the *com-*

paction profile for a model-oracle combination. See Section A.9 for more compaction profiles.

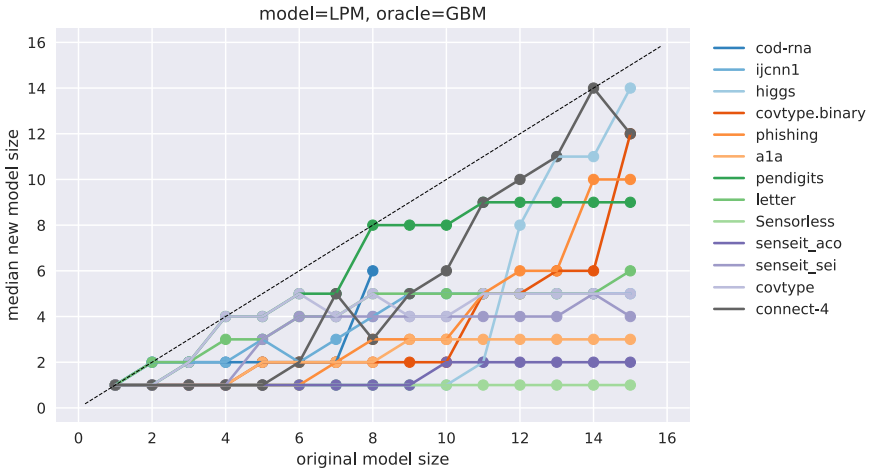


Figure 4.9: The compaction profile of LPM models using GBM as an oracle. A point (x, y) denotes the minimum size y of a model obtained using our technique that is *at least* as accurate as the baseline model of size x.

4.3.1.7 Statistical Significance

The statistical significance of the improvements presented in Section 4.3.1.6 is measured by the *one-sided* version of the paired *Wilcoxon signed-rank test*, where the pairs of scores $F1_{baseline}$ and $F1_{new}$ are used. Figure 4.10 shows the results. This test is similar to the one used in the previous chapter (Section 3.3.6) except that we conduct the test for every pair of model and oracle. The setup is as follows:

1. We compare the classifiers learned by our technique with the baseline, for a given range of model sizes. Separate tests are performed for different model size ranges since size strongly influences $\delta F1$.

2. Normalized model sizes are used for ease of comparison with Figure 4.8. Binning of model sizes is done using *Sturges rule* (Sturges, 1926).

3. The *one-sided* version of the *paired* test is performed for each bin, where pairs of scores $F1_{baseline}$ and $F1_{new}$ for a dataset, for models with sizes assigned to the bin, are compared. In cases were where multiple model sizes for a dataset fall within the same bin, $F1_{baseline}$ and $F1_{new}$ are first averaged and then compared.

4. The following hypotheses are tested:

   - $\mathbf{H_0}$, null hypothesis: accuracies of models trained using the oracle are not better.
   - $\mathbf{H_1}$, alternate hypothesis: accuracies of models trained using the oracle are better.

   p-values are shown for each bin. Small *p-values* favor $\mathbf{H_1}$, i.e., our algorithm.

5. Scores of $\delta F1 = 0$ are split equally between positive and negative ranks[6].

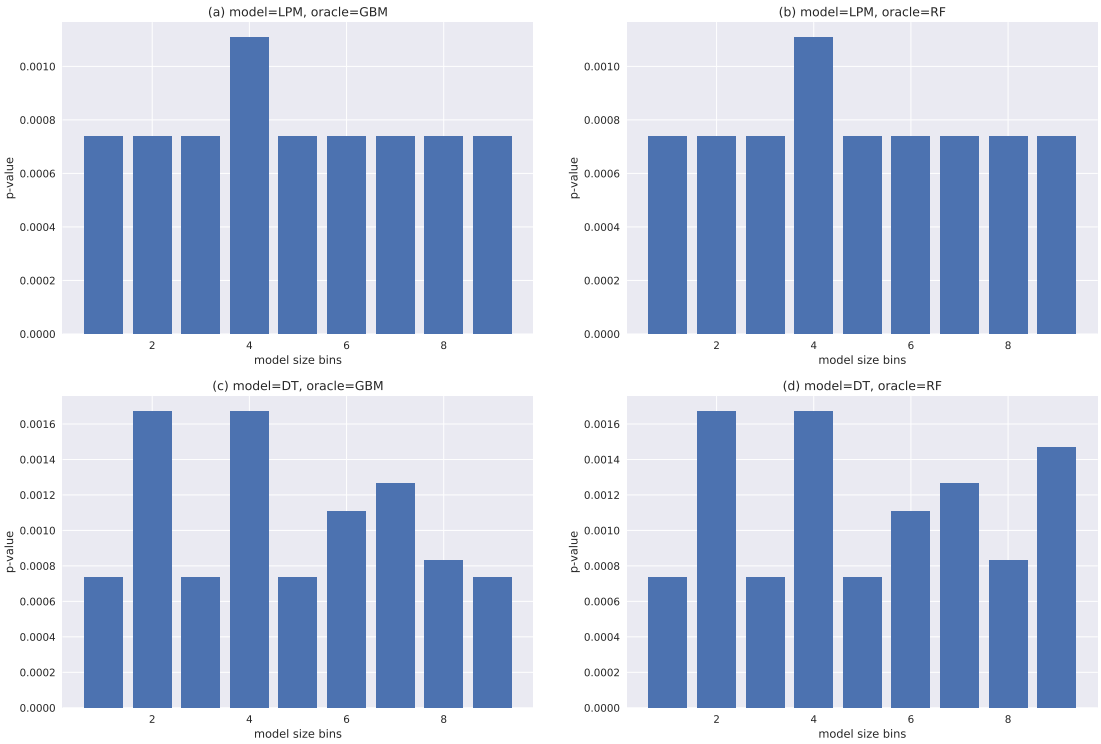


Figure 4.10: These plots show the *p-values* for the Wilcoxon signed-rank test, with the null hypothesis H_0: using the oracle does *not* produce better F1 test scores. The bin boundaries are selected using the *Sturges* rule (Sturges, 1926). Low *p-values* favor our algorithm.

We observe that the improvements from using an oracle are indeed significant for various model size and model-oracle combinations, when measured across multiple datasets.

[6]The `zplit` option in https://numpy.org/doc/stable/reference/generated/numpy.histogram_bin_edges.html is used.

4.3.1.8 Learned Distributions

It is also instructive to analyse the distributions we have learned: this includes both the parameter p_o and the parameters for the IBMM $\Psi = \{\alpha, a, b, a', b'\}$.

Figure 4.11 shows how p_o varies with normalized model size when the interpretable model is DT and the oracle is (a) GBM or (b) RF. This plot ignores the datasets where the largest tree depth explored was less than $depth(T_{opt})$ - so we can compare distributions in size regimes where our technique is effective against when it is not (recall, at sizes close to $depth(T_{opt})$ we expect $\delta F1 \approx 0$). The datasets ignored are[7]: `a1a`, `ijcnn1`, `covtype`, `connect-4`. Here, we clearly see $p_o \to 1$ as model size increases, thus implying the training algorithm tends to use more of the original distribution[8]. This observation is a **key contribution** of this work, since it challenges the conventional wisdom that the training data must be drawn from the same distribution as the test data, for effective learning. This reinforces a similar observation from the previous chapter (Figure 3.12 in Section 3.3.5).

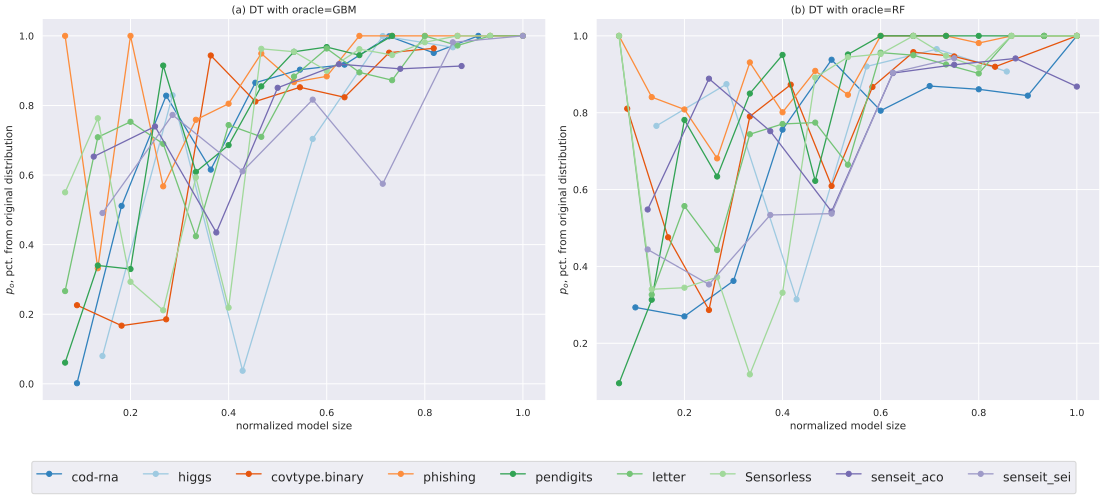


Figure 4.11: These plot shows the effect of increasing model size on p_o, when the interpretable model is DT. These plots strongly indicate that larger model sizes learn better with the original distribution. Some datasets are ignored - see text for explanation.

We now consider the IBMM distributions over the uncertainty values. These are difficult to concisely visualize since one IBMM is learned for *each* model size. Hence,

[7] These datasets are easy to identify in Table 4.3: the ones where the last column(s) is neither ≈ 0 nor "-".

[8] In theory, the parameters Ψ could have been learned such that they mimic the original distribution, but we hypothesize that it is easier for the optimizer to learn the appropriate value of one parameter p_o as opposed to equivalent values of the multiple parameters Ψ. This is why we see the clear pattern in Figure 4.11.

we propose the following plot that aggregates distributions across model sizes for a dataset:

1. We set a value for N; the number of points to sample.
2. For a model size η_i, we sample n_i points from its corresponding IBMM, where $n_i \propto \delta F1_i$, the improvement seen at this size. For example, let's say we have explored two model sizes η_1, η_2, and these have led to improvements of $\delta F1_1 = 10\%$ and $\delta F1_2 = 20\%$, respectively. Then, $n_1 = 0.33N$ and $n_2 = 0.67N$.
3. The various samples of sizes n_i are pooled together and a *Kernel Density Estimator (KDE)* fit on this data is visualized.

The KDE thus obtained is predominantly shaped by the distributions that resulted in high $\delta F1$. For the case of the LPM these are visualized in Figure 4.12 for both oracles.

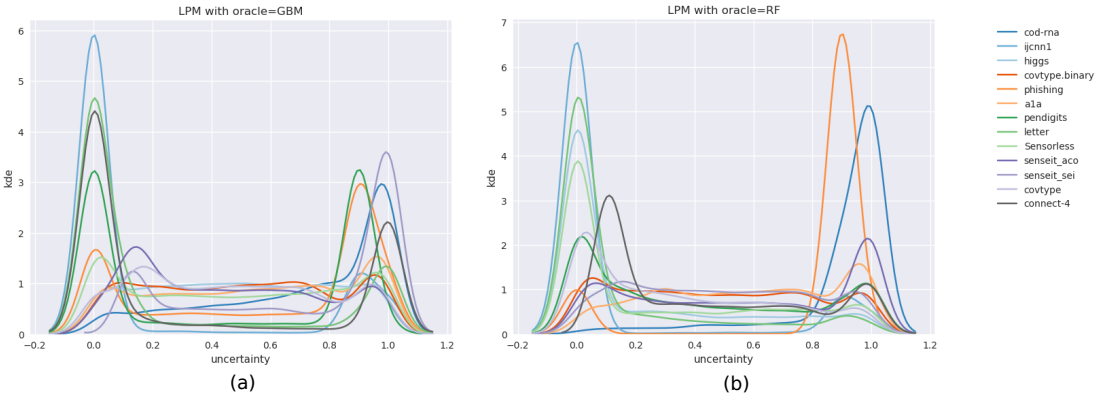


Figure 4.12: The aggregated IBMMs are visualized for LPMs, when the oracle is a (a) GBM or (b) RF. The corresponding plots for DTs are presented in Section A.8.

It is interesting to see that the optimal strategy, in general, turns out to be to sample from *both* regions of low and high uncertainties.

Going a step further we might wonder what the aggregated distribution would look like if adjusted for the number of instances with a given uncertainty value. For example, we might see a peak on the extreme right for a dataset in Figure 4.12 simply because most points receive a high uncertainty score.

We use the following technique to visualize such an *adjusted* aggregate distribution:

1. We again pick N, the number of points to sample. Exactly like in the previous case: we pool together samples from IBMMs for different model sizes, where the

relative sample sizes are decided by the respective $\delta F1$ scores. We fit a KDE to this data, which we refer to as A.

2. We fit another KDE to the uncertainty values produced by the oracle for the training data. Let's call this B.

3. For K uniformly spaced values of uncertainty $u_k \in [0, 1], 1 \leq k \leq K$, we calculate the ratio $p_A(u_k)/p_B(u_k)$, and plot a scaled version of it $c \cdot p_A(u_k)/p_B(u_k)$. The scaling factor c is picked to transform the ratios into probability masses, i.e., $\sum_{k=1}^{K} c \cdot p_A(u_k)/p_B(u_k) = 1$.

   Essentially, we *normalize* the sampling probability $p_A(u_k)$ at u_k, with $p_B(u_k)$, a quantity representing the number of instances with uncertainty u_k.

These plots are shown in Figure 4.13. The corresponding plots for the DT are shown in Figure A.5, Section A.8.

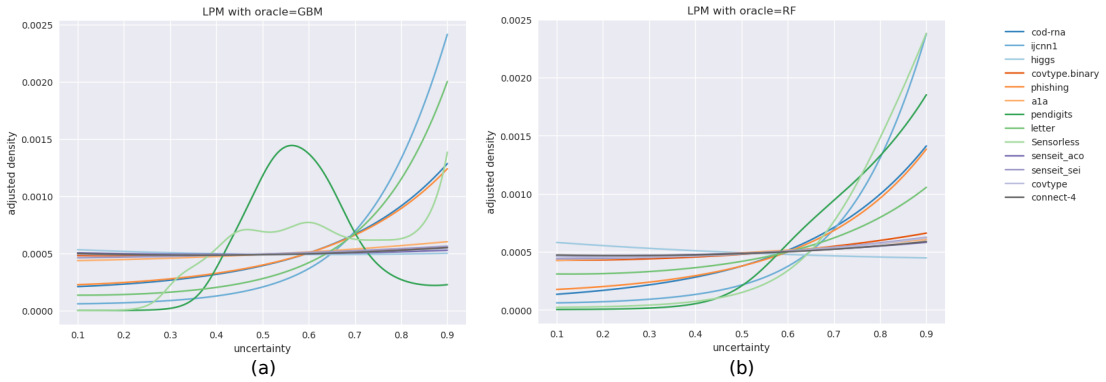
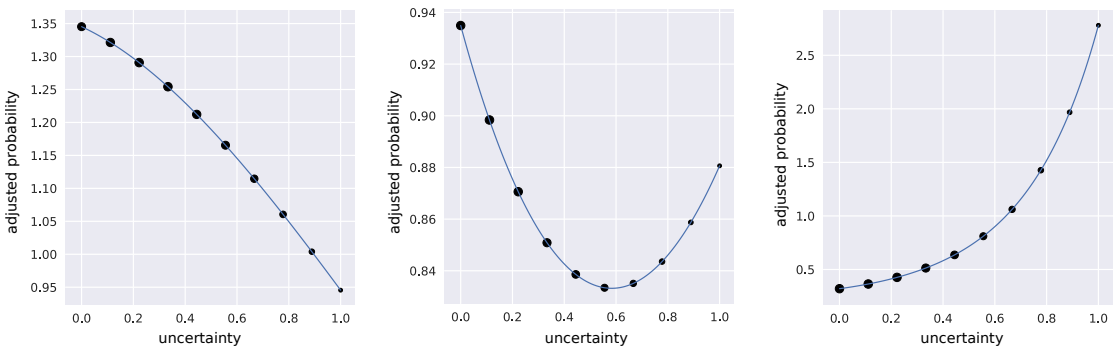


Figure 4.13: Aggregated IBMMs, adjusted for the uncertainty distribution. These plots are for the LPM, using a (a) GBM or (b) RF as an oracle. The corresponding plots for DTs may be found in Section A.8.

While the plots in Figure 4.12 are indicative of the individual distributions they aggregate (most of the individual distributions have similar shapes; see Figure A.7 in Section A.10), this is not true for the adjusted plots in Figure 4.13 - there are diverse variations that are averaged out. We show some of them in Figure 4.14, for different datasets and model sizes, for $model = LPM, oracle = GBM$. The size of the dots on the curve represent $p_B(u_k)$ at the corresponding value of u_k on the x-axis. These are intended to signify robustness of the adjustment, since they occur in the denominator of the scaled ratios.



(a) a1a, size=2 (b) letter, size=2 (c) letter, size=3

Figure 4.14: Adjusted IBMMs for some model sizes and datasets, for $model = LPM, oracle = GBM$. We observe that fairly different distributions may be learned across our experiments.

It is probably important to point out here that typical discussions of uncertainty sampling, such as the classic version (Lewis and Gale, 1994), imply the non-adjusted distributions shown in Figure 4.12.

4.3.1.9 Effect of Model Capacity

We assess the effect of model capacity here as well (in the spirit of Section 3.3.7). The question we attempt to answer is: how does model capacity influence improvements? This is difficult to answer in general since (a) there isn't a standard way to easily quantify capacity across model families, and (b) the notion of model size is subjective. And while the LPM vs DT data indicates a trend, we want to isolate this effect in a manner that is not affected by differences in the model families.

To that end, we adopt the following approach: we use two different instances of GBMs, where the notion of model size is the number of DTs in a GBM (or equivalently, the number of boosting rounds), and their model capacities are decided by the maximum depth of the constituent DTs; these are set to 2 and 5 for these GBM instances. We refer to these as the *GBM-2* and *GBM-5* "pseudo model families" respectively, where we understand *GBM-5* to possess higher capacity than *GBM-2*. Since the training algorithms and model representations are identical for *GBM-2* and *GBM-5*, this setup allows us to sidestep challenges with quantifying capacity for different model families. We use the *LightGBM* library (Ke *et al.*, 2017) for our experiments.

The oracle used is another GBM, with no size/capacity restrictions, learned on the training dataset. The model sizes explored are $\{1, 2, ..., 10\}$. Figure 4.15 shows how

$\delta F1$ varies with model size (denoted as "num_trees") for the datasets `senseit-aco`, `senseit-sei`, `cod-rna` and `higgs`, for each of the models *GBM-2* and *GBM-5*.

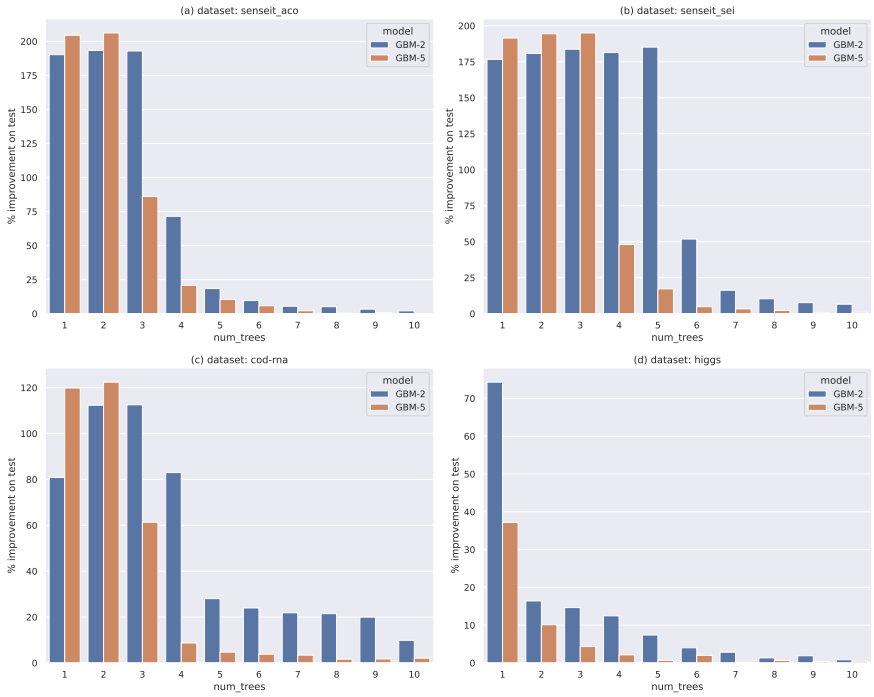


Figure 4.15: The above plots show how the capacity of a model family influences improvements, for different datasets. With a higher max_depth setting for GBMs, the improvements decline faster with an increase in number of trees.

As we might expect, we observe that improvements for *GBM-5*, the model with the higher capacity, diminish faster with increasing size, compared to *GBM-2*.

4.3.2 Comparisons

In this section, we present a comparative evaluation of our technique. We compare against the following techniques:

1. *Supervised* uncertainty sampling: the interpretable model, of a given size, is iteratively trained on a growing subset of training data; this subset starts with the b most uncertain points in the training data, with b-sized batches of the most uncertain points from the remaining training data being progressively added to it. At every iteration, the model is evaluated on a validation set, and the one with the highest $F1$-macro score is picked for comparison. We compare against this technique because:

(a) This explores an obvious possibility: can a heuristic-driven, simple algorithm outperform our algorithm?

(b) Although we borrow this technique from Active Learning (Lewis and Gale, 1994), this version is significantly more powerful, primarily because of the oracle's supervision: we have reliable uncertainty scores from a powerful model. Because of this, we are able to avoid sampling bias arising due to a partial view of the uncertainty distribution (detailed in Section A.5).

(c) Even within the Active Learning community, uncertainty sampling is a strong baseline for Logistic Regression (Yang and Loog, 2018), and by extension, we expect it to be a strong baseline for learning LPMs.

We use a batch size of $b = 10$. The algorithm is described in detail in Section A.4.

2. Density Trees: We also compare against our previous work on density trees since it uses a similar philosophy of determining an optimal distribution to build accurate small models. We use the parameter search space described in the previous chapter in Section 3.3.4.

4.3.2.1 Setup

The experimental setup is identical to the one used for the validation experiments in terms of the datasets (see Section 4.3.1.1), models (Section 4.3.1.2), oracles (Section 4.3.1.3) and optimization search space (Section 4.3.1.5). The metrics differ, and these are described next.

4.3.2.2 Metrics

To compare techniques, we wish to measure the following outcomes over multiple trials:

1. The extent to which a technique is better.

2. The proportion of times a technique is better.

The following properties are desirable for a metric that measures the first kind of outcomes:

1. It should be bounded, so that scores across different data, model sizes, etc., are on the same scale.

2. It should be easy to infer which approach is better.

We introduce a score called the *Scaled Difference in Improvement (SDI)*, that possesses these properties. The SDI is defined in terms of the improvement produced by our method, $\delta F1_{ora}$, and the alternative method, $\delta F1_{alt}$:

$$SDI = \begin{cases} \frac{\delta F1_{ora}}{H} - \frac{\delta F1_{alt}}{H}, & \text{if } H > 0 \\ 0, & \text{if } H = 0 \end{cases} \tag{4.13}$$
$$\text{where } H = \max\{\delta F1_{alt}, \delta F1_{ora}\}$$

The central idea here is that the improvements possible across the competing techniques are in $[0, H]$, and the SDI score measures the difference between the fractions of this range realized by either technique. Note that $H \geq 0$ since $\delta F1_{ora} \geq 0$ and $\delta F1_{alt} \geq 0$. This score has the following intuitive properties:

1. $SDI \in [-1, 1]$

2. $SDI > 0$ when $\delta F1_{ora} > \delta F1_{alt}$

3. $SDI = 0$ when $\delta F1_{ora} = \delta F1_{alt}$

4. $SDI < 0$ when $\delta F1_{ora} < \delta F1_{alt}$

The SDI score may be seen as the *Mean Signed Deviation*[9] *(MSD)*: $\delta F1_{ora} - \delta F1_{alt}$, normalized with the maximum possible improvement H. We don't directly use MSD as $\delta F1 \in [0, \infty)$ makes it unbounded.

[9] https://en.wikipedia.org/wiki/Mean_signed_deviation

For ease of interpretation, we average the SDI scores at the level of a dataset, across model sizes, for a given model and oracle. This averaged score is denoted by $\overline{SDI}$.

To measure the second kind of outcomes, we report the percentage of times $\delta F1_{ora} > \delta F1_{alt}$ across these model sizes. This is denoted as pct_better.

We consider the oracle-based approach to be a meaningful contribution if $\overline{SDI} > \mathbf{0}$ *and* $pct\_better > \mathbf{50\%}$ compared to alternatives.

4.3.2.3 Observations and Analysis

Table 4.4 and Table 4.5 compare our approach to Supervised Uncertainty Sampling and the Density Tree based approach, respectively. All $\delta F1_{ora}$ and $\delta F1_{alt}$ scores used are the *average over five runs*. This is the presentation format followed:

1. For each dataset, model and oracle combination we present two scores: (1) $\overline{SDI}$ and (2) pct_better.

2. Favorable outcome values - $\overline{SDI} > 0$ or $pct\_better > 50$ - are colored green, unfavorable outcomes are colored red, and tied values are unformatted.

3. In the case of Supervised Uncertainty Sampling, Table 4.4, scores are compared across the same oracles, i.e., a score using oracle GBM in our method, is compared to a score from Supervised Uncertainty Sampling using a GBM.

4. Unlike supervised uncertainty sampling, there is no notion of an oracle in the Density Tree based approach. In Table 4.5, for a combination of dataset, model and model size, improved scores from using either the GBM or RF as the oracle are compared to the same reference score from the density tree based approach.

5. We also introduce two special groupings:

   - **ANY**: For each model size, the SDI score considered is the higher of the ones obtained from using the GBM or RF as oracles. The $\overline{SDI}$ and pct_better scores are computed based on these scores. This grouping represents the ideal way to use our technique in practice: try multiple oracles and pick the best.

Table 4.4: LPM, DT compared to Supervised Uncertainty Sampling

	LPM			DT		
dataset	GBM	RF	**ANY**	GBM	RF	**ANY**
cod-rna	0.60, 100.00%	0.25, 87.50%	0.61, 100.00%	0.29, 60.00%	0.39, 60.00%	0.69, 80.00%
ijcnn1	0.28, 66.67%	0.17, 66.67%	0.37, 73.33%	-0.42, 26.67%	0.32, 80.00%	0.32, 80.00%
higgs	0.88, 100.00%	0.23, 60.00%	0.91, 100.00%	0.75, 83.33%	0.28, 66.67%	0.83, 100.00%
covtype.binary	0.49, 93.33%	0.33, 93.33%	0.59, 100.00%	0.06, 44.44%	0.18, 45.45%	0.32, 54.55%
phishing	0.62, 93.33%	0.44, 93.33%	0.65, 93.33%	0.25, 40.00%	-0.27, 13.33%	0.25, 40.00%
a1a	0.23, 93.33%	0.32, 93.33%	0.35, 100.00%	-0.13, 44.44%	0.49, 91.67%	0.58, 100.00%
pendigits	0.73, 100.00%	0.81, 100.00%	0.83, 100.00%	0.22, 60.00%	0.13, 40.00%	0.32, 60.00%
letter	0.92, 100.00%	0.95, 100.00%	0.97, 100.00%	0.46, 73.33%	0.04, 46.67%	0.55, 73.33%
Sensorless	0.63, 100.00%	0.72, 100.00%	0.73, 100.00%	0.63, 73.33%	0.41, 60.00%	0.65, 73.33%
senseit_aco	0.15, 53.33%	0.30, 86.67%	0.31, 86.67%	0.39, 71.43%	0.44, 87.50%	0.60, 87.50%
senseit_sei	0.07, 53.33%	0.27, 60.00%	0.28, 60.00%	-0.18, 37.50%	0.13, 57.14%	0.20, 50.00%
covtype	0.83, 100.00%	0.65, 93.33%	0.85, 100.00%	0.60, 73.33%	0.64, 73.33%	0.82, 86.67%
connect-4	0.13, 60.00%	0.43, 86.67%	0.50, 100.00%	0.11, 60.00%	0.13, 73.33%	0.53, 86.67%
OVERALL	0.50, 85.11%	0.46, 86.17%	0.61, 93.09%	0.23, 57.14%	0.25, 59.75%	0.50, 73.75%

- **OVERALL**: This averages results across datasets, to provide an aggregate view of the comparison.

The entries identified by **OVERALL** and **ANY** provide comparison numbers aggregated over datasets, model sizes and oracles.

Table 4.5: LPM, DT compared to the Density Tree approach.

	LPM			DT		
dataset	GBM	RF	**ANY**	GBM	RF	**ANY**
cod-rna	-0.38, 0.00%	-0.45, 0.00%	-0.33, 0.00%	0.51, 60.00%	0.50, 70.00%	0.65, 80.00%
ijcnn1	0.06, 66.67%	0.11, 80.00%	0.20, 93.33%	0.23, 53.33%	0.68, 100.00%	0.68, 100.00%
higgs	-0.07, 40.00%	-0.07, 40.00%	0.04, 46.67%	0.23, 50.00%	0.61, 83.33%	0.61, 83.33%
covtype.binary	-0.16, 40.00%	-0.33, 13.33%	-0.15, 40.00%	0.23, 66.67%	0.26, 72.73%	0.38, 81.82%
phishing	0.30, 80.00%	0.37, 86.67%	0.38, 86.67%	0.11, 26.67%	-0.00, 26.67%	0.23, 46.67%
a1a	-0.03, 60.00%	0.13, 66.67%	0.13, 66.67%	-0.06, 44.44%	0.43, 75.00%	0.52, 83.33%
pendigits	0.59, 100.00%	0.59, 93.33%	0.62, 100.00%	0.23, 60.00%	0.16, 46.67%	0.25, 60.00%
letter	0.79, 100.00%	0.81, 100.00%	0.81, 100.00%	0.02, 33.33%	-0.34, 13.33%	0.06, 40.00%
Sensorless	0.64, 100.00%	0.65, 100.00%	0.66, 100.00%	-0.23, 20.00%	-0.39, 20.00%	-0.23, 20.00%
senseit_aco	0.55, 100.00%	0.63, 100.00%	0.63, 100.00%	0.50, 85.71%	0.37, 75.00%	0.39, 75.00%
senseit_sei	0.61, 100.00%	0.66, 100.00%	0.67, 100.00%	-0.25, 42.86%	0.51, 100.00%	0.51, 100.00%
covtype	0.20, 80.00%	0.39, 93.33%	0.43, 100.00%	0.26, 66.67%	0.16, 66.67%	0.40, 80.00%
connect-4	0.23, 73.33%	0.24, 66.67%	0.38, 86.67%	-0.23, 33.33%	-0.13, 53.33%	0.08, 66.67%
OVERALL	0.28, 75.00%	0.32, 75.00%	0.37, 81.38%	0.10, 47.06%	0.16, 57.23%	0.31, 67.30%

The predominant amount of values colored green, indicate that our technique performs better in most settings. In both cases, the **OVERALL** +**ANY** entries indicate that our technique works better on average. The pct_better scores in these entries also indicate that we seem to do better much more frequently in the case of $LPMs$ than DTs.

We note here that the space of sampling distributions modeled by our technique

subsume the ones modeled by either competing technique:

1. Supervised Uncertainty Sampling assumes high uncertainty points are favorable; this may be modeled with an IBMM with appropriate parameters.

2. Density Trees learn distributions that are based on the proximity of instances to class boundaries; since uncertainty values also correlate with distance from class boundaries - a high uncertainty value for an instance indicates it's near a class boundary and vice versa - this too is well within the scope of what an IBMM may represent.

Our hypothesis as to when the competing techniques outperform our technique is that the optimal sampling distribution is easier to discover given their distributional assumptions. For example, if the optimal distribution indeed turns out to be one where instances with high uncertainty are preferred, the Supervised Uncertainty Sampling technique would quickly discover this, while our technique would need to navigate a larger search space to converge to this solution. Our technique would likely do better on such problems with a larger iteration budget or an appropriately defined prior; we leave this analysis for future work.

Both Supervised Uncertainty Sampling and our technique use distributions over uncertainty values. This makes it interesting to contrast them, and is reviewed in Section A.6.

4.3.3 Additional Applications

Viewing our technique purely as a tool to find the optimal distribution for effective learning, we explore some additional interesting applications of it in this section.

4.3.3.1 Different Feature Spaces

In our previous experiments, the feature vector representation was identical for the oracle and the interpretable model. This is also what Algorithm 9 implicitly assumes. Here, we consider the possibility of going a step further and using different feature vectors. If

$f_{\mathcal{O}}$ and $f_{\mathcal{I}}$ are the feature vector creation functions for the oracle and the interpretable model respectively, and x_i is a "raw data" instance, then:

1. The oracle is trained on instances $f_{\mathcal{O}}(x_i)$, and provides uncertainties $u_{\mathcal{O}}(f_{\mathcal{O}}(x_i))$.

2. The interpretable model is provided with data $f_{\mathcal{I}}(x_i)$, but the uncertainty scores available to it are $u_{\mathcal{O}}(f_{\mathcal{O}}(x_i))$.

The motivation for using different feature spaces is that the combination $(\mathcal{O}, f_{\mathcal{O}})$ may be known to work well together and/or a pre-trained oracle might be available only for this combination.

We illustrate this application with the example of predicting nationalities from surnames of individuals. Our dataset (Rao and McMahan, 2019) contains examples from 18 nationalities: *Arabic, Chinese, Czech, Dutch, English, French, German, Greek, Irish, Italian, Japanese, Korean, Polish, Portuguese, Russian, Scottish, Spanish, Vietnamese*. The representations and models are as follows:

1. The oracle model is a *Gated Recurrent Unit (GRU)* (Cho *et al.*, 2014), that is learned on the sequence of characters in a surname. The GRU is calibrated with *temperature scaling* (Guo *et al.*, 2017).

2. The interpretable model is a DT, where the features are character n-grams, $n \in 1, 2, 3$. The entire training set is initially scanned to construct an n-gram vocabulary, which is then used to create a sparse binary vector per surname - 1s and 0s indicating the presence and absence of an n-gram respectively.

Figure 4.16 shows a schematic of the setup.

The n-gram representation leads to a vocabulary of ~ 5000 terms, that is reduced to 600 terms based on a χ^2-test in the interest of lower running time (see Section A.12 for details). DTs of different $depth \leq 15$ were trained. A budget of $T = 3000$ iterations was used (the search space for Φ is the same as in Section 4.3.1.5), and the relative improvement in the $F1$ macro score (as in Equation 4.11) is reported, averaged over three runs. Figure 4.17 shows the results.

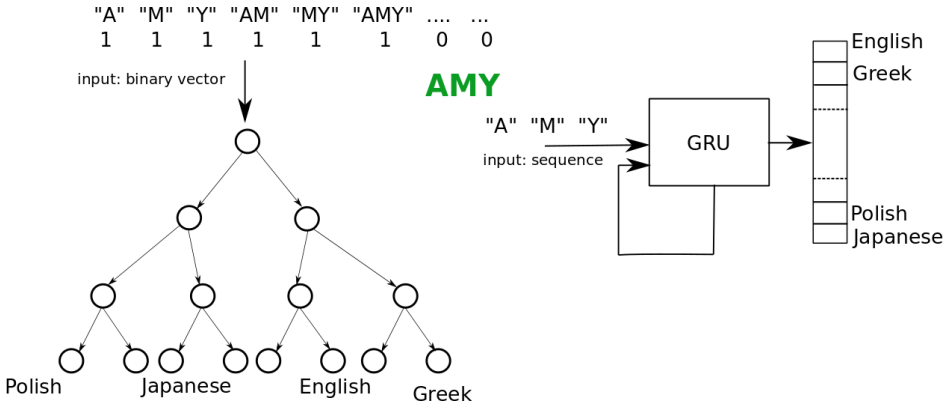


Figure 4.16: The feature representations for the oracle and the interpretable model may be different. Consider the name "Amy": the GRU is provided its letters, one at a time, in sequence, while the DT is given an n-gram representation of the name.

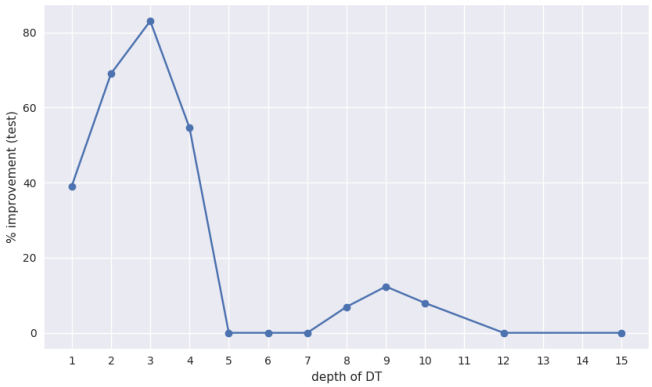


Figure 4.17: Improvements $\delta F1$ are shown for different depths of the DT.

We see large improvements at small depths, that peak with $\delta F1 = 83.04\%$ at $depth = 3$, and then again at slightly larger depths, which peak at $depth = 9$ with $\delta F1 = 12.34\%$.

To obtain a qualitative idea of the changes in the DT using a oracle produces, we look at the prediction rules for *Polish* surnames, when DT $depth = 3$. For each rule, we also present examples of true and false positives.

Baseline rules - $precision = 2.99\%, recall = 85.71\%, F1 = 5.77\%$:

Rule 1. $k \wedge ski \wedge \neg v$

- True Positives: *jaskolski, rudawski*
- False Positives: *skipper (English), babutski (Russian)*

Rule 2. $k \wedge \neg ski \wedge \neg v$

- True Positives: *wawrzaszek, koziol*
- False Positives: *konda (Japanese), jagujinsky (Russian)*

Oracle-based DT rules - $precision = 25.00\%, recall = 21.43\%, F1 = 23.08\%$:

Rule 1. $ski \wedge \neg(b \vee kin)$

- True Positives: *jaskolski, rudawski*
- False Positives: *skipper (English), aivazovski (Russian)*

We note that the baseline rules are in conflict w.r.t. the literal "ski", and taken together, they simplify to $k \wedge \neg v$. This makes them extremely permissive, especially *Rule 2*, which requires the literal "k" while needing "ski" and "v" to be absent. Not surprisingly, these rules have high recall ($= 85.71\%$) but poor precision ($= 2.99\%$), leading to $F1 = 5.77\%$.

In the case of the oracle-based DT, now we have only one rule, that requires the atypical trigram "ski". This improves precision ($= 25\%$), trading off recall ($= 21.43\%$), for a significantly improved $F1 = 23.08\%$.

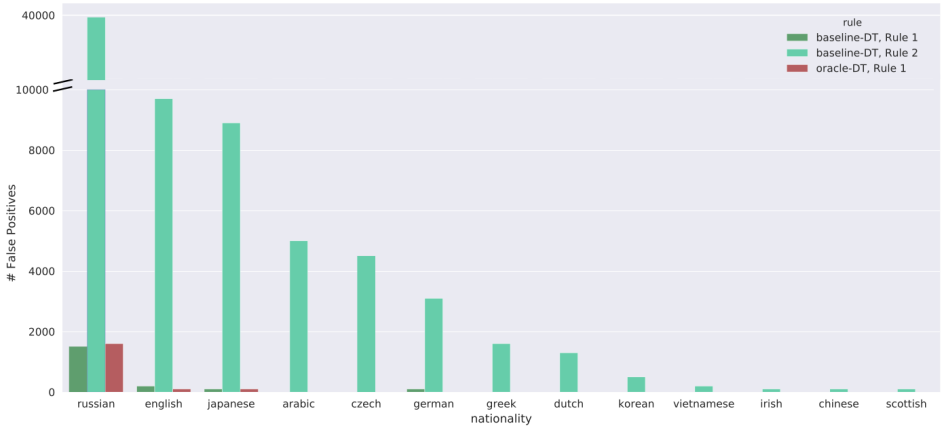


Figure 4.18: The distribution of nationalities in false positive predictions for the baseline and oracle based models, shown for predicting *Polish* names. Only nationalities with non-zero counts are shown.

The difference in rules may also be visualized by comparing the distribution of nationalities represented in their false positives, as in Figure 4.18. We see that the baseline DT rules, especially *Rule 2*, predict many nationalities, but in the case of the DT learned using the oracle, the model confusion is concentrated around *Russian* names, which is reasonable given the shared *Slavic* origin of many *Polish* and *Russian* names.

We believe this is a particularly powerful and exciting application of our technique, and opens up a wide range of possibilities for translating information between models of varied capabilities.

4.3.3.2 Size-Constrained Training Sample

Recall from Section 4.2.3, we make use of a parameter N_s, denoting sample size, that we had constrained to $\in [400, 10000]$ (Section 4.3.1.5) in our experiments. But it is possible to set this to much smaller values to study the sampling distribution for patterns, significance of regions in the input space, etc. Figure 4.19 shows an example of this: we set $N_s \in [50, 50]$ (so it can take exactly one value, 50), and for the dataset shown in Figure 4.19(a), we visualize the sampling distribution when the model is a DT of $depth = 2$ in Figure 4.19(b) vs when $depth = 4$ in Figure 4.19(c). The dataset is balanced, and the oracle used is a GBM.

We see the following interesting patterns: (a) at $depth = 2$, the DT picks points from both regions where $label = 1$, but the larger region shows higher density. This is

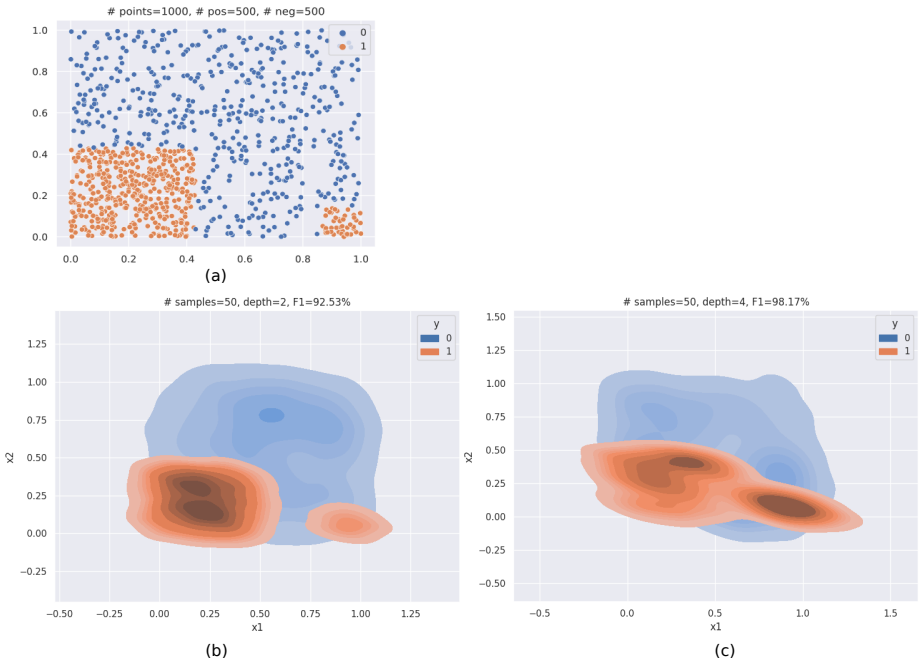


Figure 4.19: Our technique might be used to identify the optimal sample of a given size. (a) shows the original dataset. (b) and (c) visualize the learned distribution of points, using a KDE, for DTs with $depth = 2$ and $depth = 4$ respectively, for a sample size of 50. NOTE: the connection shown in (c), between the two originally disjoint regions with $label = 1$, is an artifact of the KDE.

possibly because owing to its limited capacity, the model is able to effectively parameterize only one region, and therefore it prioritizes correct classification of points around the larger region, (b) at $depth = 4$, we see increased sampling density in the smaller region with $label = 1$ as well.

4.3.3.3 Vector Model Size

Although we have been using a scalar notion of model size - depth for DT, number of terms for LPM, number of trees for a GBM - Algorithm 9 doesn't restrict us from using a vector-valued model size η. For example, in the case of GBMs, we may consider the notion of model size $\eta = [max\_depth, num\_trees]$, where the quantities respectively denote the maximum depth allowed for each constituent DT in a GBM, and the number of DTs in the GBM. In Figure 4.20 we show how improvements for GBMs vary when $1 \leq max\_depth \leq 5$ (x-axis) and $1 \leq num\_trees \leq 5$ (y-axis); the oracle used is a GBM as well (unconstrained in size), and results for these datasets are shown: (a) `higgs` (b) `cod-rna` (c) `senseit-sei` and (d) `senseit-aco`. The improve-

ments are averaged over three runs. We observe the familiar pattern that as model sizes increase, in terms of both max_depth and num_trees, improvements decrease.

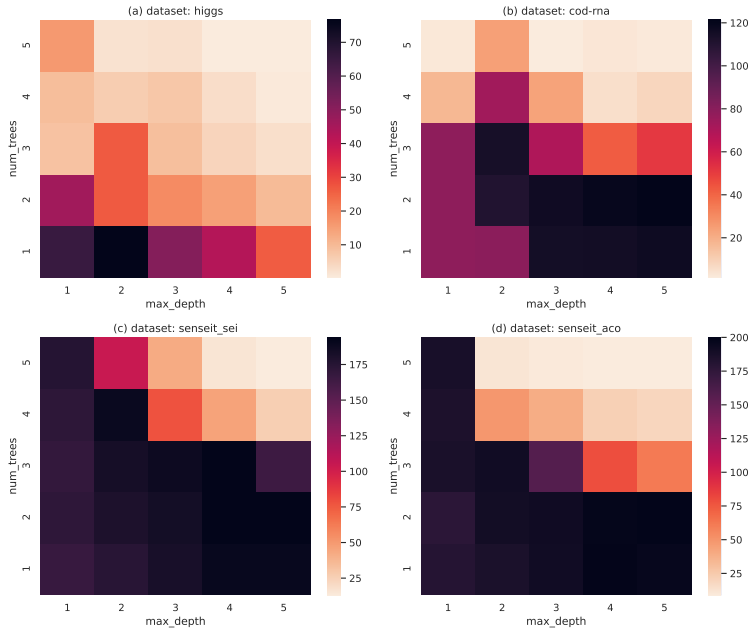


Figure 4.20: Improvements in test $F1$-macro for multiple datasets for different sizes of GBM models are shown. Here, model size is the combination of *max_depth* and *number of trees* in the GBM model. Greater improvements are seen at lower sizes.

4.3.4 Extrinsic Comparisons

Thus far our evaluations have looked at improvements produced by our technique for a given model size. Here, we show that such improvements are also competitive with *task-specific techniques* that produce accurate small models. We consider the problems of *explainable clustering* and *prototype-based classification* for this evaluation.

4.3.4.1 Explainable Clustering

The first task we investigate is the problem of *Explainable Clustering*. Introduced by Moshkovitz *et al.* (2020), the goal is to explain cluster allocations as discovered by techniques such *k-means* or *k-medians*. This is achieved by constructing axis-aligned decision trees with leaves that either exactly correspond to clusters, e.g., *Iterative Mistake Minimization (IMM)* (Moshkovitz *et al.*, 2020), or are proper subsets, e.g., *Expanding Explainable k-Means Clustering (ExKMC)* (Frost *et al.*, 2020). We consider the former

case here, i.e., a tree must possess exactly k leaves to explain k clusters.

We denote a specific clustering by C. If the assigned cluster for an instance $x_i, i = 1...N$, is denoted by $C(x_i)$ where $C(x_i) \in \{1, 2, ..., k\}$, and the cluster centroids are denoted by $\mu_j, j = 1, ..., k$, then the cost of clustering C is given by:

$$C = \frac{1}{N} \sum_{j=1}^{k} \sum_{\{x_i | C(x_i) = \mu_j\}} ||x_i - \mu_j||_2^2 \tag{4.14}$$

In the case of an explanation trees with k leaves, μ_j are centroids of leaves. Cluster explanation techniques attempt to minimize this cost.

The price of interpretability maybe measured as C_{Ex}/C_{KM}, where C_{Ex} is the cost achieved by an explanation tree, and C_{KM} is the cost obtained by a standard k-means algorithm. We refer to this ratio as the *cost ratio*; it assumes values in the range $[1, \infty]$, where the lowest cost is obtained when using k-means, i.e., C_{Ex} and C_{KM} are the same.

One may also indirectly minimize the cost in the following manner: use k-means to produce a clustering, use the cluster allocations of instances as their labels, and then learn a standard decision tree for classification, e.g., CART. This approach has been shown to be often outperformed by tree construction algorithms that directly minimize the cost in Equation 4.14.

Algorithms and Hyperparameters: Within the family of explainable clustering techniques, we use *Iterative Mistake Minimization (IMM)* (Moshkovitz *et al.*, 2020) and *ExShallow* (Laber *et al.*, 2021), that minimize the cost in Equation 4.14. We compare them with CART - without and with applying our technique; the latter is referred to as *c_CART*. CART and c_CART maximize the $F1(macro)$ score of predicting the cluster labels. The metric of comparison is cost ratio. For our technique, the oracle used is a GBM and the number of iterations is set as $T = 2000$. ExShallow is parameterized by λ, which is used to control the trade-off between clustering cost and explanation size. This is set as $\lambda = 0.03$; this value is used in the original paper for various experiments. For IMM there are no parameters to tune. The reference implementations available at `https://github.com/navefr/ExKMC` and `https://github.com/lmurtinho/ShallowTree` respectively, were used. For CART,

we used the implementation in *scikit* (Pedregosa *et al.*, 2011); during training, the maximum number of leaves were set to the number of clusters k, and the parameter *class_weight* is set to *"balanced"* to counter imbalance due to disparate cluster sizes[10].

Experiment setup: The comparison is performed over five datasets (limited to 1000 instances), and for each dataset, $k = 2, 3, ..., 10$ clusters are produced. Results are reported over *five* trials. Evaluations are performed over the following publicly available datasets: *avila*, *covtype*, *covtype.binary*, *Sensorless* (Chang and Lin, 2011) and *mice-protein* (Dua and Graff, 2017). We *specifically* picked these datasets since CART is known to perform poorly on them (Frost *et al.*, 2020; Laber *et al.*, 2021).

Observations: Figure 4.21 presents our results. For each of the dataset-specific plots (a), (b), (c), (d) and (e), the 95% confidence interval, in addition to mean normalized cost, is shown. The cost for k-means is shown for reference. The final plot (f) shows the *mean ranks* of the various techniques (lower is better), and its title shows the *p-value*$= 6.69 \times 10^{-6}$ of a Friedman's test conducted over *the top three techniques*[11]: we restrict the test to top candidates since otherwise it would be very easy to obtain a low/favorable score, due to the high normalized costs for CART.

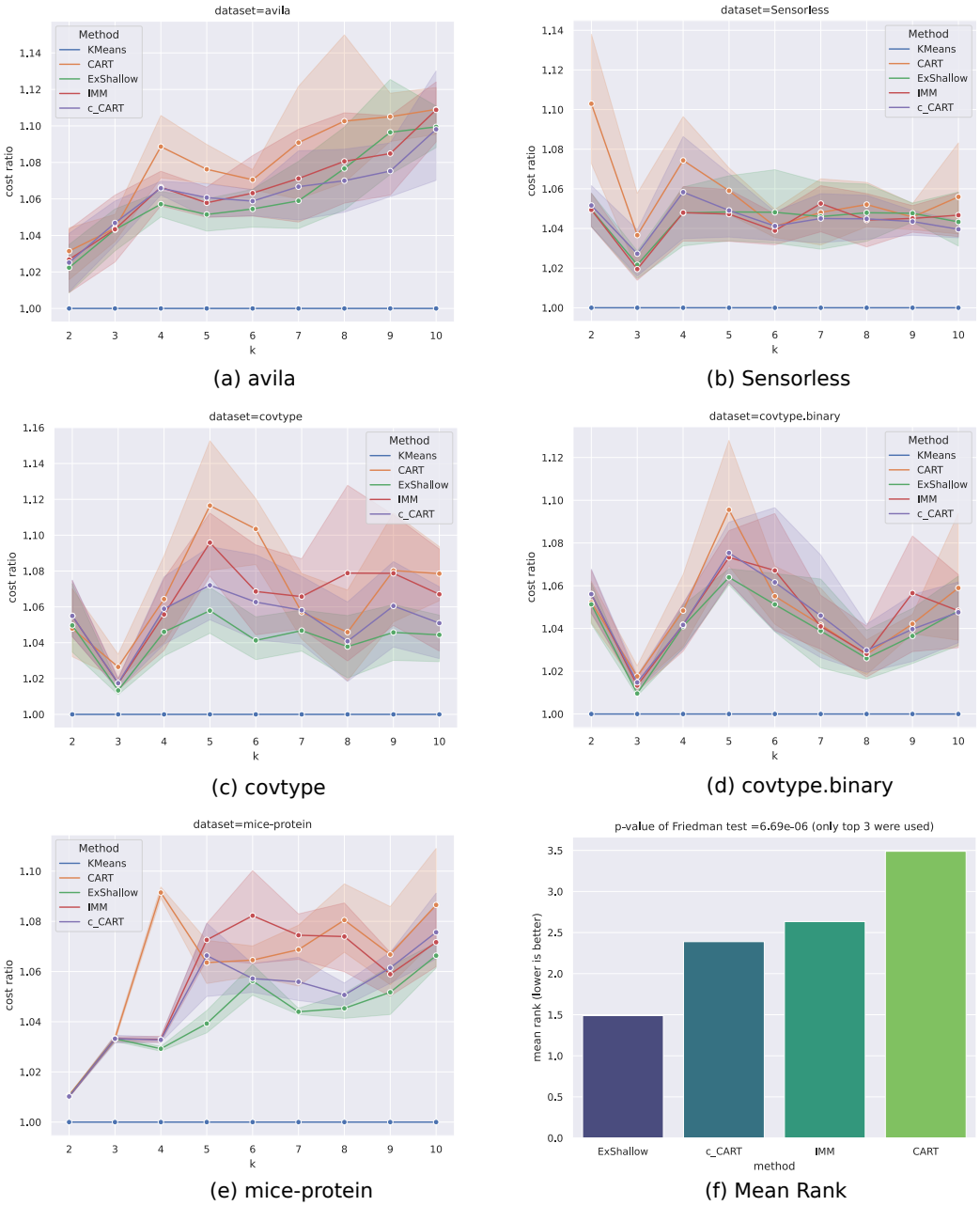
We observe that although CART performs quite poorly, the application of our technique drastically improves its performance, to the extent that it competes favorably with techniques like IMM and ExShallow; its mean rank places it between them. For a Wilcoxon signed-rank test between IMM and c_CART we have $p = 0.0155$, showing the performance of c_CART maybe interpreted as significantly different/better than IMM. This is especially surprising given that it doesn't explicitly minimize the cost in Equation 4.14.

4.3.4.2 Prototype-based Classification

Next, we study the problem of prototype-based classification. At training time, such techniques identify "prototypes" within the data (actual training instances or generated instances), that maybe used to classify a test instance based on their similarity to them.

[10]Not surprisingly, we have observed that CART obtains better cost ratios with this setting.

[11]The test is typically performed over datasets, but here we use the combinations of datasets and k, the number of clusters. Scores across trials are averaged.



(a) avila (b) Sensorless

(c) covtype (d) covtype.binary

(e) mice-protein (f) Mean Rank

Figure 4.21: Comparisons over various explainable clustering algorithms are shown. Sub-figures (a), (b), (c), (d), (e) are specific to datasets, mentioned in the title. (f) shows the mean ranks of techniques; the Friedman test is conducted over the **top three** techniques only, with $p = 6.69 \times 10^{-6}$.

A popular technique in this family is the *k-Nearest Neighbor (kNN)*. These are simple to interpret, and if a small but effective set of protoypes maybe identified, they can be convenient to deploy on edge devices (Gupta *et al.*, 2017; Zhang *et al.*, 2020*b*). Research in this area has focused on minimizing the number of prototypes that need to be retained while minimally trading off accuracy. This is also the notion of model size we use.

Algorithms and Hyperparameters: We compare to the following algorithms:

1. Fast Condensed Nearest Neighbor Rule (Angiulli, 2005): This technique identifies a minimal subset of points, known as a "consistent subset" designed to maximize nearest neighbor based classification accuracy. Among its variations we use FCNN1 (described in the original paper). This maximizes accuracy wrt the "1NN" rule, i.e., predict the same label as the closest neighbor. We used our own implementation of the algorithm.

2. ProtoNN (Gupta *et al.*, 2017): This technique uses a *Radial Basis Function (RBF)* kernel to aggregate influence of neighbors. Synthetic prototypes are learned and additionally a "score" is learned for each of them that designates their contribution towards each label. The prediction function sums the influence of neighbors using the RBF kernel, weighing contribution towards each class using the learned score values.The method also allows for reducing dimensionality, but we don't use this aspect[12]. The various parameters are learned via gradient based optimization. The RBF kernel accepts the kernel bandwidth γ as a parameter; to identify an appropriate value, we perform a grid search over the following space: $[0.001, 0.01, 0.1, 1, 10]$.

   We used the reference implementation available as part of the *Microsoft EdgeML* library (Dennis, Don Kurian and Gaurkar, Yash and Gopinath, Sridhar and Goyal, Sachin and Gupta, Chirag and Jain, Moksh and Jaiswal, Shikhar and Kumar, Ashish and Kusupati, Aditya and Lovett, Chris and Patil, Shishir G and Saha, Oindrila and Simhadri, Harsha Vardhan, 2021).

[12]The implementation provides no way to switch off learning a projection, so we set the dimensionality of the projection to be equal to the original number of dimensions. This setting might however learn a transformation of the data to space within the same number of dimensions, e.g., translation, rotation.

3. Stochastic Neighbor Compression (SNC) (Kusner *et al.*, 2014): This also uses a RBF kernel to aggregate influence of neighbors, but unlike ProtoNN, the prediction is performed via the 1NN rule. The technique bootstraps with randomly sampled prototypes, and then modifies their coordinates for greater accuracy using gradient based optimization. The technique maybe extended to reduce the dimensionality of the data (and prototypes). We don't use this aspect, and hence refer to as SNC_basic in our experiments. A grid search over values of γ (parameter for the RBF kernel) is performed: $[0.001, 0.01, 0.1, 1, 10]$.

   We were unable to locate the reference implementation mentioned in the paper, so we implemented our own version, with the help of the *JAXopt* library (Blondel *et al.*, 2021).

The above algorithms are compared to *Radial Basis Function Networks (RBFN)* (Broomhead and Lowe, 1988). As the standard version, we use centroids of clusters discovered by *k-means* as protoypes - we refer to this as *KM_RBFN*. The oracle based technique directly provides prototypes to the RBFN via sampling (their precise number is controlled by fixing the sample size $N_{s,t}$ in Algorithm 9), and is referred to as *c_RBFN*. Because there is no way to perform dimensionality reduction using RBFNs, to keep comparisons fair we don't use this aspect of SNC or ProtoNN.

Experiment setup: We compare the techniques across five datasets and five trials. The following publicly available datasets are used: *covtype.binary, senseit-sei, senseit-aco, phishing* (Chang and Lin, 2011) and *adult* (Dua and Graff, 2017). The training and test dataset sizes were limited to 1000 instances each. The number of prototypes experimented with are $\{2, 4, 6, ..., 20\}$. This does not apply to FCNN1 since it does not accept size of the prototype set as input; instead it iteratively constructs a prototype set till it meets a stopping criteria. The size of this set at different iterations may vary across trials; hence these are binned to compute confidence intervals of prediction accuracy across trials. The metric of comparison is the F1 (macro) score for classification.

Observations: Figure 4.22 shows results from our experiments. At the outset, we note FCNN1 performs quite poorly - this matches the observations in Kusner *et al.* (2014). Plot (e) shows that ProtoNN performs the best. While c_RBFN improves

upon KM_RBFN, it still ranks behind SNC_basic. For a Friedman's test[13], we exclude FCNN1 (otherwise it would be easy to obtain a favorable score because of its poor performance), and report a *p-value*$= 3.50 \times 10^{-8}$. The low score indicates a statistically significant performance difference across techniques. We note the following *p-values* from the Wilcoxon signed-rank test:

1. SNC_basic vs c_RBFN, *p-value*$= 0.1260$: The test scores of SNC_basic aren't significantly better than c_RBFN, despite the difference in their mean ranks.. In fact if our confidence threshold were $\alpha = 0.1$, we wouldn't have been able to conclude SNC_basic is better.

2. c_RBFN vs KM_RBFN, *p-value*$= 0.00016$: This tells us that the improvement of c_RBFN upon KM_RBFN is statistically significant.

We conclude that c_RBFN significantly improves the standard KM_RBFN, making it competitive with SNC_basic, a technique that is specialized for this task.

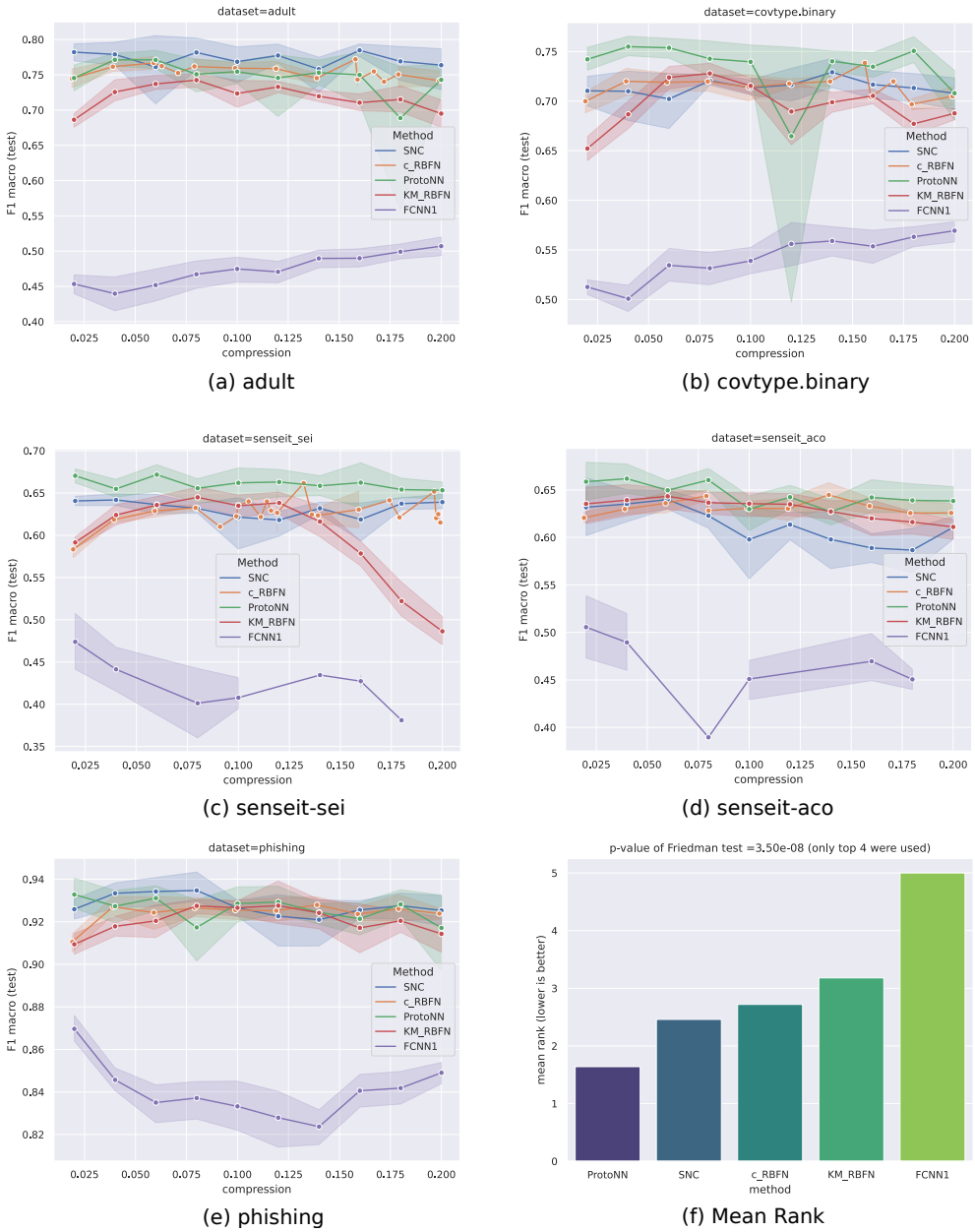
4.3.5 Summary

We summarize our observations from our experiments here:

1. For all combinations of interpretable and oracle models - $\{LPM, DT\} \times \{GBM, RF\}$ - we see good improvements, $\delta F1$, especially at small sizes (Section 4.3.1.6). Sometimes these may be $> 100\%$. For model sizes beyond a point, we observe $\delta F1 \approx 0$.

2. The results in Section 4.3.2 strongly indicate that the precise relationship of the sampling distribution and the uncertainty needs to be learned, and a heuristic strategy of exclusively sampling high uncertainty points is not optimal . We believe this is an important result, especially given that this is true for the supervised version of uncertainty sampling, which is significantly more powerful than standard uncertainty sampling.

[13] As in the case of explainable clustering, we perform the test over a combination of dataset and number of prototypes



(a) adult (b) covtype.binary

(c) senseit-sei (d) senseit-aco

(e) phishing (f) Mean Rank

Figure 4.22: Different prototype-based classifiers are compared. Sub-figures (a), (b), (c), (d), (e) are specific to datasets, mentioned in the title. (f) shows the mean ranks of techniques; the Friedman test is conducted over the **top four** techniques only, with $p = 3.50 \times 10^{-8}$.

3. Our approach produces better accuracy, in general, compared to both supervised uncertainty sampling and the density tree based approach.

   The results in Table 4.4 and Table 4.5 from Section 4.3.2.3 are summarized in Table 4.6. Recall that the combination **OVERALL** + **ANY** averages over datasets and oracles; Table 4.6 shows these summary statistics.

   Table 4.6: Summary comparison results, **OVERALL** + **ANY**

compared to	model	$\overline{SDI}$	pct_better
supervised uncertainty sampling	LPM	0.61	93.09%
	DT	0.50	73.75%
density trees	LPM	0.37	81.38%
	DT	0.31	67.30%

   We observe that density trees are more competitive to our technique than supervised uncertainty sampling: smaller $\overline{SDI}$ and pct_better scores. This is to be expected since the density tree based approach is capable of learning flexible distributions over the input space.

4. A remarkable fact of practical value is that we *don't tune the parameters* Φ for a specific problem. The value ranges for these are fixed across tasks, with only the iteration budget T being changed - as described in Section 4.3.1.5. This highlights another strength of the technique: Φ need not be tuned for obtaining meaningful improvements, as long as it admits a broad enough set of uncertainty distributions.

5. Section 4.3.3 showcases the generality of the proposed technique: we successfully used it with differing feature spaces across the oracle and the interpretable model, to identify the optimal training sample for a given size, and with vector valued model sizes. These applications considerably broaden the impact of our work.

6. Section 4.3.4 shows that improvements produced by our technique are competitive with task-specific specialized techniques. This is surprising, and further attests to the broad utility of the technique.

Importantly, the various positive results from this section should be seen as representative of the proposed *framework*, and not just our *implementation*. In other words,

these results establish a lower bound for the outcomes, because they may be potentially improved by using different components within the larger framework, e.g., by using a different optimizer from among the ones discussed in Feurer and Hutter (2019*b*) or Turner *et al.* (2021) .

4.4 Discussion

Having looked at both the theory and empirical outcomes, we revisit a few points of interest in this section.

1. **Effect of flattening**: We first consider the question: does flattening (Section 4.2.5) help? Table 4.7 contrasts *improved* $F1$ scores obtained without (rows denoted as "original") and with (denoted "flattened") flattening the uncertainty distribution. This is shown for the datasets `Sensorless` and `covtype.binary`, for model $size \in \{1, 2, 3\}$, with $model = LPM$ and $oracle = GBM$. Two different parameter settings are used: (a) Setting 1 is what we have used in the experiments in Section 4.3: maximum allowed $Beta$ components are 500 and $scale = 10000$ (b) Setting 2 looks at much lower values of these parameters where maximum allowed components is 50 and $scale = 10$. The scores presented are the average over three trials.

   We observe that while flattening influences results, other parameters determine the magnitude of its effect. At Setting 1, `Sensorless` is affected at $size = 1$ (flattening is better), but at higher sizes the differences seem to be from random

Table 4.7: Improved scores averaged over three trials, shown for different parameter settings, with and without flattening. Here, Setting 1 is $\{max\_components = 500, scale = 10000\}$ and Setting 2 is $\{max\_components = 50, scale = 10\}$. "curr." signifies this is the current setting for our experiments in Section 4.3, while "low" signifies lower values of parameters. Highlighted cells indicate positve effect of flattening.

		Setting 1 (curr.)			Setting 2 (low)		
dataset	dist.	1	2	3	1	2	3
Sensorless	original	0.39	0.54	0.57	0.38	0.42	0.41
	flattened	0.44	0.53	0.55	0.43	0.54	0.59
covtype.binary	original	0.66	0.69	0.71	0.64	0.66	0.71
	flattened	0.68	0.73	0.73	0.65	0.71	0.71

variations across trials. At Setting 2 however, the differences are seen for $size \in \{1, 2, 3\}$ (flattening is better). For `covtype.binary` only $size = 2$ seems to be affected in either setting.

Recall we had noted in Figure 4.7 that the datasets `Sensorless` and `covtype.binary` have non-smooth and smooth uncertainty distributions respectively. The observations in Table 4.7 align well with the expectation that `Sensorless` is positively affected by the transformation, while results for `covtype.binary` remain mostly unchanged.

Based on these tests, we hypothesize that for non-smooth uncertainty distributions, flattening makes our technique robust across parameter settings. It does not affect smooth distributions in a significant way. Of course, rigorous and extensive tests are required to conclusively establish this effect.

2. **Alternative Parameterization**: Instead of using shape variables $\{a, b, a', b'\}$ to characterize the IBMM (Section 4.2.3), which lie in the interval $(0, \infty)$, one might wonder if its simpler to parameterize based on the mean, $\mu \in [0, 1]$ (bounded by the range of uncertainty values), and standard deviation, $\sigma \in [0, 0.5]$ (also bounded; this range is a property of the $Beta$ distribution). While this is appealing, we need to consider that unlike the *Normal* distribution, μ and σ are not independent for a $Beta$ distribution. For instance, $\mu \to 1 \implies \sigma \to 0$. The optimization would need to account for this dependence, and we would lose our current convenience of using only box constraints. The scatter plot in Figure 4.23 marks the different combinations of μ and σ for which valid $Beta$ distributions exist.

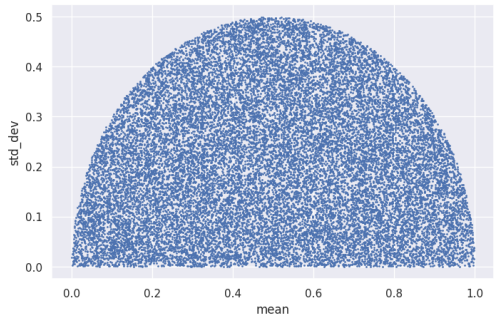


Figure 4.23: Blue dots indicate a valid $Beta$ distribution exists for the corresponding mean and standard deviation values.

3. **Measuring compaction**: As we have indicated earlier, a possible area of application of this work might be model compression. We would like to point out that the *compaction profile* (Figure 4.9, Figure A.6) plots emphasize this use-case: they're a visual tool to determine the minimal model size achievable using our technique, given a baseline model size.

   To formalize this connection, we introduce the score *Compaction Index (CI)* that denotes the extent of model size decrease possible, up to a size where $\delta F1 \approx 0$. Figure 4.24 shows a sample compaction profile. The CI score, where $CI \in [0, 1]$, is the ratio of the area in red to the area in green.

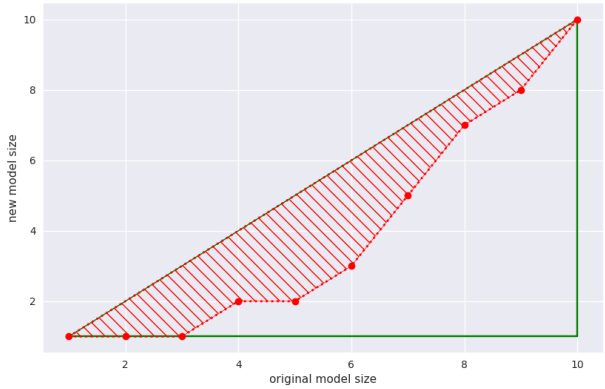


Figure 4.24: Compaction Index

The more reduction in model size our technique can obtain, the closer the red curve is to the green boundary, and $CI \approx 1$. If no reduction is possible at any model size, the red line coincides with the diagonal and $CI = 0$. Clearly, this score is specific to a model family $\mathcal{F}$, a training algorithm f and a specific notion of model size. And ideally, this should be averaged over all possible datasets and oracles.

Here are the CI scores for our experiments :

- $LPM : CI = 0.57$
- $DT : CI = 0.17$

These scores indicate that $LPMs$ may be compacted better than DTs, for the respective notions of size we use here - this may also be seen from the plots in Figure 4.8, where the improvements for DTs decrease faster, with growing model

size, than those for $LPMs$. Section 5.3 presents some additional discussion on model compression.

4. **Upper bound of improvements**: In Equation 4.2, and then in Equations 4.3 and 4.4, the improved accuracy of the interpretable model is shown bounded by the oracle accuracy. For example, see the rightmost term in Equation 4.2, reproduced below:

$$accuracy(M_{\mathcal{I}p\eta}, p) \lessapprox accuracy(M_{\mathcal{I}q\eta}, p) \lessapprox accuracy(M_{\mathcal{O}p*}, p) \tag{4.15}$$

We empirically show this to be true now. In Figure 4.25, we show the distribution of relative difference between the improved accuracy of a LPM model and the accuracy of a GBM oracle.

Using the notation in the equation above, we calculate the relative difference $\Delta F1$ as:

$$\Delta F1 = \frac{accuracy(M_{\mathcal{I}q\eta}, p) - accuracy(M_{\mathcal{O}p*}, p)}{accuracy(M_{\mathcal{O}p*}, p)} \tag{4.16}$$

Here, of course, we measure accuracy using the $F1$ macro score.

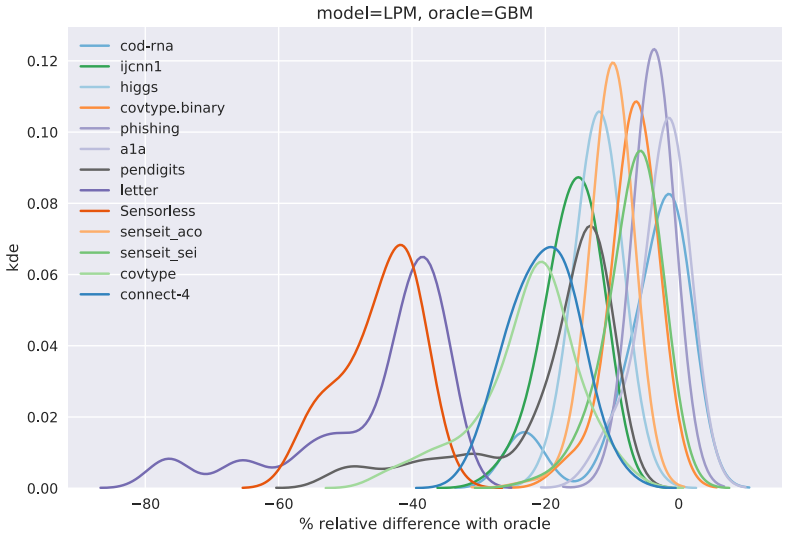


Figure 4.25: Distribution of %age accuracy difference from the oracle accuracy.

There is one distribution plotted per dataset, where the distribution uses information from multiple runs, for multiple model sizes. It may be seen in Figure 4.25 that all relative differences are at most 0 (there is some spillover to the right of 0 owing to the use of KDEs for visualization).

For precise numbers, we look at Table 4.8, which lists the %age of cases where the interpretable model's accuracy exceeded that of the oracle, and the average value of the relative difference for the cases where it is positive. We also consider the statistical significance of these values: using a one-sided *Wilcoxon signed-rank test*, we determine the *p-value* for the null hypothesis that accuracy scores produced our technique exceed those of the oracle. This is a *paired* test where oracle and model scores for a dataset are paired, for a given combination of an interpretable model and an oracle. Only the largest model built for the dataset is used.

For instance, for dataset *a1a*, we note from Table 4.3 that using the GBM oracle, LPMs of sizes $1 \ldots 15$ are constructed; however, this test uses *only* the LPM of size 15. We don't use all sizes since that biases results in our favor, as smaller models are not expected to match the oracle's accuracy.

Table 4.8: The percentage of cases where we see positive relative difference w.r.t. oracle, and the mean of these positive difference are shown. The *p-value* for the hypothesis that model improvements exceed oracle are also provided.

model	oracle	%age +ve cases	mean +ve value	*p-value* (largest models)
LPM	GBM	1.60%	0.81	0.000737
LPM	RF	4.47%	1.36	0.001183
DT	GBM	3.51%	1.84	0.001488
DT	RF	2.85%	0.65	0.000936

4.5 Conclusion

In this chapter we introduced a second novel technique to reduce the trade-off between the size and accuracy of a model. The practical implication of this work is that instead of picking an interpretable model family based on accuracy, one may use our method to construct accurate models for their preferred model family. A Python implementation has been made publicly available: Ghose (2020).

Producing an accurate model is formulated as an optimization problem of identifying training data that maximizes learning, where the optimization is guided by an oracle. It retains the following favorable properties of the density tree based approach:

1. Irrespective of the dimensionality of the data, the optimization uses a small fixed set of seven variables (density trees use eight).

2. A reasonable choice of box constraints over the search space produces good results across datasets.

3. The technique is model-agnostic, allowing for use with an arbitrary interpretable model family.

4. Its a framework, which leaves open the possibility of conveniently improving upon it as better optimizers become available.

Additionally, it improves upon the density tree based approach in these significant ways:

1. It is more accurate.

2. It is more flexible in the sense of being able to use an arbitrary oracle.

3. The oracle and the interpretable models may use different feature spaces.

This work also reaffirms some intriguing deeper findings:

1. Train and test distributions need not be identical for optimal learning.

2. Our observations here point to a "small model effect": this difference in distributions exists for small model sizes, and it is in this model size regime that we observe most improvements.

The general theme of the proposed technique, that of shaping data density to influence accuracy, as well as the deeper results, offer promising directions for future research - we discuss them in Section 5.3.

CHAPTER 5

Conclusions and Future Directions

In this chapter, we first revisit our primary metric, $\delta F1$, and discuss an alternate calculation. We then conclude the dissertation by summarizing our contributions and suggesting avenues for future research.

5.1 Analysis of Small Improvements

As mentioned in Chapters 3 and 4, the lower bound of $\delta F1$ is set to 0. This is done since the baseline data distribution, i.e., $p_o = 1$, is part of the search space of distributions available to the optimizer, and thus in the ideal setting of a much larger optimization budget and number of trials, $\delta F1 \not< 0$.

However, to analyze negative improvements in our setup that has finite budget, we note:

1. The baseline doesn't provide a fair reference since it possesses multiple advantages over a model trained within the optimizer: more available training data, model selection via a robust cross-validation process and in the case of DTs, exploration of a larger parameter space (discussed in Sections 3.3.2 and 4.3.1.2).

   This baseline was picked for our experiments to reflect the practical scenario that a baseline model would be trained in a standard fashion, i.e., external to our technique.

2. In general, the baseline is not a good reference for small values of $\delta F1$, negative *and* positive, because of its inherent advantages.

3. The model produced at the first iteration serves as a fair reference. Recall from the discussions of Algorithms 7 and 9, here the configuration settings of $p_o \rightarrow 1$ and $N_s \rightarrow |D_{train}|$ mimic the data distribution available to the baseline models, and

being constructed within the optimizer, it has access to the exact same resources as the optimized model.

We recompute $\delta F1$ with the model at the first iteration as the baseline. To account for variability across runs, we perform a *t-test* between the *validation* $F1$ macro scores at the first iteration and the optimal iteration, across runs. This is only shown for the oracle-based technique, since (a) we recommend its use over the density tree technique because it is more accurate in general (Section 4.3.2.3), and (b) we expect similar behavior for the alternate $\delta F1$ in the case of density trees, since the optimizer plays a similar role.

Our null hypothesis is the model at the first iteration is better, and we accept the alternative only when $p < 0.1$. Improvements $\delta F1$ are reported on test scores. Here, $\delta F1 \in (-\infty, \infty)$.

These improvements are shown in Table 5.1, where negative improvements are indicated in red. Also note that some of the small positive $\delta F1$ scores from Table 4.3, e.g., dataset *a1a* for model size of one, now become exactly 0.

Table 5.1: This table shows the average improvements, $\delta F1$, over five runs for different combinations of models and oracles: $\{LPM, DT\} \times \{GBM, RF\}$. The improvements are measured relative to the model at the first iteration.The best improvement for a model size and oracle is indicated in bold. Here, $\delta F1 \in (-\infty, \infty)$. Negative improvements are shown in red.

dataset	model_ora	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
cod-rna	lpm_gbm	1.39	12.53	**14.76**	**15.73**	14.97	12.00	0.00	**0.08**	-	-	-	-	-	-	-
	lpm_rf	**2.66**	**13.91**	14.69	15.34	**16.06**	**12.49**	**8.30**	0.00	-	-	-	-	-	-	-
	dt_gbm	**0.00**	**0.00**	0.00	1.26	0.00	**0.00**	0.00	0.00	-0.28	0.08	-	-	-	-	-
	dt_rf	0.00	0.00	**1.78**	**2.28**	**0.39**	-0.02	**0.17**	**0.47**	**0.00**	**0.72**	-	-	-	-	-
ijcnn1	lpm_gbm	-0.16	**3.36**	**3.93**	0.00	**5.19**	**4.18**	**3.85**	**3.79**	**3.69**	2.99	**2.97**	3.21	3.11	3.26	3.02
	lpm_rf	**0.19**	2.80	3.36	**3.65**	3.33	1.94	3.58	3.30	3.46	**3.81**	2.66	**4.65**	**3.99**	**3.82**	**4.85**
	dt_gbm	1.96	12.00	**10.15**	**11.37**	**10.63**	7.18	3.63	**4.52**	**2.91**	**1.78**	**1.93**	**2.29**	1.47	**2.26**	0.00
	dt_rf	**4.06**	**12.10**	8.95	10.75	10.13	**8.25**	**5.38**	2.46	2.63	1.25	1.46	1.37	**1.91**	0.00	**1.38**
higgs	lpm_gbm	**29.29**	**17.80**	11.40	6.56	3.06	2.68	**3.16**	**2.90**	**2.67**	**2.82**	2.65	1.79	2.62	2.19	1.63
	lpm_rf	26.71	17.29	**15.06**	**10.60**	**5.35**	**4.04**	2.35	2.03	1.66	1.89	**2.91**	**2.94**	**3.31**	**2.58**	**2.22**
	dt_gbm	0.00	0.00	**1.86**	0.26	0.93	0.45	-	-	-	-	-	-	-	-	-
	dt_rf	**4.04**	**1.26**	1.74	**1.32**	**1.54**	**0.91**	-	-	-	-	-	-	-	-	-
covtype.binary	lpm_gbm	76.52	**66.39**	**29.17**	**12.51**	**9.18**	**5.28**	**4.94**	**4.56**	**3.92**	**3.56**	**3.62**	**3.31**	**2.59**	**2.83**	**2.39**
	lpm_rf	**96.77**	63.38	14.36	9.61	6.79	3.94	2.93	2.81	2.96	2.84	2.31	2.26	2.00	2.43	2.22
	dt_gbm	**0.00**	**0.00**	**2.35**	1.27	1.18	1.11	0.00	0.00	0.00	-	-	-	-	-	-
	dt_rf	0.00	0.00	2.10	**2.33**	**2.44**	**2.39**	**1.84**	**2.19**	**1.65**	0.70	-	0.89	-	-	-
phishing	lpm_gbm	**0.00**	1.88	2.88	3.05	3.22	3.25	2.99	1.69	1.42	**1.45**	**1.29**	0.00	0.00	0.00	0.00
	lpm_rf	0.00	**2.14**	**3.29**	**3.22**	**3.59**	**3.79**	**3.29**	**2.05**	**1.42**	1.44	1.24	**1.23**	**1.16**	**1.26**	**1.02**
	dt_gbm	**0.00**	0.00	**0.00**	0.07	**0.39**	**0.00**	**0.28**	0.22	**0.44**	**0.23**	0.00	**0.00**	**0.00**	0.00	**0.00**
	dt_rf	0.00	**0.72**	0.00	**0.57**	0.00	-0.17	0.13	**0.48**	0.13	0.05	**0.03**	-0.03	-0.28	**0.00**	-0.16
a1a	lpm_gbm	**0.00**	2.55	7.58	8.98	8.40	8.03	8.90	8.23	8.17	7.90	5.96	7.10	6.97	6.18	5.73
	lpm_rf	0.00	**4.17**	**8.81**	**9.92**	**9.88**	**9.47**	**8.99**	**9.31**	**9.19**	**9.26**	**9.33**	**8.25**	**7.15**	**7.55**	**7.98**
	dt_gbm	**0.00**	5.54	2.39	3.84	**3.55**	2.55	1.51	2.25	4.87	-	-	-	-	-	-
	dt_rf	0.00	**6.44**	**3.36**	**5.60**	3.40	**5.94**	**6.06**	**4.97**	**4.89**	4.01	4.73	5.21	-	-	4.53
pendigits	lpm_gbm	**51.39**	**23.44**	16.18	**8.95**	**8.84**	**6.63**	4.86	**1.83**	2.27	2.16	**2.44**	2.16	**3.33**	2.97	**2.73**
	lpm_rf	46.28	22.74	**21.72**	8.80	8.47	6.29	**6.48**	1.69	**3.03**	**2.79**	2.34	**2.68**	2.70	**3.02**	0.00
	dt_gbm	14.02	**6.72**	5.11	13.14	6.42	4.20	2.46	**1.09**	**0.98**	**0.16**	-0.26	**0.00**	**0.00**	**0.00**	**0.00**
	dt_rf	**21.46**	4.18	**5.22**	**14.51**	**7.36**	**4.55**	**2.86**	0.00	0.00	0.00	**0.00**	0.00	0.00	0.00	0.00
letter	lpm_gbm	57.06	48.48	59.85	**29.76**	**36.09**	19.27	20.37	16.08	17.55	15.16	17.26	16.51	18.46	17.19	15.55
	lpm_rf	**61.06**	**65.34**	**64.26**	23.69	35.20	**26.15**	**22.10**	**20.74**	**20.91**	**20.31**	**19.28**	**21.40**	**20.77**	**19.39**	**18.18**
	dt_gbm	**0.00**	**13.98**	25.05	**33.96**	32.05	15.49	**11.17**	**0.00**	**4.26**	**3.50**	**1.99**	0.00	0.00	0.00	**0.00**
	dt_rf	0.00	12.21	**28.67**	33.47	**33.51**	**18.41**	8.10	0.00	1.84	1.21	1.31	**0.67**	**0.61**	**0.11**	-0.08
Sensorless	lpm_gbm	216.47	**257.56**	**178.31**	**117.01**	**90.70**	**83.90**	**73.50**	**65.95**	61.57	57.97	56.54	57.15	55.45	66.24	68.24
	lpm_rf	**224.18**	210.28	134.44	115.00	85.85	74.96	66.77	61.10	**66.88**	**64.65**	**69.00**	**70.09**	**72.91**	**80.14**	**82.15**
	dt_gbm	-0.01	42.42	**68.13**	44.38	**17.39**	**10.32**	1.82	**1.44**	**0.79**	**0.64**	0.41	0.12	**0.00**	-0.02	**0.34**
	dt_rf	**0.00**	**52.54**	57.10	**44.61**	16.63	6.19	**2.19**	0.96	0.51	0.00	**0.48**	**0.33**	0.00	**0.00**	0.10
senseit_aco	lpm_gbm	173.71	170.68	63.95	**44.20**	33.49	22.99	19.14	13.50	10.29	7.59	6.26	5.92	**5.30**	4.89	4.32
	lpm_rf	**177.67**	**181.26**	**79.86**	42.86	**37.60**	**28.80**	**23.75**	**19.06**	**13.91**	**10.74**	**8.48**	**6.09**	5.20	**5.32**	**4.62**
	dt_gbm	14.89	0.00	**3.71**	2.32	**4.85**	0.81	**0.00**	-	-	-	-	-	-	-	-
	dt_rf	**20.03**	**2.54**	3.64	**5.91**	3.34	**2.63**	0.00	0.00	-	-	-	-	-	-	-
senseit_sei	lpm_gbm	160.59	**65.27**	23.44	10.48	6.76	4.86	4.82	4.46	4.79	4.12	4.54	**5.17**	3.91	4.21	**4.46**
	lpm_rf	**165.98**	63.72	**31.58**	**14.94**	**9.07**	**5.79**	**4.95**	**5.07**	**5.24**	**4.70**	**4.60**	3.74	**4.30**	**4.35**	4.35
	dt_gbm	**2.66**	**1.01**	**3.49**	**2.29**	**0.95**	**1.30**	**1.37**	0.00	-	-	-	-	-	-	-
	dt_rf	2.33	0.00	3.36	1.65	0.87	0.00	-1.23	-	-	-	-	-	-	-	-
covtype	lpm_gbm	**36.87**	**49.24**	**12.78**	**11.21**	7.84	7.15	7.15	8.07	7.70	8.25	**10.94**	**8.35**	4.37	8.77	5.84
	lpm_rf	32.15	39.49	10.49	8.53	**8.11**	**8.59**	**9.61**	**11.99**	**11.22**	**9.91**	8.47	8.16	**10.34**	**13.76**	**12.92**
	dt_gbm	342.27	92.85	43.23	**20.04**	8.14	**8.05**	**5.67**	3.26	**4.92**	**3.52**	**2.72**	0.00	**0.00**	**0.00**	**1.74**
	dt_rf	**354.45**	**98.94**	**50.87**	14.10	**9.46**	7.38	4.76	**4.20**	0.94	1.81	2.30	**0.71**	-0.37	0.00	0.00
connect-4	lpm_gbm	**37.62**	11.66	12.01	6.84	5.68	6.82	4.58	2.10	3.82	3.21	**3.02**	**3.64**	**2.32**	**2.97**	**3.40**
	lpm_rf	33.77	**12.99**	**17.60**	**14.66**	**15.91**	**10.73**	**6.38**	**5.35**	**7.07**	**6.98**	2.84	3.14	2.09	2.52	2.46
	dt_gbm	89.33	**29.23**	20.20	**12.10**	9.73	9.88	7.82	7.43	0.57	4.61	1.08	**3.35**	2.23	**1.15**	**1.55**
	dt_rf	**113.71**	21.91	**20.52**	11.23	**16.86**	**10.96**	**10.64**	**9.11**	**6.51**	**5.88**	**6.76**	2.16	**2.97**	0.61	0.00

We also perform a Wilcoxon signed-rank test to measure statistical significance of the $\delta F1$ scores here. The setup is identical to that in Section 4.3.1.7: one-sided test, paired over datasets, performed separately for ranges of normalized bin sizes. Scores of $\delta F1 = 0$ are split equally between positive and negative ranks. A key difference here is that $\delta F1 \in (-\infty, \infty)$.

The results are shown in Figure 5.1.

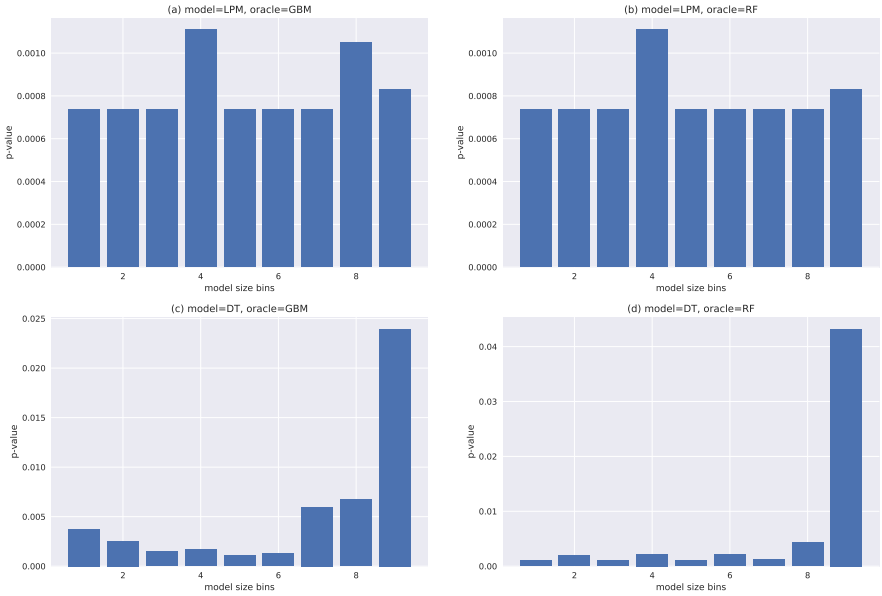


Figure 5.1: These plots show the *p-values* for the Wilcoxon signed-rank test, with the null hypothesis H_0: using the oracle does *not* produce better F1 test scores. The bin boundaries are selected using the *Sturges* rule (Sturges, 1926). Low *p-values* favor our algorithm.

We note that *p-values* are small here as well. In comparison to Figure 4.10, for decision trees, the *p-values* are typically larger. This is to be expected since we now have cases where $\delta F1 < 0$. We don't see this for LPMs possibly because they are not constructed all the way up to their maximum possible size, i.e., when number of non-zero coefficients is equal to the number of features, and so the impact of our technique is significant irrespective of how improvements are calculated.

5.2 Summary of Contributions

This dissertation proposes two techniques for building compact models, i.e., models that minimize the trade-off between size and accuracy. Such techniques find use in a growing number of applications that require interpretability, where small accurate models are typically preferred.

We demonstrated that it is possible to do so by learning a sampling distribution over the training data. In developing our techniques, we make the following novel contributions:

1. We show that the training distribution that leads to the highest test accuracy is, in general, different from the test distribution at small model sizes. We also clearly show that the ideal training distribution progressively grows closer to the test distribution as model size increases.

2. We provide two techniques to learn such a distribution. These are empirically validated across a diverse set of data and models.

3. Both approaches formulate the problem of identifying this distribution as an optimization problem in a way that the number of optimization variables does not depend on the dimensionality of data - the density tree and oracle based approaches are specified by a fixed set of eight and seven variables respectively. This makes the techniques scalable.

4. It is shown that reasonable defaults exist for these parameters which make it possible to achieve good results without hyperparameter tuning. This augments their practical value.

5. Both approaches are frameworks in the sense that the key optimization step only identifies certain attributes that an optimizer must possess. This makes it convenient to replace the optimizer used in our implementation with better optimizers as they become available.

6. The density tree based approach introduces an innovative tool to control the extent of class boundary information available to a learner.

7. The oracle based approach is shown to be highly flexible in terms of both the model family of the oracle that may be used and the difference in its feature space wrt to that of the interpretable model.

The oracle based technique has also been publicly released as a Python library (Ghose, 2020).

5.3 Future Directions

We present potential avenues of future work in this section. This is divided into two groups: (a) algorithmic improvements and additional applications (b) research directions. While most suggestions here apply to both the density tree based and oracle based approaches, those that apply exclusively to one are explicitly indicated.

We start with the former group. Our algorithms are reasonably abstracted from low level details, which enables various extensions like the following;

1. **Smoothing**: Applies to density trees. We had hinted at alternatives to Laplace smoothing in Section 3.2.5. 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 $P \times S^k, k \in \mathbb{Z}_{\geq 0}$ 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 the similarity is; this can replace λ 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. This is not a challenge for the oracle based approach since it can work across disparate feature spaces (as demonstrated in Section 4.3.3.1).

   For the density tree based approach, there are a couple of ways to handle data with such features:

   (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

Table 5.2: Classification Results with GBMs. Both $F1_{new}$ and $\delta F1$ are shown. A reproduction of Table 3.4.

		boosting rounds =	1	2	3	4	5	6	7	8	9	10
datasets	max depth	score type										
Sensorless	2	$F1_{new}$	0.76	0.77	0.78	0.80	0.80	0.80	0.81	0.81	0.81	0.81
		$\delta F1$	3.19	3.35	3.10	5.05	4.12	1.75	3.21	1.96	1.90	2.43
	5	$F1_{new}$	0.91	0.92	0.93	0.94	0.94	0.94	0.94	0.95	0.95	0.95
		$\delta F1$	0.29	0.25	0.16	0.40	0.00	0.18	0.36	0.30	0.00	0.26
senseit_aco	2	$F1_{new}$	0.22	0.24	0.31	0.37	0.52	0.59	0.61	0.62	0.63	0.63
		$\delta F1$	0.00	6.81	41.41	67.44	69.29	9.39	6.83	4.70	2.33	1.10
	5	$F1_{new}$	0.22	0.30	0.42	0.51	0.58	0.62	0.65	0.66	0.67	0.68
		$\delta F1$	0.00	36.80	85.44	46.66	9.72	2.91	1.17	0.34	0.39	0.40
senseit_sei	2	$F1_{new}$	0.60	0.60	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.62
		$\delta F1$	171.08	171.28	173.05	174.66	173.47	165.00	78.73	48.52	24.61	17.63
	5	$F1_{new}$	0.62	0.64	0.64	0.64	0.64	0.65	0.64	0.64	0.65	0.66
		$\delta F1$	180.46	185.59	186.24	181.13	64.66	28.13	11.30	3.11	1.37	0.59

See Table A.1 for complete data.

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 $train_{\mathcal{F}}()$ 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 $train_{\mathcal{F}}()$ function, this transformation would be modeled as a part of it.

3. **Model compression**: An interesting possible application of our techniques is model compression[1]. This is possible with either technique. We demonstrate the idea with the GBM analysis data - see Table 5.2- for density trees.

   Consider the column $boosting\ round = 1$ for the *senseit_sei* dataset in Table 5.2. 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, $F1_{baseline} =$

[1] This was briefly discussed in Section 4.4

$F1_{new}/(1+\delta F1/100) = 0.62/(1+1.8046) = 0.22$. 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 η_e (subscript "e" for "equivalent") in relation to a size $\eta > \eta_e$ such that:

$$accuracy(train_{\mathcal{F}}(p^*_{\eta_e}, \eta_e), p) \approx accuracy(train_{\mathcal{F}}(p, \eta), p) \tag{5.1}$$

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 algorithms 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 algorithms useful for applications like *cohort analysis*, common in marketing studies, where the objective is to study the behaviour of a segment - say, based on age and income - within a larger population. Our algorithms are especially appropriate since traditionally such analyses have emphasized interpretability.

5. **Multidimensional size**: The notion of model size is not restricted to being a scalar quantity. We demonstrated this for the oracle based approach in Section 4.3.3.3, but this also applies to the density tree based approach.

6. **Using different optimizers**: As mentioned earlier, both our algorithms are frameworks, and as better optimizers become available, our algorithms may be easily improved by using them to re-implement $suggest()$ in Algorithm 7 and Algorithm 9. This is especially enabled by the conscious choice of using a fixed set of optimization variables.

   We show examples in Figure 5.2, where accuracies against number of iterations are shown for the datasets (a) *covtype* and (b) *letter*. The model used is LPM - size 2 and 3 respectively - and the oracle is GBM. This setup uses *ten trials*, each with an optimizer budget of 1000 iterations. The datasets contain 2000 instances, and the splits are the same as in our other experiments, i.e., $D_{train} : D_{val} : D_{test} :: 60 : 20 : 20$. The F1 macro score on D_{train} is shown since the optimizer's objective

function is defined on this split, making it a good choice for tracking progress.

In addition to TPE (denoted as "*hyperopt*", for the name of the library), optimizers *LIPO* (Malherbe and Vayatis, 2017) and *pySOT* (Eriksson *et al.*, 2019) are used. The *pySOT* library supports multiple search strategies; we show results for *Stochastic Radial Basis Function* (denoted as "pysot(srbf)") (Regis and Shoemaker, 2007, 2009) and *Dynamic Coordinate Search* (denoted as "pysot(dycors)") (Regis and Shoemaker, 2013).

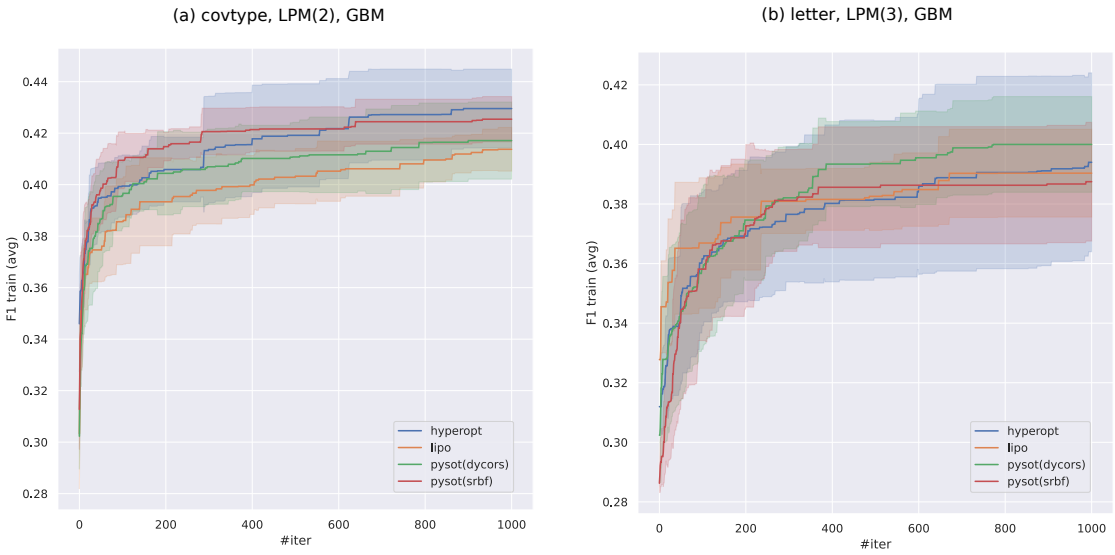


Figure 5.2: Use of different optimizers is illustrated; see text for details. The plot titles have the following format: *dataset, model family (model size), oracle family*.

Note the different behavior of the optimizers: in Figure 5.2(a), while *hyperopt* eventually does better, stopping before the 600^{th} iteration favors *pysot(srbf)*. In Figure 5.2(b), *pysot(dycors)* nearly entirely dominates *hyperopt*.

The strong influence of optimizers on convergence in our setup make them a meaningful area of study.

7. **Fast Sampling**: Even though the data that we apply the IBMM to is univariate, the sampling can still be expensive, especially when there are multiple $Beta$ components (we present some representative running times in Section A.13). This suggests an obvious direction for improvement: use a faster sampling procedure. One way to attain this would be to use a distribution like the *Kumaraswamy* distribution (briefly discussed in Section 3.4), which leads to faster sampling due to the simple nature of its *Cumulative Distribution Function*.

From the perspective of research, we find the following future directions appealing:

1. **Applicability to Deep Learning**: A significant extension would be to test if the principle of learning the training distribution to reduce model size applies also to Deep Learning (DL). A direct application of our techniques is infeasible since they train multiple models, which can be prohibitively expensive for DL. Alternative optimization strategies need to be explored, such as those based on gradient information (see next point) or a hybrid of early-stopping and BO (Falkner *et al.*, 2018).

   This has a related benefit. The primary reason why experiments here are restricted to tabular datasets, and do not include unstructured datasets such as *ImageNet* (Deng *et al.*, 2009), is the infeasibility of using DL. The proposed extension will also open up paths to analyzing such datasets.

2. **Differentiable optimization:** We might think of our techniques as learning a sampling distribution $p'(x_i)$ indirectly on the input space:

   - via $p(u_{M_O}(x_i))$, the distribution over uncertainty scores, in the case of the oracle based approach, and
   - using the depth distribution and the pmfs over nodes, in the case of density trees.

   An alternative view might be to directly learn instance weights w_i instead, where $w_i = p'(x_i)$. This approach clearly suffers from challenges in scaling - there are as many weights as training instances. However, recent work suggests that for *differentiable* model losses, this problem might be efficiently solved by formulating it as a *bi-level optimization* problem (Pedregosa, 2016; Lorraine *et al.*, 2020); which makes this a feasible direction to explore. The expected benefit is this might be faster[2], at least for moderately sized datasets.

   This approach brings its own challenges that future work would need to consider: (a) since gradient information is required, the loss function must be known; therefore this approach is not model-agnostic (b) even if an automatic differentiation

[2] Pedregosa (2016) compares this approach against BO for the task of hyperparameter tuning; these numbers should be assumed to be indicative only, since the BO algorithm used is not TPE.

framework is used, such as *JAX* (Bradbury *et al.*, 2018), to generalize to unseen loss functions, model families like DTs remain out of scope since their loss isn't differentiable[3].

3. **Evaluating Active Learners:** The standard way to evaluate active learning algorithms is to evaluate model accuracy against the number of labeled training data instances. It is interesting to consider an alternative approach: for a given budget of labeled instances, measure the divergence between the sampling distribution our method learns[4] and the one that an active learner proposes for labeling. Such an analysis is insightful since it can indicate precisely which points an active learner is *supposed* to label.

4. **Ensemble of small models:** We noted that improvements from our techniques diminish as model size grows. For larger model sizes, a possible direction to explore might be to "chain" together multiple small models. This is similar to gradient boosting, and it would be especially informative to compare the two approaches.

5. **Theoretical framework:** An obvious question to ask is if our observations around the impact of training distribution on accuracy may be theoretically explained. There is some recent work in the area of KD that might serve as fruitful starting points: (a) Dao *et al.* (2021) provide theoretical tools to analyze distillation by treating it as a semi-parameteric inference problem (b) Menon *et al.* (2021) propose a connection between the effectiveness of a teacher and its ability to approximate Bayes class-probabilities. (c) study of sample re-weighting on the effectiveness of distillation (Zhang *et al.*, 2021; Lu *et al.*, 2021).

6. **Acceptable difference between the capabilities of an interpretable model and an oracle:** While the example in Section 4.3.3.1 shows that the oracle based approach can work across different feature spaces, it is also easy to think of pathological examples where it's likely to fail. Consider the mapping $f : \mathcal{X}' \rightarrow \mathcal{X}''$, between the feature vector $x' \in \mathcal{X}'$ for an instance in the oracle feature space and its counterpart $x'' \in \mathcal{X}''$ in the feature space of the interpretable model. For a

[3]There is recent interesting research in this area as well, see Bolte *et al.* (2021)

[4]As in Section 4.3.3.2; but the density tree based approach may also be used to produce similar output.

dataset, if the number of *unique* vectors x' is much larger than x'', it is easy to see that information is lost between the two models. It would be useful to determine the properties of the mapping f that decide if our technique can produce useful results accounting for such loss.

7. **Quantifying importance of data:** It would also be interesting to explore the connection between the sample of a given size our method finds (Section 4.3.3.2) and the *data Shapley value* (Ghorbani and Zou, 2019): a per-instance value quantifying the contribution of an instance to predictor accuracy. Some questions of interest are: (a) does an instance that has a high sampling probability across a range of sample sizes, per our method, also receive a high data Shapley value? (b) if there is indeed a correspondence between the two techniques, what algorithmic ideas may be borrowed from one technique to another?

APPENDIX A

APPENDIX

A.1 Implementation Details

Setting the lower bound of the N_s parameter to ensure statistical significance is not sufficient in itself. Since our sample comes from both the density trees and the original training data, we must ensure these samples lead to statistically significant results *individually*.

In order to do so the our implementation internally adjusts the quantity p_o. Recall that $p_o \in [0, 1]$. A low value of p_o can result in a small sample of size $p_o N_s$ from the original training data, while a high value of p_o might result in a small sample of size $(1 - p_o)N_s$ from the density trees. Interestingly however, $p_o = 0$ and $p_o = 1$ should be allowed as valid values, since the sample is then drawn from only one of the sources and is therefore not small!

The adjustment we make is shown in Figure A.1. The x-axis shows the value of p_o proposed by the optimizer, while the y-axis shows the adjusted version presented to the sampler.

Below a user specified threshold for p_o, it is adjusted to $p_o = 0$. Beyond a certain user specified threshold, it is adjusted to $p_o = 1$. The lack of smoothness or differentiability of the adjustment does not impact our optimization, since a BO would construct its version of the objective function anyway.

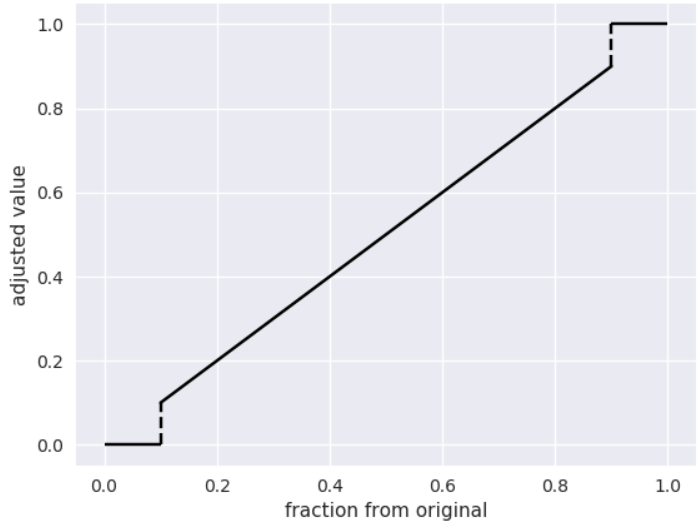


Figure A.1: Adjustments to p_o.

A.2 GBM Results

Table A.1 represents the improvements seen using GBMs where we have $max\_depth = 2$ or $max\_depth = 5$ for the base classifier trees. This is an expanded version of data presented in Table 3.4. Note here that much like DTs and LPMs, we see the largest $\delta F1$ values typically for relatively smaller model sizes.

Table A.1: $F1_{new}$ and $\delta F1$ scores are shown for GBM models with $max\_depth = 2$ and $max\_depth = 5$. All scores are averaged over five runs. Underlined entries denote the best improvement for a dataset.

		boosting rounds =	1	2	3	4	5	6	7	8	9	10
datasets	max depth	score type										
cod-rna	2	$F1$	0.42	0.64	0.69	0.70	0.70	0.71	0.74	0.78	0.82	0.84
		$\delta F1$	4.78	59.01	72.16	38.09	3.58	3.64	6.68	12.46	16.91	5.22
	5	$F1$	0.44	0.78	0.83	0.84	0.86	0.87	0.88	0.88	0.89	0.89
		$\delta F1$	11.24	95.76	35.22	2.11	0.63	0.45	0.61	0.20	0.19	0.33
ijcnn1	2	$F1$	0.71	0.71	0.71	0.71	0.72	0.72	0.71	0.71	0.71	0.72
		$\delta F1$	9.55	9.77	8.11	7.28	8.28	6.98	4.70	5.87	5.18	7.15
	5	$F1$	0.77	0.77	0.77	0.78	0.78	0.79	0.79	0.79	0.79	0.80
		$\delta F1$	4.76	3.92	3.56	3.44	3.31	3.97	3.85	3.87	2.04	3.05
higgs	2	$F1$	0.61	0.62	0.63	0.63	0.63	0.63	0.63	0.64	0.64	0.64
		$\delta F1$	60.32	32.30	14.09	6.76	3.02	2.28	2.23	1.32	1.79	1.85
	5	$F1$	0.62	0.64	0.64	0.65	0.66	0.66	0.67	0.67	0.67	0.68
		$\delta F1$	29.78	11.48	3.25	0.96	1.32	1.13	0.06	1.07	0.77	0.48
covtype.binary	2	$F1$	0.73	0.73	0.73	0.73	0.74	0.74	0.74	0.74	0.74	0.74
		$\delta F1$	0.14	0.55	0.51	0.67	0.75	0.92	0.64	0.70	0.46	0.74
	5	$F1$	0.76	0.76	0.77	0.76	0.76	0.77	0.77	0.77	0.77	0.77
		$\delta F1$	1.15	0.75	0.77	0.49	0.99	0.76	0.50	0.07	0.15	0.06
phishing	2	$F1$	0.90	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91	0.91
		$\delta F1$	152.30	5.02	0.29	0.02	0.01	0.03	0.36	0.40	0.32	0.21
	5	$F1$	0.92	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.93	0.94
		$\delta F1$	156.68	1.87	1.08	1.26	0.89	0.72	0.56	0.56	0.67	0.43
a1a	2	$F1$	0.71	0.72	0.72	0.72	0.73	0.73	0.73	0.73	0.73	0.74
		$\delta F1$	4.69	3.95	4.46	5.87	5.86	5.86	4.90	4.87	5.05	5.03
	5	$F1$	0.74	0.74	0.75	0.75	0.74	0.74	0.75	0.75	0.75	0.76
		$\delta F1$	3.88	2.57	2.86	3.72	2.77	2.73	3.40	3.05	3.63	3.56
pendigits	2	$F1$	0.76	0.80	0.81	0.82	0.82	0.83	0.83	0.84	0.84	0.84
		$\delta F1$	3.73	1.95	1.46	0.70	0.91	0.64	0.40	0.47	0.47	0.67
	5	$F1$	0.92	0.94	0.94	0.95	0.95	0.95	0.95	0.96	0.96	0.96
		$\delta F1$	0.17	0.21	0.17	0.17	0.05	0.02	0.11	0.00	0.02	0.07
letter	2	$F1$	0.53	0.58	0.59	0.61	0.61	0.62	0.63	0.63	0.63	0.64
		$\delta F1$	3.04	0.97	1.45	0.97	2.05	0.68	1.16	0.41	0.79	0.43
	5	$F1$	0.71	0.76	0.77	0.78	0.79	0.80	0.80	0.80	0.81	0.82
		$\delta F1$	1.03	0.11	0.00	0.03	0.00	0.00	0.00	0.10	0.00	0.00
Sensorless	2	$F1$	0.76	0.77	0.78	0.80	0.80	0.80	0.81	0.81	0.81	0.81
		$\delta F1$	3.19	3.35	3.10	5.05	4.12	1.75	3.21	1.96	1.90	2.43
	5	$F1$	0.91	0.92	0.93	0.94	0.94	0.94	0.94	0.95	0.95	0.95
		$\delta F1$	0.29	0.25	0.16	0.40	0.00	0.18	0.36	0.30	0.00	0.26
senseit_aco	2	$F1$	0.22	0.24	0.31	0.37	0.52	0.59	0.61	0.62	0.63	0.63
		$\delta F1$	0.00	6.81	41.41	67.44	69.29	9.39	6.83	4.70	2.33	1.10
	5	$F1$	0.22	0.30	0.42	0.51	0.58	0.62	0.65	0.66	0.67	0.68
		$\delta F1$	0.00	36.80	85.44	46.66	9.72	2.91	1.17	0.34	0.39	0.40
senseit_sei	2	$F1$	0.60	0.60	0.61	0.61	0.61	0.61	0.61	0.61	0.61	0.62
		$\delta F1$	171.08	171.28	173.05	174.66	173.47	165.00	78.73	48.52	24.61	17.63
	5	$F1$	0.62	0.64	0.64	0.64	0.64	0.65	0.64	0.64	0.65	0.66
		$\delta F1$	180.46	185.59	186.24	181.13	64.66	28.13	11.30	3.11	1.37	0.59
covtype	2	$F1$	0.41	0.42	0.41	0.41	0.41	0.40	0.41	0.40	0.40	0.40
		$\delta F1$	12.04	17.93	17.60	11.32	12.42	16.22	16.03	19.05	14.57	19.58
	5	$F1$	0.47	0.48	0.48	0.48	0.49	0.49	0.50	0.50	0.49	0.50
		$\delta F1$	0.64	2.55	0.17	1.17	1.50	0.52	1.16	0.69	0.43	1.32
connect-4	2	$F1$	0.43	0.44	0.45	0.45	0.47	0.46	0.47	0.47	0.47	0.47
		$\delta F1$	21.59	19.36	20.89	22.65	23.41	14.61	14.13	19.09	9.78	9.98
	5	$F1$	0.48	0.49	0.50	0.51	0.51	0.52	0.52	0.53	0.53	0.53
		$\delta F1$	4.30	5.26	2.68	4.72	1.89	3.43	4.69	2.89	5.87	1.92

A.3 Harmonic Numbers

The N^{th} harmonic number is defined as:

$$H_N = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{N} = \sum_{k=1}^{N} \frac{1}{k} \tag{A.1}$$

Clearly $H_N \propto N$, since increasing N adds positive terms to H_N. Figure A.2 shows the relationship of H_N and N for $N = 1, 2, ..., 100$

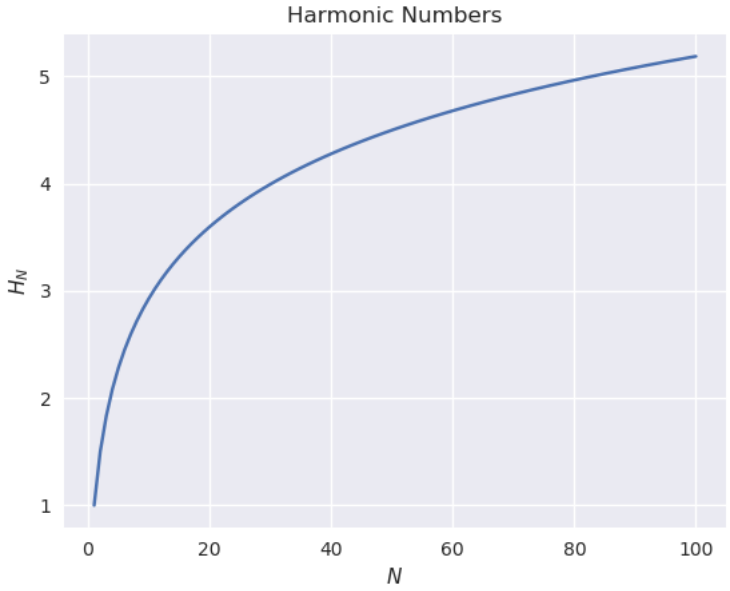


Figure A.2: Variation of H_N with increasing N.

A.4 Supervised Uncertainty Sampling

In Algorithm 10:

1. The loop in lines 5-11 runs $\lceil |D_{train}|/b \rceil$ times, where every iteration adds the b most uncertain points to the current training dataset D_t. If b doesn't evenly divide $|D_{train}|$, the last iteration picks all remaining points.

2. In our implementation, $u_{M_\mathcal{O}}(x_i)$ in line 6 is precomputed and stored as a lookup table to reduce execution time.

3. In our experiments, we use a batch size $b = 10$. Note that this gives us optimal

Algorithm 10: Supervised Uncertainty Sampling

Data: Dataset D, model size η, $train_{\mathcal{O},h}()$, $train_{\mathcal{I},g}()$, batch size b
Result: Test set accuracy s_{test}, and interpretable model M^*

1 Create stratified splits $D_{train}, D_{val}, D_{test}$ from D
2 $M_O \leftarrow train_{\mathcal{O},h}(D_{train}, *)$
3 $I_{remaining} \leftarrow \{1, 2, ..., |D_{train}|\}$ be an index set of D_{train}
4 $I_{current} \leftarrow \{\}$
5 **for** $t \leftarrow 1$ **to** $\lceil |D_{train}|/b \rceil$ **do**
6 $\quad I_U \leftarrow$ set of top b entries from $I_{remaining}$, based on $u_{M_{\mathcal{O}}}(x_i), i \in I_{remaining}$
7 $\quad I_{remaining} \leftarrow I_{remaining} - I_U$
8 $\quad I_{current} \leftarrow I_{current} \cup I_U$
9 $\quad D_t \leftarrow \{D_{train,i} | i \in I_{current}\}$
10 $\quad M_t \leftarrow train_{\mathcal{I},g}(D_t, \eta)$
11 $\quad s_t \leftarrow accuracy(M_t, D_{val})$
12 **end**
13 $t^* \leftarrow \arg\max_t \{s_1, s_2, ..., s_{T-1}, s_T\}$
14 $M^* \leftarrow M_{t^*}$
15 $s_{test} \leftarrow accuracy(M^*, D_{test})$
16 **return** s_{test}, M^*

models as per Algorithm 10, for all batch sizes of the form $10k$, where $k \in \{1, 2, ..., \lfloor |D_{train}|/10 \rfloor\}$

The modified algorithm is a **significantly** more powerful version compared to the ones typically used in Active Learning setups, due to the following reasons:

1. We do not assume a cost for procuring or applying the oracle, which contrasts with the typical active learning setup. Thus, our oracle utilizes complete label information and our model has access to reliable uncertainty scores; this avoids the sample bias discussed in Section A.5 (visualized in Figure A.3).

2. Since we have complete label information, we have a validation set D_{val} available to us. In active learning, a validation set would be created from within the current labelled subset of data, which often makes it statistically insignificant or non-representative of the true distribution, especially at early iterations.

3. We do not have to estimate how many times the loop in lines 5-11 must run - this is executed till all data from D_{train} has been used up to train the model. Estimating the number of iterations is required when performing active learning since every iteration incurs a cost - that of calling the oracle to compute I_U. Consequently,

here, we have the liberty of being able to *pick* the best model based on a validation set D_{val}.

A.5 Pitfalls of Simple Uncertainty Sampling

In active learning, the goal is to learn a model when we are given none or few of the labels of our training data, but we are allowed to query for labels for a cost Settles (2009). This is helpful in scenarios where acquiring labels is expensive, and instead of asking for labels for a random 1000 points to train on, we could ask for the labels of a specific 200 points, chosen in some manner, that leads to comparable model accuracy. *Uncertainty Sampling* was introduced in Lewis and Gale (1994) to solve this problem. We begin by requesting the labels of small batch of randomly sampled points - this is the labelled subset of the data. The following steps are then repeated:

1. Construct a classifier on the current labelled subset.
2. Use it to provide uncertainty scores for unlabelled points in the data, and then request labels for the top b (the precise value of b may be task specific) uncertain points. These now become part of the labelled subset.

Although intuitive, this approach was shown to suffer from sample bias Dasgupta and Hsu (2008); Dasgupta (2011). We illustrate this in Figure A.3.

We consider the simple case where our data is located on a line, has two labels (denoted by red and green in the figure) and most of the data is located at the extremes of the line segment, as shown by blocks P and Q, each of which represent 45% of the overall data. Here, learning a classifier is equivalent to identifying a single point on the line, and the classification rule is we assign labels green and red, to left and right of this point, respectively. B and C show two possible classifiers.

In the active learning setup, we observe only the points but not their labels. To use uncertainty sampling, we pick our first small batch of points randomly and query their labels. Because of the distribution of the data, its highly likely that we would only see points from P and Q. The best classifier on this sample is C, which is midway

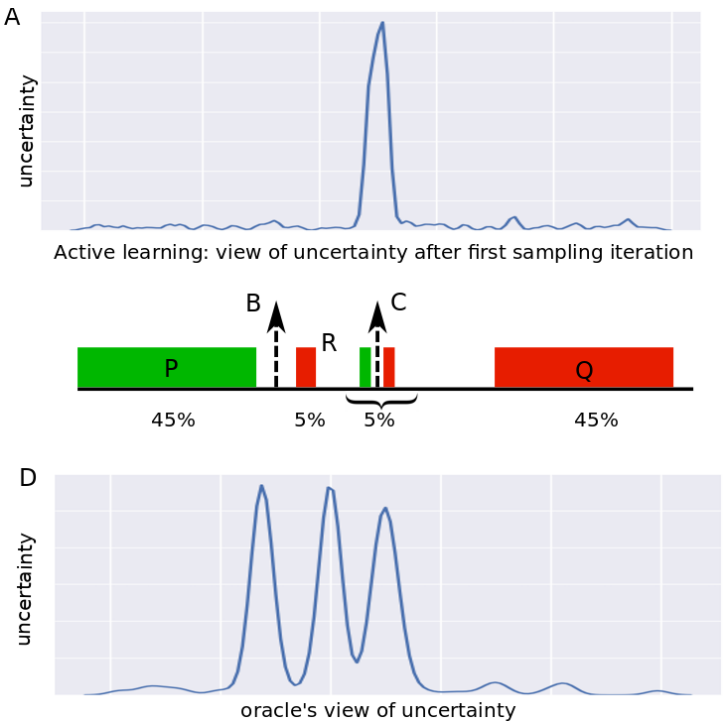


Figure A.3: Uncertainty estimates from classifier after first iteration. Smaller boundaries are missed since the sample predominantly comes from P and Q.

between P and Q. Plot A shows what the uncertainty across the input space looks like according to C. In the next iteration, we will sample close to C, since that's where the highest uncertainties are, and the new classifier constructed would again be at location C. Subsequent iterations would further reinforce the belief that C is the only class boundary. Here, the classification error of C is 5%, but the optimal classifier is B, with an error of 2.5%, which uncertainty sampling fails to discover. The key problem here is we may never see some boundaries, like those defined by R, because of the combination of initial sample bias and subsequent aggressive sampling.

This problem does not affect us since the oracle has access to the complete training data. Plot D shows the uncertainty distribution as per the oracle. However, as our results show, even with its complete view of uncertainty landscape, simple uncertainty sampling is not optimal.

A.6 Comparison of Uncertainty Distributions

It is instructive to look at some specific adjusted IBMMs in the context of the relative performance of techniques. Figure A.4 shows the plots from Figure 4.14 annotated with SDI scores. These are for $LPMs$ using GBM as the oracle.

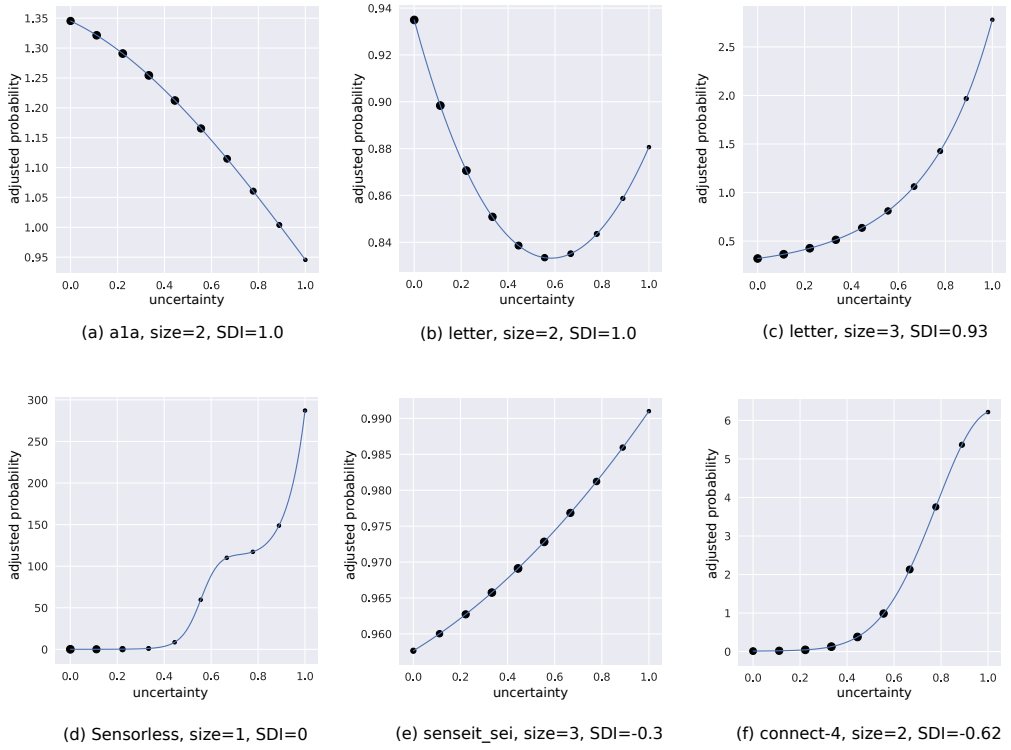


Figure A.4: Examples of adjusted distributions are shown, and the SDI scores, measured against supervised uncertainty sampling, are mentioned. The plots in the top row are the same as in Figure 4.14. The top row - (a), (b), (c) - shows instances where our technique performed relatively better, and the bottom row shows cases where uncertainty sampling was competitive - (d) - or better - (e), (f).

The top row - (a), (b), (c) in Figure A.4 - shows instances where our technique did much better ($SDI > 0$); it would seem that these are cases where sampling exclusively at high uncertainties is not an optimal distribution. Figure A.4(d) shows a case where the optimal distribution is composed exclusively of high uncertainty points - so its not surprising that uncertainty sampling is at par with our technique ($SDI = 0$). (e) and (f) show similar trends.

While these plots are helpful in developing intuition for the underlying process, we would like to add the caveat that they are not conclusive in isolation. An example of this is (c) - it is not clear why uncertainty sampling does so poorly here. Possibly, instances with low uncertainties need to be sampled in a very specific manner that cannot be

approximated by selecting the top n uncertain points, for any n.

A.7 Flattening of the Uncertainty Distribution

Algorithm 11 details the flattening process mentioned in Section 4.2.5.

Algorithm 11: Flatten distribution of uncertainty scores $\{u(x_1), u(x_2), ..., u(x_N)\}$

Data: $\{u(x_1), u(x_2), ..., u(x_N)\}$, number of bins B
Result: $\{u'(x_1), u'(x_2), ..., u'(x_N)\}$
1 $bin\_size \leftarrow \lceil N/B \rceil, bin\_range \leftarrow 1/B$
2 $bin\_min \leftarrow [\,], bin\_max \leftarrow [\,]$
3 Let $sortedIndex(i) \in \{1, 2, ..., N\}$ be the index of $u(x_i)$ in the sequence of scores ordered by non-decreasing values.
4 **for** $j \leftarrow 1$ **to** B **do**
5 $\quad bin\_min[j] \leftarrow \min\{u(x_i) | i \in \{1, 2, ..., N\} \wedge sortedIndex(i) = j\}$
6 $\quad bin\_max[j] \leftarrow \max\{u(x_i) | i \in \{1, 2, ..., N\} \wedge sortedIndex(i) = j\}$
7 **end**
8 **for** $i \leftarrow 1$ **to** N **do**
9 $\quad j \leftarrow sortedIndex(i)$
10 $\quad bin\_num \leftarrow \lceil j/bin\_size \rceil$
11 $\quad boundary\_low \leftarrow (bin\_num - 1) \times bin\_range + \delta$
12 $\quad boundary\_high \leftarrow bin\_num \times bin\_range - \delta$
13 $\quad u'(x_i) \leftarrow low + \frac{u(x_i) - bin\_min[j]}{bin\_max[j] - bin\_min[j]} \times (boundary\_high - boundary\_low)$
14 **end**
15 **return** $\{u'(x_1), u'(x_2), ..., u'(x_N)\}$

In lines 11 and 12 of Algorithm 11, we offset bin boundary limits by a small positive value δ to avoid assignment conflicts across adjacent bins at their boundaries.

This algorithm produces a transformation that looks like the uniform distribution. We prefer the likeness to the uniform distribution since it makes all regions within the interval $[0, 1]$ equally easy to discover.

A.8 Uncertainty Distribution for DT

The uncertainty distributions learned when using a DT with different oracles are shown in Figure A.5. The first row shows visualizes the aggregation of the IBMMs that were

learned, while the second row shows them adjusted with the uncertainty distribution from the oracle. These are analogues of the LPM plots in Figure 4.12 and Figure 4.13.

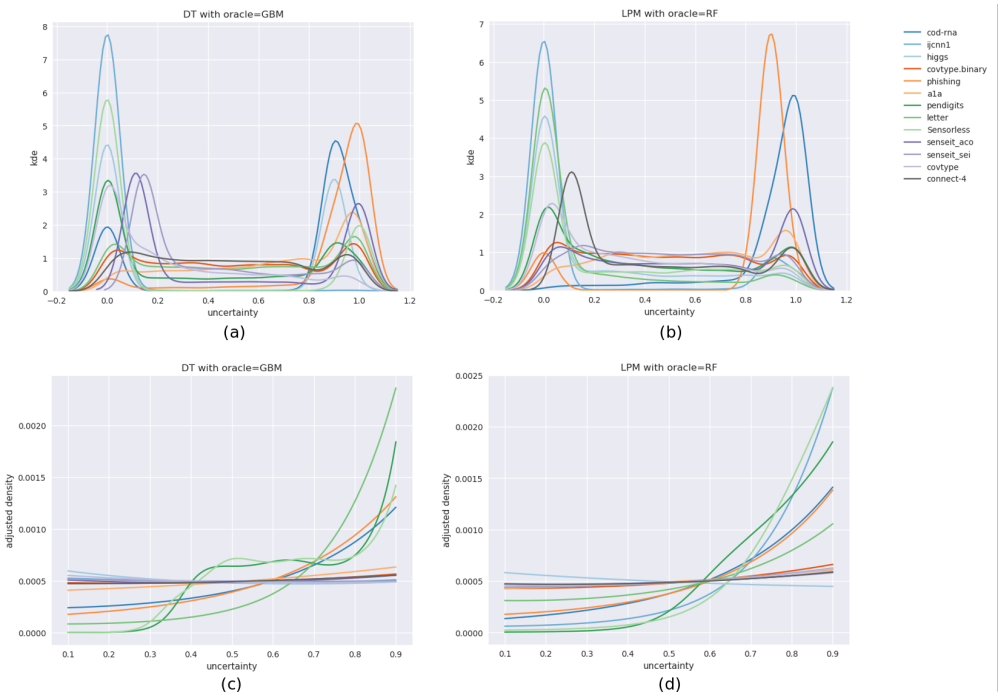


Figure A.5: The aggregated IBMMs visualized when using a DT as our interpretable model. The top row shows the aggregated IBMMs for different oracles: GBM (left) and RF (right). The bottom row visualizes the IBMMs adjusted for the uncertainty distribution.

The patterns we observe here are similar to what we saw for LPMs:

1. Top-row: the IBMMs seem to prefer both low and high uncertainty regions.
2. Bottom-row: when adjusted with the oracle's uncertainty distribution, there is sampling across the entire range of uncertainty values, with slight/occasional preference for higher uncertainties.

A.9 Compaction Profiles

Figure A.6 shows the compaction profiles for all model-oracle combinations. These are discussed in Section 4.3.1.6, in reference to Figure 4.9.

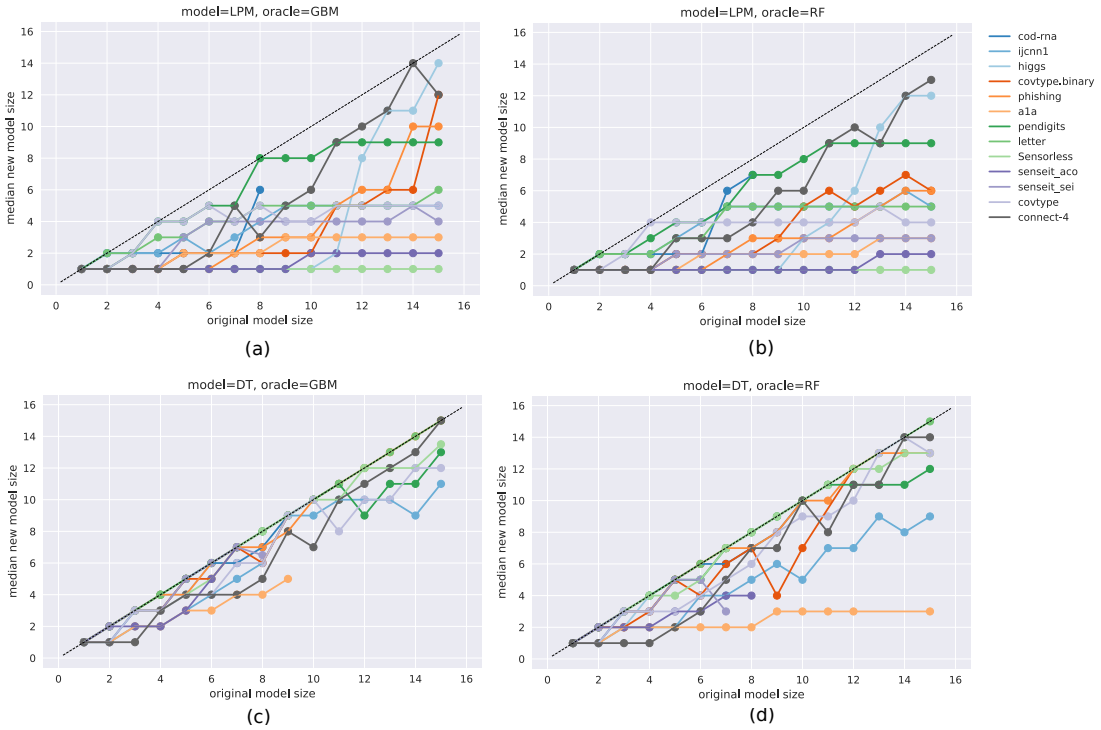


Figure A.6: For different combinations of models and oracles: $\{LPM, DT\} \times \{GBM, RF\}$, these plots show the size of an improved model (y-axis), that may replace a traditionally trained model of a given size (x-axis). A model is considered as a replacement for another if its accuracy is at least as high as the latter.

A.10 Distributions for Different Model Sizes

Figure A.7 shows the IBMMs learned over uncertainties for individual model sizes of the LPM, with GBM as the oracle,. These are *not* adjusted with the density of the uncertainty distribution. The plot shows them for the datasets (a) `covtype.binary` and `Sensorless`. We observe that the unified IBMM weighted by improvements, shown in Figure 4.12, are indicative of the individual distributions in this Figure A.7.

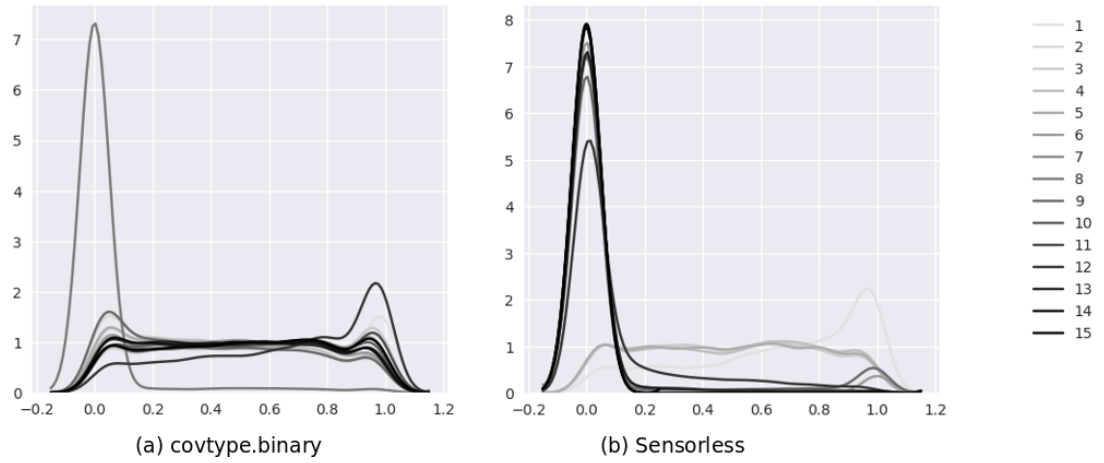


Figure A.7: IBMM distributions for model sizes $\{1, 2, ..., 15\}$, for the datasets (a) `covtype.binary` and (b) `Sensorless`. These are for the combination of using LPM as the model with GBM as an oracle. Darker curves indicate higher model sizes.

A.11 Improvements Relative to Oracle

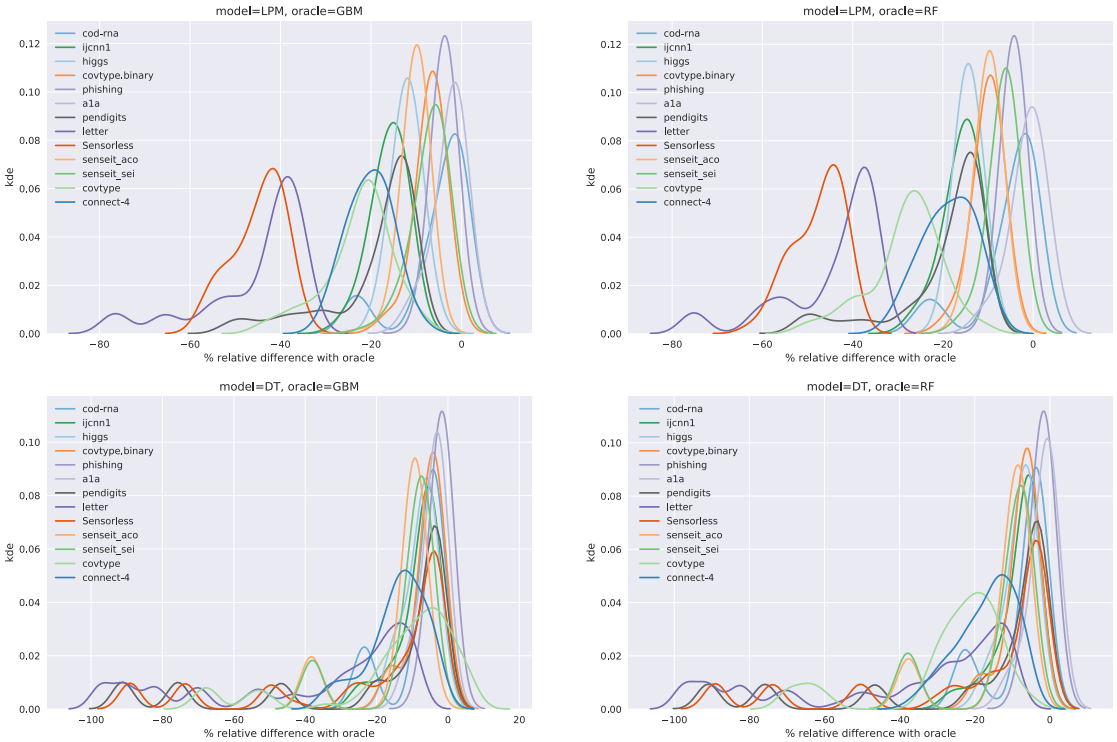


Figure A.8: These plots show the distribution of the percentage relative difference of a model's improved score w.r.t. to the accuracy of the oracle it is trained with. We note that this quantity is almost always non-positive as claimed in Equations 4.2, 4.3 and 4.4.

Some of the positive values we see in Figure A.8 may be attributed to spillovers due to the *kde* fit. Their magnitudes and occurrences are typically small: these are detailed in Table 4.8.

A.12 Feature Selection for n-gram DT

For the experiments in Section 4.3.3.1, we perform feature selection to reduce their running time. After the n-gram ($n \in \{1, 2, 3\}$) vocabulary is created from the training data, we perform a χ^2-test to select the $k-$best features. The original number of features is 5308. To pick the smallest useful set of features, we test different values of $k \leq 1000$. A test constitutes of:

1. Construct a DT, for a given max_depth, on the original set of features. Obtain its test accuracy, $F1_{all}$.
2. Construct a DT, with the same max_depth, using only the k best features as per the χ^2-test, and obtain its test accuracy $F1_k$.
3. Report:
$$\delta F1 = 100 \times \frac{F1_k - F1_{all}}{F1_{all}}$$

We use the "macro" averaging for the $F1$ score to be consistent with other experiments in the chapter. All reported $\delta F1$ are *averaged over ten runs*.

Figure A.9 shows how $\delta F1$ varies with k.

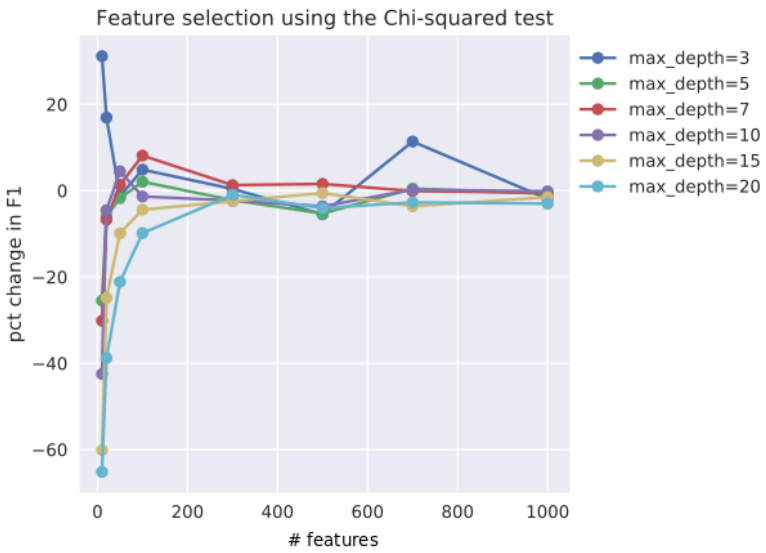


Figure A.9: The relationship between $\delta F1$ and $k \leq 1000$. Each data point is an average over ten runs.

We observe that at around 600 features, $\delta F1 \approx 0\%$. The only exception is the case for $max\_depth = 3$, but that is admissible since $\delta F1 > 0$, i.e., we seem to be

improving the accuracy.

A.13 Running Time for Sampling

In this section, we analyze the running time of the sampling process in the oracle based setup (Algorithm 8) relative to the model fitting time in the overall algorithm (Algorithm 9). Of course, these numbers will vary wrt the dataset used and the model being fit, but we present some sample numbers here.

We vary the size of the dataset used for training and use the following values, $N = \{100, 500, 1000, 1500, ..., 5000\}$. At each dataset size, we measure the wall-clock time in seconds for the following and report values averaged over *ten trials*:

1. Sampling N instances from the IBMM. The $Beta$ priors are randomly selected in the range $[0.1, 10]$. The α for the Dirichlet Process is fixed at a given value.

2. Fitting a CART decision tree on a dataset of size N, for values of $max\_depth \in \{3, 5\}$. To get a broad sense of running times, we use two datasets `-cod-rna` and `a1a`, with 123 and 8 features respectively. Each has two classes.

Multiple values of α were tested, $\alpha \in \{0.1, 2, 4, 6, 8, 10\}$. The plots are presented in Figure A.10.

Not surprisingly, it is seen that as the number of clusters increase (higher α), the time for sampling increases due to sampling from multiple $Beta$ components. In fact, surpasses the time to learn a CART decision tree. As mentioned, this will vary across models. It is also important to note that because of the model *selection* performed in practice, requiring multiple models to be trained, the total time model training time is higher than what is shown.

Nevertheless, this points to an interesting direction for future work: accelerate the sampling process, e.g., use a distribution like the *Kumaraswamy* distribution (Kumaraswamy, 1980), where sampling is fast because of its simple form for the *Cumulative Distribution Function (CDF)*.

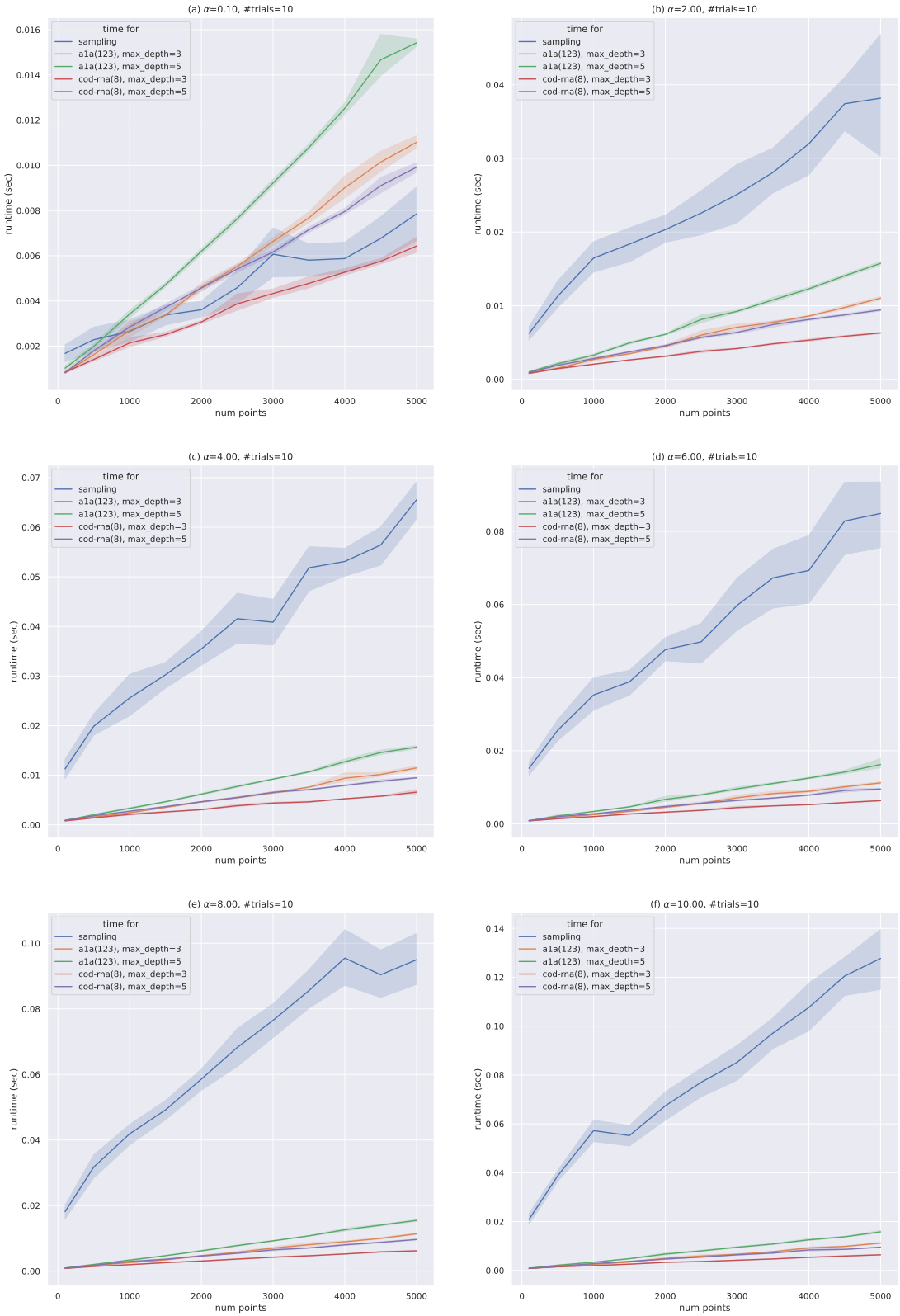


Figure A.10: The running time (*y-axis*) of sampling is shown in comparison to model fit time, for various dataset sizes (*x-axis*). Plots (a)-(f) show these comparisons for $\alpha \in \{0.1, 2, 4, 6, 8, 10\}$ and $\alpha = 10$ respectively. A legend of the form "a1a(123), max_depth=3" represents a model fit on the dataset `a1a`, which has 123 features, where the tree was restricted to $max\_depth = 3$. Reported values were averaged over *ten* trials. Note that the scale of the *y-axes* vary across the plots.

REFERENCES

1. **Ackley, D. H.**, *A connectionist machine for genetic hillclimbing*. Kluwer Boston Inc.,Hingham, MA, United States, 1987. URL https://www.osti.gov/biblio/6210712. Availability: Kluwer Boston Inc., 190 Old Derby St., Hingham, MA 02043.

2. **Agarwal, A.**, **Y. S. Tan**, **O. Ronen**, **C. Singh**, and **B. Yu**, Hierarchical shrinkage: Improving the accuracy and interpretability of tree-based models. *In* **K. Chaudhuri**, **S. Jegelka**, **L. Song**, **C. Szepesvari**, **G. Niu**, and **S. Sabato** (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*. PMLR, 2022. URL https://proceedings.mlr.press/v162/agarwal22b.html.

3. **Alaa, A. M.** and **M. van der Schaar**, Attentive state-space modeling of disease progression. *In* **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019*a*. URL https://proceedings.neurips.cc/paper/2019/file/1d0932d7f57ce74d9d9931a2c6db8a06-Paper.pdf.

4. **Alaa, A. M.** and **M. van der Schaar**, Demystifying black-box models with symbolic metamodels. *In* **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019*b*. URL https://proceedings.neurips.cc/paper/2019/file/567b8f5f423af15818a068235807edc0-Paper.pdf.

5. **Aldous, D. J.**, Exchangeability and related topics. *In* **P. L. Hennequin** (ed.), *École d'Été de Probabilités de Saint-Flour XIII – 1983*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1985. ISBN 978-3-540-39316-0.

6. **Alimoglu, F.** and **E. Alpaydin**, 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 (TAINN 96*. 1996.

7. **Alloway, T.** (2015). Big data: Credit where credit's due. https://www.ft.com/content/7933792e-a2e6-11e4-9c06-00144feab7de.

8. **Alvarez-Melis, D.** and **T. S. Jaakkola** (2018). On the robustness of interpretability methods. *Proceedings of the 2018 ICML Workshop in Human Interpretability in Machine Learning*, **abs/1806.08049**. URL http://arxiv.org/abs/1806.08049.

9. **Alvi, A.**, **B. Ru**, **J.-P. Calliess**, **S. Roberts**, and **M. A. Osborne**, Asynchronous batch Bayesian optimisation with improved local penalisation. *In* **K. Chaudhuri** and

R. Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*. PMLR, Long Beach, California, USA, 2019. URL http://proceedings.mlr.press/v97/alvi19a.html.

10. **Ancona, M.**, **C. Oztireli**, and **M. Gross**, Explaining deep neural networks with a polynomial time algorithm for shapley value approximation. *In* **K. Chaudhuri** and **R. Salakhutdinov** (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*. PMLR, Long Beach, California, USA, 2019. URL http://proceedings.mlr.press/v97/ancona19a.html.

11. **Angelino, E.**, **N. Larus-Stone**, **D. Alabi**, **M. Seltzer**, and **C. Rudin**, Learning certifiably optimal rule lists. *In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17. ACM, New York, NY, USA, 2017. ISBN 978-1-4503-4887-4. URL http://doi.acm.org/10.1145/3097983.3098047.

12. **Angiulli, F.**, Fast condensed nearest neighbor rule. *In Proceedings of the 22nd International Conference on Machine Learning*, ICML '05. Association for Computing Machinery, New York, NY, USA, 2005. ISBN 1595931805. URL https://doi.org/10.1145/1102351.1102355.

13. **Angwin, J.**, **J. Larson**, **S. Mattu**, and **L. Kirchner** (2016). Machine Bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing.

14. **Arya, V.**, **R. K. E. Bellamy**, **P.-Y. Chen**, **A. Dhurandhar**, **M. Hind**, **S. C. Hoffman**, **S. Houde**, **Q. V. Liao**, **R. Luss**, **A. Mojsilović**, **S. Mourad**, **P. Pedemonte**, **R. Raghavendra**, **J. Richards**, **P. Sattigeri**, **K. Shanmugam**, **M. Singh**, **K. R. Varshney**, **D. Wei**, and **Y. Zhang**, Ai explainability 360: Hands-on tutorial. *In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450369367. URL https://doi.org/10.1145/3351095.3375667.

15. **Avellaneda, F.** (2020). Efficient inference of optimal decision trees. *Proceedings of the AAAI Conference on Artificial Intelligence*, **34**(04), 3195–3202. URL https://ojs.aaai.org/index.php/AAAI/article/view/5717.

16. **Bachem, O.**, **M. Lucic**, and **A. Krause** (2017). Practical Coreset Constructions for Machine Learning. Preprint at https://ui.adsabs.harvard.edu/abs/2017arXiv170306476B.

17. **Baldi, P.**, **P. Sadowski**, and **D. Whiteson** (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, **5**(1), 4308. ISSN 2041-1723. URL https://doi.org/10.1038/ncomms5308.

18. **Barceló, P.**, **M. Monet**, **J. Pérez**, and **B. Subercaseaux**, Model interpretability through the lens of computational complexity. *In* **H. Larochelle**, **M. Ranzato**, **R. Hadsell**, **M. Balcan**, and **H. Lin** (eds.), *Advances in Neural Information Processing Systems*, volume 33. Curran Associates, Inc.,

2020. URL https://proceedings.neurips.cc/paper/2020/file/b1adda14824f50ef24ff1c05bb66faf3-Paper.pdf.

19. **Barredo Arrieta, A.**, **N. Díaz-Rodríguez**, **J. Del Ser**, **A. Bennetot**, **S. Tabik**, **A. Barbado**, **S. Garcia**, **S. Gil-Lopez**, **D. Molina**, **R. Benjamins**, **R. Chatila**, and **F. Herrera** (2020). Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, **58**, 82–115. ISSN 1566-2535. URL https://www.sciencedirect.com/science/article/pii/S1566253519308103.

20. **Bastings, J.**, **W. Aziz**, and **I. Titov**, Interpretable neural predictions with differentiable binary variables. *In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Florence, Italy, 2019. URL https://www.aclweb.org/anthology/P19-1284.

21. **Benavoli, A.**, **G. Corani**, and **F. Mangili** (2016). Should we really use post-hoc tests based on mean-ranks? *Journal of Machine Learning Research*, **17**(5), 1–10. URL http://jmlr.org/papers/v17/benavoli16a.html.

22. **Bergstra, J.**, **R. Bardenet**, **Y. Bengio**, and **B. Kégl**, Algorithms for hyper-parameter optimization. *In Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11. Curran Associates Inc., USA, 2011. ISBN 978-1-61839-599-3. URL http://dl.acm.org/citation.cfm?id=2986459.2986743.

23. **Bergstra, J.**, **D. Yamins**, and **D. D. Cox**, 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. JMLR.org, 2013. URL http://dl.acm.org/citation.cfm?id=3042817.3042832.

24. **Bertsimas, D.**, **A. Delarue**, **P. Jaillet**, and **S. Martin** (2019). The price of interpretability. *CoRR*, **abs/1907.03419**. URL http://arxiv.org/abs/1907.03419.

25. **Bhatt, U.**, **A. Xiang**, **S. Sharma**, **A. Weller**, **A. Taly**, **Y. Jia**, **J. Ghosh**, **R. Puri**, **J. M. F. Moura**, and **P. Eckersley**, Explainable machine learning in deployment. *In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, FAT* '20. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450369367. URL https://doi.org/10.1145/3351095.3375624.

26. **Blackard, J. A.** (1998). *Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types*. Ph.D. thesis, USA. AAI9921979.

27. **Blackwell, D.** and **J. B. MacQueen** (1973). Ferguson distributions via polya urn schemes. *Ann. Statist.*, **1**(2), 353–355. URL https://doi.org/10.1214/aos/1176342372.

28. **Blaser, R.** and **P. Fryzlewicz** (2016). Random rotation ensembles. *Journal of Machine Learning Research*, **17**(4), 1–26. URL http://jmlr.org/papers/v17/blaser16a.html.

29. **Blei, D.** (2007). COS 597C Notes, Bayesian Nonparametrics. https://www.cs.princeton.edu/courses/archive/fall07/cos597C/scribe/20070921.pdf.

30. **Blondel, M.**, **Q. Berthet**, **M. Cuturi**, **R. Frostig**, **S. Hoyer**, **F. Llinares-López**, **F. Pedregosa**, and **J.-P. Vert** (2021). Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*.

31. **Bloniarz, A.**, **A. Talwalkar**, **B. Yu**, and **C. Wu**, Supervised neighborhoods for distributed nonparametric regression. *In* **A. Gretton** and **C. C. Robert** (eds.), *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *Proceedings of Machine Learning Research*. PMLR, Cadiz, Spain, 2016. URL https://proceedings.mlr.press/v51/bloniarz16.html.

32. **Bolte, J.**, **T. Le**, **E. Pauwels**, and **A. Silveti-Falls** (2021). Nonsmooth Implicit Differentiation for Machine Learning and Optimization. URL https://hal.archives-ouvertes.fr/hal-03251332. Working paper or preprint.

33. **Box, G. E. P.** and **D. R. Cox** (1964). An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, **26**(2), 211–252. ISSN 00359246. URL http://www.jstor.org/stable/2984418.

34. **Bradbury, J.**, **R. Frostig**, **P. Hawkins**, **M. J. Johnson**, **C. Leary**, **D. Maclaurin**, **G. Necula**, **A. Paszke**, **J. VanderPlas**, **S. Wanderman-Milne**, and **Q. Zhang** (2018). JAX: composable transformations of Python+NumPy programs. URL http://github.com/google/jax.

35. **Breiman, L.** *et al.*, *Classification and Regression Trees*. Chapman & Hall, New York, 1984. ISBN 0-412-04841-8.

36. **Brochu, E.**, **V. M. Cora**, and **N. de Freitas** (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, **abs/1012.2599**.

37. **Broderick, T.** (2015). Bayesian nonparametrics. https://tamarabroderick.com/tutorial_2015_mlss_tubingen.html.

38. **Broomhead, D.** and **D. Lowe** (1988). Multivariable functional interpolation and adaptive networks. *Complex Systems*, **2**, 321–355.

39. **Bucilă, C.**, **R. Caruana**, and **A. Niculescu-Mizil**, Model compression. *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06. Association for Computing Machinery, New York, NY, USA, 2006. ISBN 1595933395. URL https://doi.org/10.1145/1150402.1150464.

40. **Caruana, R.**, **Y. Lou**, **J. Gehrke**, **P. Koch**, **M. Sturm**, and **N. Elhadad**, 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. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3664-2. URL http://doi.acm.org/10.1145/2783258.2788613.

41. **Castellanos, S.** and **K. S. Nash** (2018). Bank of America Confronts AI's 'Black Box' With Fraud Detection Effort. https://blogs.wsj.com/cio/2018/05/11/bank-of-america-confronts-ais-black-box-with-fraud-detection-effort/.

42. **Chang, C.-C.** and **C.-J. Lin**, Ijcnn 2001 challenge: Generalization ability and text decoding. *In Proceedings of IJCNN. IEEE*. 2001.

43. **Chang, C.-C.** and **C.-J. Lin** (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, **2**, 27:1–27:27. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

44. **Chawla, N. V.**, **K. W. Bowyer**, **L. O. Hall**, and **W. P. Kegelmeyer** (2002). Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, **16**(1), 321–357. ISSN 1076-9757.

45. **Chen, J.**, **L. Song**, **M. Wainwright**, and **M. Jordan**, Learning to explain: An information-theoretic perspective on model interpretation. *In* **J. Dy** and **A. Krause** (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*. PMLR, 2018. URL http://proceedings.mlr.press/v80/chen18j.html.

46. **Cho, K.**, **B. van Merriënboer**, **C. Gulcehre**, **D. Bahdanau**, **F. Bougares**, **H. Schwenk**, and **Y. Bengio**, Learning phrase representations using RNN encoder–decoder for statistical machine translation. *In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 2014. URL https://www.aclweb.org/anthology/D14-1179.

47. **Clarke, Y. D.** (2019). Algorithmic Accountability Act of 2019. https://www.congress.gov/bill/116th-congress/house-bill/2231.

48. **Collobert, R.**, **S. Bengio**, and **Y. Bengio**, A parallel mixture of svms for very large scale problems. *In* **T. G. Dietterich**, **S. Becker**, and **Z. Ghahramani** (eds.), *Advances in Neural Information Processing Systems 14*. MIT Press, 2002, 633–640. URL http://papers.nips.cc/paper/1949-a-parallel-mixture-of-svms-for-very-large-scale-problems.pdf.

49. **Craven, M.** and **J. Shavlik**, Extracting tree-structured representations of trained networks. *In* **D. Touretzky**, **M. Mozer**, and **M. Hasselmo** (eds.), *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995. URL https://proceedings.neurips.cc/paper/1995/file/45f31d16b1058d586fc3be7207b58053-Paper.pdf.

50. **Dai, W.**, **Q. Yang**, **G.-R. Xue**, and **Y. Yu**, Boosting for transfer learning. *In Proceedings of the 24th International Conference on Machine Learning*, ICML '07. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-793-3. URL http://doi.acm.org/10.1145/1273496.1273521.

51. **Dai, Z.**, **H. Yu**, **B. K. H. Low**, and **P. Jaillet**, Bayesian optimization meets Bayesian optimal stopping. *In* **K. Chaudhuri** and **R. Salakhutdinov** (eds.), *Proceedings of*

the 36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*. PMLR, Long Beach, California, USA, 2019. URL http://proceedings.mlr.press/v97/dai19a.html.

52. **Danka, T.** and **P. Horvath** (2018). modAL: A modular active learning framework for Python. URL https://github.com/cosmic-cortex/modAL. Available on arXiv at https://arxiv.org/abs/1805.00979.

53. **Dao, T.**, **G. M. Kamath**, **V. Syrgkanis**, and **L. Mackey**, Knowledge distillation as semiparametric inference. *In International Conference on Learning Representations*. 2021.

54. **Dasgupta, S.** (2011). Two faces of active learning. *Theor. Comput. Sci.*, **412**(19), 1767–1781. ISSN 0304-3975. URL http://dx.doi.org/10.1016/j.tcs.2010.12.054.

55. **Dasgupta, S.** and **D. Hsu**, Hierarchical sampling for active learning. *In Proceedings of the 25th International Conference on Machine Learning*, ICML '08. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-205-4. URL http://doi.acm.org/10.1145/1390156.1390183.

56. **Dempster, A. P.**, **N. M. Laird**, and **D. B. Rubin** (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, **39**(1), 1–22. URL https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1977.tb01600.x.

57. **Demšar, J.** (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, **7**(1), 1–30. URL http://jmlr.org/papers/v7/demsar06a.html.

58. **Deng, J.**, **W. Dong**, **R. Socher**, **L.-J. Li**, **K. Li**, and **L. Fei-Fei**, Imagenet: A large-scale hierarchical image database. *In 2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009.

59. **Dennis, Don Kurian and Gaurkar, Yash and Gopinath, Sridhar and Goyal, Sachin and Gupta, Chirag and Jain, Moksh and Jaiswal, Shikhar and Kumar, Ashish and Kusupati, Aditya and Lovett, Chris and Patil, Shishir G and Saha, Oindrila and Simhadri, Harsha Vardhan** (2021). EdgeML: Machine Learning for resource-constrained edge devices. URL https://github.com/Microsoft/EdgeML.

60. **Desai, S.** and **H. G. Ramaswamy**, Ablation-cam: Visual explanations for deep convolutional network via gradient-free localization. *In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. 2020.

61. **Devlin, J.**, **M.-W. Chang**, **K. Lee**, and **K. Toutanova**, BERT: Pre-training of deep bidirectional transformers for language understanding. *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 2019. URL https://www.aclweb.org/anthology/N19-1423.

62. **Dhurandhar, A.**, **K. Shanmugam**, **R. Luss**, and **P. A. Olsen**, Improving simple models with confidence profiles. *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/972cda1e62b72640cb7ac702714a115f-Paper.pdf.

63. **Di Castro, F.** and **E. Bertini** (2019). Surrogate decision tree visualization interpreting and visualizing black-box classification models with surrogate decision tree. *CEUR Workshop Proceedings*, **2327**. ISSN 1613-0073. 2019 Joint ACM IUI Workshops, ACMIUI-WS 2019 ; Conference date: 20-03-2019.

64. **Doshi-Velez, F.** and **B. Kim** (2017). Towards a rigorous science of interpretable machine learning. *arXiv*. URL https://arxiv.org/abs/1702.08608.

65. **Dua, D.** and **C. Graff** (2017). UCI machine learning repository. URL http://archive.ics.uci.edu/ml.

66. **Duarte, M. F.** and **Y. H. Hu** (2004). Vehicle classification in distributed sensor networks. *J. Parallel Distrib. Comput.*, **64**(7), 826–838. ISSN 0743-7315. URL https://doi.org/10.1016/j.jpdc.2004.03.020.

67. **Dziugaite, G. K.**, **S. Ben-David**, and **D. M. Roy** (2020). Enforcing interpretability and its statistical impacts: Trade-offs between accuracy and interpretability. URL https://arxiv.org/abs/2010.13764.

68. **Efron, B.**, **T. Hastie**, **I. Johnstone**, and **R. Tibshirani** (2004). Least angle regression. *The Annals of Statistics*, **32**(2), 407 – 499. URL https://doi.org/10.1214/009053604000000067.

69. **Elsken, T.**, **J. H. Metzen**, and **F. Hutter** (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, **20**(55), 1–21. URL http://jmlr.org/papers/v20/18-598.html.

70. **Engen, S.** (1975). A note on the geometric series as a species frequency model. *Biometrika*, **62**(3), 697–699. ISSN 0006-3444. URL https://doi.org/10.1093/biomet/62.3.697.

71. **Eric, B.**, **N. Freitas**, and **A. Ghosh**, Active preference learning with discrete choice data. *In* **J. Platt**, **D. Koller**, **Y. Singer**, and **S. Roweis** (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL https://proceedings.neurips.cc/paper/2007/file/b6a1085a27ab7bff7550f8a3bd017df8-Paper.pdf.

72. **Eriksson, D.**, **D. Bindel**, and **C. A. Shoemaker** (2019). pysot and poap: An event-driven asynchronous framework for surrogate optimization. URL https://arxiv.org/abs/1908.00420.

73. **Ewens, W. J.**, *Population Genetics Theory - The Past and the Future*. Springer Netherlands, Dordrecht, 1990. ISBN 978-94-009-0513-9, 177–227. URL https://doi.org/10.1007/978-94-009-0513-9_4.

74. **Falkner, S.**, **A. Klein**, and **F. Hutter**, Bohb: Robust and efficient hyperparameter optimization at scale. *In ICML*. 2018. URL http://proceedings.mlr.press/v80/falkner18a.html.

75. **Feldman, J.** (2000). Minimization of boolean complexity in human concept learning. *Nature*, **407**, 630–3.

76. **Ferguson, T. S.** (1973). A Bayesian Analysis of Some Nonparametric Problems. *The Annals of Statistics*, **1**(2), 209 – 230. URL https://doi.org/10.1214/aos/1176342360.

77. **Feurer, M.** and **F. Hutter**, Hyperparameter optimization. *In* Hutter *et al.* (2019), 3–38.

78. **Feurer, M.** and **F. Hutter**, *Hyperparameter Optimization*. Springer International Publishing, Cham, 2019*b*. ISBN 978-3-030-05318-5, 3–33. URL https://doi.org/10.1007/978-3-030-05318-5_1.

79. **Fountain-Jones, N. M.**, **G. Machado**, **S. Carver**, **C. Packer**, **M. Recamonde-Mendoza**, and **M. E. Craft** (2019). How to make more from exposure data? an integrated machine learning pipeline to predict pathogen exposure. *bioRxiv*. URL https://www.biorxiv.org/content/early/2019/03/06/569012.

80. **Freitas, A. A.** (2014). Comprehensible classification models: A position paper. *SIGKDD Explor. Newsl.*, **15**(1), 1–10. ISSN 1931-0145. URL https://doi.org/10.1145/2594473.2594475.

81. **Friedman, J. H.** (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, **29**(5), 1189 – 1232. URL https://doi.org/10.1214/aos/1013203451.

82. **Friedman, J. H.** and **B. E. Popescu** (2008). Predictive learning via rule ensembles. *The Annals of Applied Statistics*, **2**(3), 916–954. ISSN 19326157. URL http://www.jstor.org/stable/30245114.

83. **Frost, N.**, **M. Moshkovitz**, and **C. Rashtchian** (2020). Exkmc: Expanding explainable k-means clustering. *arXiv preprint arXiv:2006.02399*.

84. **Gelbart, M. A.**, **J. Snoek**, and **R. P. Adams**, Bayesian optimization with unknown constraints. *In Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14. AUAI Press, Arlington, Virginia, United States, 2014. ISBN 978-0-9749039-1-0. URL http://dl.acm.org/citation.cfm?id=3020751.3020778.

85. **Gelfand, S. B.** and **S. K. Mitter** (1989). Simulated annealing with noisy or imprecise energy measurements. *Journal of Optimization Theory and Applications*, **62**, 49–62.

86. **Gelman, A.**, **J. B. Carlin**, **H. S. Stern**, **D. B. Dunson**, **A. Vehtari**, and **D. B. Rubin**, *Bayesian Data Analysis*. Chapman and Hall/CRC, 2021, 3 edition. URL http://www.stat.columbia.edu/~gelman/book/BDA3.pdf.

87. **Ghorbani, A.** and **J. Zou**, Data shapley: Equitable valuation of data for machine learning. *In* **K. Chaudhuri** and **R. Salakhutdinov** (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*. PMLR, 2019. URL https://proceedings.mlr.press/v97/ghorbani19c.html.

88. **Ghose, A.** (2020). compactem. URL https://pypi.org/project/compactem/.

89. **Ghose, A.** and **B. Ravindran** (2020). Interpretability with accurate small models. *Frontiers in Artificial Intelligence*, **3**, 3. ISSN 2624-8212. URL https://www.frontiersin.org/article/10.3389/frai.2020.00003.

90. **Goodman, B.** and **S. Flaxman** (2017). European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, **38**(3), 50–57. URL https://ojs.aaai.org/index.php/aimagazine/article/view/2741.

91. **Gou, J.**, **B. Yu**, **S. J. Maybank**, and **D. Tao** (2021). Knowledge distillation: A survey. *International Journal of Computer Vision*, **129**(6), 1789–1819. ISSN 1573-1405. URL https://doi.org/10.1007/s11263-021-01453-z.

92. **Graves, A.** (2013). Generating sequences with recurrent neural networks. *CoRR*, **abs/1308.0850**. URL http://arxiv.org/abs/1308.0850.

93. **Grill, J.-B.**, **M. Valko**, **R. Munos**, and **R. Munos**, Black-box optimization of noisy functions with unknown smoothness. *In* **C. Cortes**, **N. D. Lawrence**, **D. D. Lee**, **M. Sugiyama**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015, 667–675. URL http://papers.nips.cc/paper/5721-black-box-optimization-of-noisy-functions-with-unknown-smoothness.pdf.

94. **Guidotti, R.**, **A. Monreale**, **F. Giannotti**, **D. Pedreschi**, **S. Ruggieri**, and **F. Turini** (2019). Factual and counterfactual explanations for black box decision making. *IEEE Intelligent Systems*, **34**(6), 14–23.

95. **Gunning, D.** (2016). Explainable Artificial Intelligence. https://www.darpa.mil/program/explainable-artificial-intelligence.

96. **Guo, C.**, **G. Pleiss**, **Y. Sun**, and **K. Q. Weinberger**, On calibration of modern neural networks. *In Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17. JMLR.org, 2017.

97. **Gupta, C.**, **A. S. Suggala**, **A. Goyal**, **H. V. Simhadri**, **B. Paranjape**, **A. Kumar**, **S. Goyal**, **R. Udupa**, **M. Varma**, and **P. Jain**, ProtoNN: Compressed and accurate kNN for resource-scarce devices. *In* **D. Precup** and **Y. W. Teh** (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*. PMLR, 2017. URL https://proceedings.mlr.press/v70/gupta17a.html.

98. **Gutjahr, W. J.** and **G. C. Pflug** (1996). Simulated annealing for noisy cost functions. *Journal of Global Optimization*, **8**(1), 1–13. ISSN 1573-2916. URL https://doi.org/10.1007/BF00229298.

99. **Hansen, N.** and **S. Kern**, Evaluating the CMA evolution strategy on multimodal test functions. *In* **X. Yao** *et al.* (eds.), *Parallel Problem Solving from Nature PPSN VIII*, volume 3242 of *LNCS*. Springer, 2004.

100. **Hansen, N.** and **A. Ostermeier** (2001). Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, **9**(2), 159–195. ISSN 1063-6560. URL http://dx.doi.org/10.1162/106365601750190398.

101. **He, H.**, **Y. Bai**, **E. A. Garcia**, and **S. Li**, Adasyn: Adaptive synthetic sampling approach for imbalanced learning. *In 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. 2008.

102. **Herman, B.** (2017). The promise and peril of human evaluation for model interpretability. URL http://arxiv.org/abs/1711.07414. Presented at NIPS 2017 Symposium on Interpretable Machine Learning. Available at: https://arxiv.org/abs/1711.09889v3.

103. **Hernández-Lobato, J. M.**, **M. A. Gelbart**, **R. P. Adams**, **M. W. Hoffman**, and **Z. Ghahramani** (2016). A general framework for constrained bayesian optimization using information-based search. *J. Mach. Learn. Res.*, **17**(1), 5549–5601. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2946645.3053442.

104. **Hinton, G.**, **O. Vinyals**, and **J. Dean**, Distilling the knowledge in a neural network. *In NIPS Deep Learning and Representation Learning Workshop*. 2015. URL http://arxiv.org/abs/1503.02531.

105. **Howard, J.** and **S. Ruder**, Universal language model fine-tuning for text classification. *In ACL*. Association for Computational Linguistics, 2018. URL http://arxiv.org/abs/1801.06146.

106. **Hsu, C.-W.** and **C.-J. Lin** (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, **13**, 415–25.

107. **Hu, X.**, **C. Rudin**, and **M. Seltzer**, Optimal sparse decision trees. *In* **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, 7265–7273. URL http://papers.nips.cc/paper/8947-optimal-sparse-decision-trees.pdf.

108. **Hutter, F.**, **H. H. Hoos**, and **K. Leyton-Brown**, Sequential model-based optimization for general algorithm configuration. *In Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-25565-6. URL http://dx.doi.org/10.1007/978-3-642-25566-3_40.

109. **Hutter, F.**, **L. Kotthoff**, and **J. Vanschoren** (eds.), *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

110. **Hyafil, L.** and **R. L. Rivest** (1976). Constructing optimal binary decision trees is np-complete. *Inf. Process. Lett.*, **5**, 15–17.

111. **Ioffe, S.** and **C. Szegedy**, Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15. JMLR.org, 2015.

112. **Ishwaran, H.** and **L. F. James** (2001). Gibbs sampling methods for stick-breaking priors. *Journal of the American Statistical Association*, **96**(453), 161–173. ISSN 01621459. URL http://www.jstor.org/stable/2670356.

113. **James Bergstra**, **Dan Yamins**, and **David D. Cox**, Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. *In* **Stéfan van der Walt**, **Jarrod Millman**, and **Katy Huff** (eds.), *Proceedings of the 12th Python in Science Conference*. 2013.

114. **Japkowicz, N.** and **M. Shah**, *Evaluating Learning Algorithms: A Classification Perspective*. Cambridge University Press, 2011.

115. **Japkowicz, N.** and **S. Stephen** (2002). The class imbalance problem: A systematic study. *Intell. Data Anal.*, **6**(5), 429–449. ISSN 1088-467X. URL http://dl.acm.org/citation.cfm?id=1293951.1293954.

116. **Jeh, G.** and **J. Widom**, Simrank: A measure of structural-context similarity. *In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02. ACM, New York, NY, USA, 2002. ISBN 1-58113-567-X. URL http://doi.acm.org/10.1145/775047.775126.

117. **Jones, D. R.**, **C. D. Perttunen**, and **B. E. Stuckman** (1993). Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, **79**(1), 157–181. ISSN 1573-2878. URL https://doi.org/10.1007/BF00941892.

118. **Jones, E.**, **T. Oliphant**, **P. Peterson**, *et al.* (2001). SciPy: Open source scientific tools for Python. URL http://www.scipy.org/. http://www.scipy.org/.

119. **Juan, Y.**, **Y. Zhuang**, **W.-S. Chin**, and **C.-J. Lin**, Field-aware factorization machines for ctr prediction. *In Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16. Association for Computing Machinery, New York, NY, USA, 2016. ISBN 9781450340359. URL https://doi.org/10.1145/2959100.2959134.

120. **Jurafsky, D.** and **J. Martin** (2019). Speech and language processing. Preprint on webpage at https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf.

121. **Kamishima, T.**, **M. Hamasaki**, and **S. Akaho**, Trbagg: A simple transfer learning method and its application to personalization in collaborative tagging. *In Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09. IEEE Computer Society, Washington, DC, USA, 2009. ISBN 978-0-7695-3895-2. URL https://doi.org/10.1109/ICDM.2009.9.

122. **Kaur, H.**, **H. Nori**, **S. Jenkins**, **R. Caruana**, **H. Wallach**, and **J. Wortman Vaughan**, Interpreting interpretability: Understanding data scientists' use of interpretability tools for machine learning. *In CHI 2020*. 2020. URL https://www.microsoft.com/en-us/research/publication/interpreting-interpretability-

understanding-data-scientists-use-of-interpretability-tools-for-machine-learning/. CHI 2020 Honorable Mention Award.

123. **Ke, G.**, **Q. Meng**, **T. Finley**, **T. Wang**, **W. Chen**, **W. Ma**, **Q. Ye**, and **T.-Y. Liu**, Lightgbm: A highly efficient gradient boosting decision tree. *In Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17. Curran Associates Inc., USA, 2017. ISBN 978-1-5108-6096-4. URL http://dl.acm.org/citation.cfm?id=3294996.3295074.

124. **Kennedy, J.** and **R. Eberhart**, Particle swarm optimization. *In Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4. 1995.

125. **Kim, J.** and **J. Canny**, Interpretable learning for self-driving cars by visualizing causal attention. *In IEEE International Conference on Computer Vision (ICCV)*. 2017.

126. **Kim, J.**, **A. Rohrbach**, **T. Darrell**, **J. Canny**, and **Z. Akata**, Textual explanations for self-driving vehicles. *In Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.

127. **Kirkpatrick, S.**, **C. D. Gelatt**, and **M. P. Vecchi** (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680. ISSN 00368075. URL http://www.jstor.org/stable/1690046.

128. **Krishnan, M.** (2020). Against interpretability: a critical examination of the interpretability problem in machine learning. *Philosophy & Technology*, **33**(3), 487–502. ISSN 2210-5441. URL https://doi.org/10.1007/s13347-019-00372-9.

129. **Kulesza, T.**, **S. Stumpf**, **M. Burnett**, **S. Yang**, **I. Kwan**, and **W.-K. Wong**, Too much, too little, or just right? ways explanations impact end users' mental models. *In 2013 IEEE Symposium on Visual Languages and Human Centric Computing*. 2013.

130. **Kumaraswamy, P.** (1980). A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, **46**(1), 79 – 88. ISSN 0022-1694. URL http://www.sciencedirect.com/science/article/pii/0022169480900360.

131. **Kusner, M.**, **S. Tyree**, **K. Weinberger**, and **K. Agrawal**, Stochastic neighbor compression. *In* **E. P. Xing** and **T. Jebara** (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*. PMLR, Bejing, China, 2014. URL https://proceedings.mlr.press/v32/kusner14.html.

132. **Laber, E. S.**, **L. Murtinho**, and **F. Oliveira** (2021). Shallow decision trees for explainable k-means clustering. *CoRR*, **abs/2112.14718**. URL https://arxiv.org/abs/2112.14718.

133. **Lage, I.**, **E. Chen**, **J. He**, **M. Narayanan**, **B. Kim**, **S. J. Gershman**, and **F. Doshi-Velez** (2019). Human evaluation of models built for interpretability. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, **7**(1), 59–67. URL https://ojs.aaai.org/index.php/HCOMP/article/view/5280.

134. **Lakkaraju, H.**, **S. H. Bach**, and **J. Leskovec**, Interpretable decision sets: A joint framework for description and prediction. *In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4232-2. URL http://doi.acm.org/10.1145/2939672.2939874.

135. **Lapuschkin, S.**, **S. Wäldchen**, **A. Binder**, **G. Montavon**, **W. Samek**, and **K.-R. Müller** (2019). Unmasking clever hans predictors and assessing what machines really learn. *Nature Communications*, **10**(1), 1096. ISSN 2041-1723. URL https://doi.org/10.1038/s41467-019-08987-4.

136. **Larson, J.**, **S. Mattu**, **L. Kirchner**, and **J. Angwin** (2016). How We Analyzed the COMPAS Recidivism Algorithm. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm.

137. **Lee, E.**, **D. Braines**, **M. Stiffler**, **A. Hudler**, and **D. Harborne**, Developing the sensitivity of LIME for better machine learning explanation. *In* **T. Pham** (ed.), *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006. International Society for Optics and Photonics, SPIE, 2019. URL https://doi.org/10.1117/12.2520149.

138. **Lei, T.**, **R. Barzilay**, and **T. Jaakkola**, Rationalizing neural predictions. *In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, 2016. URL https://www.aclweb.org/anthology/D16-1011.

139. **Letham, B.**, **B. Karrer**, **G. Ottoni**, and **E. Bakshy** (2017). Constrained bayesian optimization with noisy experiments. *Bayesian Analysis*.

140. **Letham, B.**, **C. Rudin**, **T. H. McCormick**, and **D. Madigan** (2015). Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *The Annals of Applied Statistics*, **9**(3), 1350 – 1371. URL https://doi.org/10.1214/15-AOAS848.

141. **Levesque, J.-C.**, **A. Durand**, **C. Gagné**, and **R. Sabourin** (2017). Bayesian optimization for conditional hyperparameter spaces. *2017 International Joint Conference on Neural Networks (IJCNN)*, 286–293.

142. **Lewis, D. D.** and **W. A. Gale**, A sequential algorithm for training text classifiers. *In Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94. Springer-Verlag New York, Inc., New York, NY, USA, 1994. ISBN 0-387-19889-X. URL http://dl.acm.org/citation.cfm?id=188490.188495.

143. **Li, C.**, **S. Gupta**, **S. Rana**, **V. Nguyen**, **S. Venkatesh**, and **A. Shilton**, High dimensional bayesian optimization using dropout. *In Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. 2017*a*. URL https://doi.org/10.24963/ijcai.2017/291.

144. **Li, L.**, **K. Jamieson**, **G. DeSalvo**, **A. Rostamizadeh**, and **A. Talwalkar** (2017*b*). Hyperband: A novel bandit-based approach to hyperparameter optimization. *J. Mach.*

Learn. Res., **18**(1), 6765–6816. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=3122009.3242042.

145. **Liao, X.**, **Y. Xue**, and **L. Carin**, Logistic regression with an auxiliary data source. *In Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05. ACM, New York, NY, USA, 2005. ISBN 1-59593-180-5. URL http://doi.acm.org/10.1145/1102351.1102415.

146. **Lijoi, A.** and **I. Pruenster** (2009). Models beyond the Dirichlet process. ICER Working Papers - Applied Mathematics Series 23-2009, ICER - International Centre for Economic Research. URL https://ideas.repec.org/p/icr/wpmath/23-2009.html.

147. **Lim, M.** and **T. Hastie** (2015). Learning interactions via hierarchical group-lasso regularization. *J Comput Graph Stat*, **24**(3), 627–654. ISSN 1061-8600. URL https://www.ncbi.nlm.nih.gov/pubmed/26759522. 26759522[pmid].

148. **Lin, J.**, **C. Zhong**, **D. Hu**, **C. Rudin**, and **M. Seltzer**, Generalized and scalable optimal sparse decision trees. *In* **H. D. III** and **A. Singh** (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR, 2020. URL https://proceedings.mlr.press/v119/lin20g.html.

149. **Lipton, Z. C.** (2018). The mythos of model interpretability. *Queue*, **16**(3), 30:31–30:57. ISSN 1542-7730. URL http://doi.acm.org/10.1145/3236386.3241340.

150. **Liu, F. T.**, **K. M. Ting**, and **Z.-H. Zhou**, Isolation forest. *In 2008 Eighth IEEE International Conference on Data Mining*. 2008.

151. **Lorraine, J.**, **P. Vicol**, and **D. Duvenaud**, Optimizing millions of hyperparameters by implicit differentiation. *In* **S. Chiappa** and **R. Calandra** (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*. PMLR, 2020. URL https://proceedings.mlr.press/v108/lorraine20a.html.

152. **Lou, Y.**, **R. Caruana**, **J. Gehrke**, and **G. Hooker**, Accurate intelligible models with pairwise interactions. *In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2174-7. URL http://doi.acm.org/10.1145/2487575.2487579.

153. **Lu, P.**, **A. Ghaddar**, **A. Rashid**, **M. Rezagholizadeh**, **A. Ghodsi**, and **P. Langlais**, RW-KD: Sample-wise loss terms re-weighting for knowledge distillation. *In Findings of the Association for Computational Linguistics: EMNLP 2021*. Association for Computational Linguistics, Punta Cana, Dominican Republic, 2021. URL https://aclanthology.org/2021.findings-emnlp.270.

154. **Lundberg, S. M.**, **G. G. Erion**, and **S.-I. Lee** (2018). Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*.

155. **Lundberg, S. M.** and **S.-I. Lee**, A unified approach to interpreting model predictions. *In* **I. Guyon**, **U. V. Luxburg**, **S. Bengio**, **H. Wallach**, **R. Fergus**, **S. Vishwanathan**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc., 2017, 4765–4774. URL http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

156. **Ma, L.**, **J. Cui**, and **B. Yang**, Deep neural architecture search with deep graph bayesian optimization. *In 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. 2019.

157. **Malherbe, C.** and **N. Vayatis**, Global optimization of Lipschitz functions. *In* **D. Precup** and **Y. W. Teh** (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*. PMLR, 2017. URL https://proceedings.mlr.press/v70/malherbe17a.html.

158. **Malkomes, G.** and **R. Garnett**, Automating bayesian optimization with bayesian optimization. *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, 5984–5994. URL http://papers.nips.cc/paper/7838-automating-bayesian-optimization-with-bayesian-optimization.pdf.

159. **McCloskey, J. W.** (1965). *A model for the distribution of individuals by species in an environment.*. Ph.D. thesis, Michigan State University.

160. **Menon, A. K.**, **A. S. Rawat**, **S. Kumar**, **S. Reddi**, and **S. Kim**, A statistical perspective on distillation. *In International Conference on Machine Learning (ICML) 2021*. 2021.

161. **Merrill, E.**, **A. Fern**, **X. Fern**, and **N. Dolatnia** (2021). An empirical study of bayesian optimization: Acquisition versus partition. *Journal of Machine Learning Research*, **22**(4), 1–25. URL http://jmlr.org/papers/v22/18-220.html.

162. **Michie, D.**, **D. J. Spiegelhalter**, **C. C. Taylor**, and **J. Campbell** (eds.), *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, USA, 1995. ISBN 013106360X.

163. **Mihalkova, L.** and **R. Mooney**, Transfer learning with markov logic networks. *In Proceedings of the ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning*. Pittsburgh, PA, 2006. URL http://www.cs.utexas.edu/users/ai-lab/?mihalkova:icml-wkshp06.

164. **Miller, T.** (2019). Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, **267**, 1–38. ISSN 0004-3702. URL https://www.sciencedirect.com/science/article/pii/S0004370218305988.

165. **Ming, Y.**, **H. Qu**, and **E. Bertini** (2019). Rulematrix: Visualizing and understanding classifiers with rules. *IEEE Transactions on Visualization and Computer Graphics*, **25**(1), 342–352.

166. **Mita, G.**, **P. Papotti**, **M. Filippone**, and **P. Michiardi**, Libre: Learning interpretable boolean rule ensembles. *In* **S. Chiappa** and **R. Calandra** (eds.), *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*. PMLR, 2020. URL https://proceedings.mlr.press/v108/mita20a.html.

167. **Mohammad, R. M.**, **F. Thabtah**, and **L. McCluskey**, An assessment of features related to phishing websites using an automated technique. *In 2012 International Conference for Internet Technology and Secured Transactions*. 2012. ISSN null.

168. **Molnar, C.**, *Interpretable Machine Learning*. 2022, 2 edition. URL https://christophm.github.io/interpretable-ml-book.

169. **Mood, C.** (2010). Logistic regression : Why we cannot do what we think we can do, and what we can do about it. *European Sociological Review*, **26**(1), 67–82.

170. **Moshkovitz, M.**, **S. Dasgupta**, **C. Rashtchian**, and **N. Frost**, Explainable k-means and k-medians clustering. *In* **H. D. III** and **A. Singh** (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*. PMLR, 2020. URL https://proceedings.mlr.press/v119/moshkovitz20a.html.

171. **Munteanu, A.** and **C. Schwiegelshohn** (2018). Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI - Künstliche Intelligenz*, **32**(1), 37–53. ISSN 1610-1987. URL https://doi.org/10.1007/s13218-017-0519-3.

172. **Murdoch, W. J.**, **C. Singh**, **K. Kumbier**, **R. Abbasi-Asl**, and **B. Yu** (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, **116**(44), 22071–22080. ISSN 0027-8424. URL https://www.pnas.org/content/116/44/22071.

173. **Murphy, K. P.**, *Machine learning: a probabilistic perspective*. MIT press, 2012.

174. **Nayebi, A.**, **A. Munteanu**, and **M. Poloczek**, A framework for Bayesian optimization in embedded subspaces. *In* **K. Chaudhuri** and **R. Salakhutdinov** (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*. PMLR, Long Beach, California, USA, 2019. URL http://proceedings.mlr.press/v97/nayebi19a.html.

175. **Nori, H.**, **S. Jenkins**, **P. Koch**, and **R. Caruana** (2019). Interpretml: A unified framework for machine learning interpretability. *arXiv preprint arXiv:1909.09223*. URL https://interpret.ml.

176. **Ohlssen, D. I.**, **L. D. Sharples**, and **D. J. Spiegelhalter** (2007). Flexible random-effects models using bayesian semi-parametric models: applications to institutional comparisons. *Statistics in Medicine*, **26**(9), 2088–2112. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/sim.2666.

177. **Olkin, I.** and **T. Trikalinos** (2014). Constructions for a bivariate beta distribution. *Statistics & Probability Letters*, **96**.

178. **Ozaki, Y.**, **Y. Tanigaki**, **S. Watanabe**, and **M. Onishi**, Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. *In Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450371285. URL https://doi.org/10.1145/3377930.3389817.

179. **Pan, J.-Y.**, **H.-J. Yang**, **C. Faloutsos**, and **P. Duygulu**, Automatic multimedia cross-modal correlation discovery. *In Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04. ACM, New York, NY, USA, 2004. ISBN 1-58113-888-1. URL http://doi.acm.org/10.1145/1014052.1014135.

180. **Pan, S. J.** and **Q. Yang** (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, **22**(10), 1345–1359. ISSN 1041-4347.

181. **Parsopoulos, K. E.** and **M. N. Vrahatis**, Particle swarm optimizer in noisy and continuously changing environments. *In M.H. Hamza (Ed.), Arti cial Intelligence and Soft Computing, IASTED/ACTA*. IASTED/ACTA Press, 2001.

182. **Paschke, F.**, **C. Bayer**, **M. Bator**, **U. Mönks**, **A. Dicks**, **O. Enge-Rosenblatt**, and **V. Lohweg**, Sensorlose zustandsüberwachung an synchronmotoren. *In Proceedings of Computational Intelligence Workshop*. 2013.

183. **Pearson, K.** (1894). Contributions to the mathematical theory of evolution. *Philosophical Transactions of the Royal Society of London. A*, **185**, 71–110. ISSN 02643820. URL http://www.jstor.org/stable/90667.

184. **Pedregosa, F.**, Hyperparameter optimization with approximate gradient. *In Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16. JMLR.org, 2016.

185. **Pedregosa, F.**, **G. Varoquaux**, **A. Gramfort**, **V. Michel**, **B. Thirion**, **O. Grisel**, **M. Blondel**, **P. Prettenhofer**, **R. Weiss**, **V. Dubourg**, **J. Vanderplas**, **A. Passos**, **D. Cournapeau**, **M. Brucher**, **M. Perrot**, and **E. Duchesnay** (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, **12**, 2825–2830.

186. **Perrone, V.**, **R. Jenatton**, **M. W. Seeger**, and **C. Archambeau**, Scalable hyperparameter transfer learning. *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, 6845–6855. URL http://papers.nips.cc/paper/7917-scalable-hyperparameter-transfer-learning.pdf.

187. **Perry, W. L.**, **B. McInnis**, **C. C. Price**, **S. Smith**, and **J. S. Hollywood**, *Predictive Policing: The Role of Crime Forecasting in Law Enforcement Operations*. RAND Corporation, Santa Monica, CA, 2013.

188. **Pitman, J.** and **M. Yor** (1997). The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *Ann. Probab.*, **25**(2), 855–900. URL https://doi.org/10.1214/aop/1024404422.

189. **Platt, J.**, Fast training of support vector machines using sequential minimal optimization. *In Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998. URL https://www.microsoft.com/en-us/research/publication/fast-training-of-support-vector-machines-using-sequential-minimal-optimization/.

190. **Platt, J. C.**, Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *In ADVANCES IN LARGE MARGIN CLASSIFIERS*. MIT Press, 1999.

191. **Plumb, G.**, **D. Molitor**, and **A. S. Talwalkar**, Model agnostic supervised local explanations. *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/b495ce63ede0f4efc9eec62cb947c162-Paper.pdf.

192. **Poursabzi-Sangdeh, F.**, **D. Goldstein**, **J. Hofman**, **J. Wortman Vaughan**, and **H. Wallach**, Manipulating and measuring model interpretability. *In CHI 2021*. 2021. URL https://www.microsoft.com/en-us/research/publication/manipulating-and-measuring-model-interpretability/.

193. **Prokhorov, D.** (2001). IJCNN 2001 Neural Network Competition. http://www.geocities.ws/ijcnn/nnc_ijcnn01.pdf.

194. **Pröllochs, N.**, **S. Feuerriegel**, and **D. Neumann**, Learning interpretable negation rules via weak supervision at document level: A reinforcement learning approach. *In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, Minneapolis, Minnesota, 2019. URL https://www.aclweb.org/anthology/N19-1038.

195. **Quinlan, J. R.** (1990). Learning logical definitions from relations. *Machine Learning*, **5**(3), 239–266. ISSN 1573-0565. URL https://doi.org/10.1023/A:1022699322624.

196. **Quinlan, J. R.**, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

197. **Quinlan, J. R.** (2004). C5.0. https://rulequest.com/.

198. **Ram, P.** and **A. G. Gray**, Density estimation trees. *In Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11. Association for Computing Machinery, New York, NY, USA, 2011. ISBN 9781450308137. URL https://doi.org/10.1145/2020408.2020507.

199. **Rana, S.**, **C. Li**, **S. Gupta**, **V. Nguyen**, and **S. Venkatesh**, High dimensional Bayesian optimization with elastic Gaussian process. *In* **D. Precup** and **Y. W. Teh** (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*. PMLR, International Convention Centre, Sydney, Australia, 2017. URL http://proceedings.mlr.press/v70/rana17a.html.

200. **Rao, D.** and **B. McMahan**, *Natural Language Processing with PyTorch*. O'Reilly, 2019. ISBN 978-1491978238. https://www.amazon.com/Natural-Language-Processing-PyTorch-Applications/dp/1491978236/ and https://github.com/joosthub/PyTorchNLPBook.

201. **Rasmussen, C. E.**, The infinite gaussian mixture model. *In Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99. MIT Press, Cambridge, MA, USA, 1999. URL http://dl.acm.org/citation.cfm?id=3009657.3009736.

202. **Regis, R. G.** and **C. A. Shoemaker** (2007). A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, **19**(4), 497–509. URL https://doi.org/10.1287/ijoc.1060.0182.

203. **Regis, R. G.** and **C. A. Shoemaker** (2009). Parallel stochastic global optimization using radial basis functions. *INFORMS Journal on Computing*, **21**(3), 411–426. URL https://doi.org/10.1287/ijoc.1090.0325.

204. **Regis, R. G.** and **C. A. Shoemaker** (2013). Combining radial basis function surrogates and dynamic coordinate search in high-dimensional expensive black-box optimization. *Engineering Optimization*, **45**(5), 529–555. URL https://doi.org/10.1080/0305215X.2012.687731.

205. **Ren, L.**, **L. Du**, **L. Carin**, and **D. Dunson** (2011). Logistic stick-breaking process. *J. Mach. Learn. Res.*, **12**(null), 203–239. ISSN 1532-4435.

206. **Ribeiro, M. T.**, **S. Singh**, and **C. Guestrin**, "Why Should I Trust You?": Explaining the predictions of any classifier. *In Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4232-2. URL http://doi.acm.org/10.1145/2939672.2939778.

207. **Ribeiro, M. T.**, **S. Singh**, and **C. Guestrin**, Anchors: High-precision model-agnostic explanations. 2018. URL https://aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16982.

208. **Rodriguez, J. J.**, **L. I. Kuncheva**, and **C. J. Alonso** (2006). Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Anal. Mach. Intell.*, **28**(10), 1619–1630. ISSN 0162-8828. URL https://doi.org/10.1109/TPAMI.2006.211.

209. **Romero, A.**, **N. Ballas**, **S. E. Kahou**, **A. Chassang**, **C. Gatta**, and **Y. Bengio**, Fitnets: Hints for thin deep nets. *In* **Y. Bengio** and **Y. LeCun** (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. 2015. URL http://arxiv.org/abs/1412.6550.

210. **Rudin, C.** (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, **1**(5), 206–215. ISSN 2522-5839. URL https://doi.org/10.1038/s42256-019-0048-x.

211. **Rudin, C.**, **C. Wang**, and **B. Coker** (2020). The age of secrecy and unfairness in recidivism prediction. *Harvard Data Science Review*, **2**(1). URL https://hdsr.mitpress.mit.edu/pub/7z10o269. Https://hdsr.mitpress.mit.edu/pub/7z10o269.

212. **Sanh, V.**, **L. Debut**, **J. Chaumond**, and **T. Wolf** (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, **abs/1910.01108**. URL http://arxiv.org/abs/1910.01108.

213. **Santhiappan, S.**, **J. Chelladurai**, and **B. Ravindran**, A novel topic modeling based weighting framework for class imbalance learning. *In Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, CoDS-COMAD '18. Association for Computing Machinery, New York, NY, USA, 2018. ISBN 9781450363419. URL https://doi.org/10.1145/3152494.3152496.

214. **Santurkar, S.**, **D. Tsipras**, **A. Ilyas**, and **A. Madry**, How does batch normalization help optimization? *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/905056c1ac1dad141560467e0a99e1cf-Paper.pdf.

215. **Scheffer, T.**, **C. Decomain**, and **S. Wrobel**, Active hidden markov models for information extraction. *In Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, IDA '01. Springer-Verlag, London, UK, UK, 2001. ISBN 3-540-42581-0. URL http://dl.acm.org/citation.cfm?id=647967.741626.

216. **Scholbeck, C. A.**, **C. Molnar**, **C. Heumann**, **B. Bischl**, and **G. Casalicchio**, Sampling, intervention, prediction, aggregation: A generalized framework for model-agnostic interpretations. *In* **P. Cellier** and **K. Driessens** (eds.), *Machine Learning and Knowledge Discovery in Databases*. Springer International Publishing, Cham, 2020. ISBN 978-3-030-43823-4.

217. **Selvaraju, R. R.**, **M. Cogswell**, **A. Das**, **R. Vedantam**, **D. Parikh**, and **D. Batra**, Grad-cam: Visual explanations from deep networks via gradient-based localization. *In 2017 IEEE International Conference on Computer Vision (ICCV)*. 2017. ISSN 2380-7504.

218. **Sethuraman, J.** (1994). A constructive definition of Dirichlet priors. *Statistica Sinica*, **4**, 639–650.

219. **Settles, B.** (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison. URL http://axon.cs.byu.edu/~martinez/classes/778/Papers/settles.activelearning.pdf.

220. **Setzu, M.**, **R. Guidotti**, **A. Monreale**, **F. Turini**, **D. Pedreschi**, and **F. Giannotti** (2021). Glocalx - from local to global explanations of black box ai models. *Artificial Intelligence*, **294**, 103457. ISSN 0004-3702. URL https://www.sciencedirect.com/science/article/pii/S0004370221000084.

221. **Shahriari, B.**, **K. Swersky**, **Z. Wang**, **R. P. Adams**, and **N. de Freitas** (2016). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, **104**(1), 148–175. ISSN 0018-9219.

222. **Sharchilev, B.**, **Y. Ustinovskiy**, **P. Serdyukov**, and **M. de Rijke**, Finding influential training samples for gradient boosted decision trees. *In* **J. Dy** and **A. Krause** (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*. PMLR, 2018. URL http://proceedings.mlr.press/v80/sharchilev18a.html.

223. **Shrikumar, A.**, **P. Greenside**, and **A. Kundaje**, Learning important features through propagating activation differences. *In* **D. Precup** and **Y. W. Teh** (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*. PMLR, International Convention Centre, Sydney, Australia, 2017. URL http://proceedings.mlr.press/v70/shrikumar17a.html.

224. **Simonyan, K.**, **A. Vedaldi**, and **A. Zisserman**, Deep inside convolutional networks: Visualising image classification models and saliency maps. *In Workshop at International Conference on Learning Representations*. 2014.

225. **Simonyan, K.** and **A. Zisserman**, Very deep convolutional networks for large-scale image recognition. *In International Conference on Learning Representations*. 2015.

226. **Slack, D.**, **S. Hilgard**, **E. Jia**, **S. Singh**, and **H. Lakkaraju**, Fooling lime and shap: Adversarial attacks on post hoc explanation methods. *In Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20. Association for Computing Machinery, New York, NY, USA, 2020. ISBN 9781450371100. URL https://doi.org/10.1145/3375627.3375830.

227. **Slack, D. Z.**, **S. Hilgard**, **S. Singh**, and **H. Lakkaraju**, Reliable post hoc explanations: Modeling uncertainty in explainability. *In* **A. Beygelzimer**, **Y. Dauphin**, **P. Liang**, and **J. W. Vaughan** (eds.), *Advances in Neural Information Processing Systems*. 2021. URL https://openreview.net/forum?id=rqfq0CYIekd.

228. **Snoek, J.**, **H. Larochelle**, and **R. P. Adams**, Practical bayesian optimization of machine learning algorithms. *In* **F. Pereira**, **C. J. C. Burges**, **L. Bottou**, and **K. Q. Weinberger** (eds.), *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, 2951–2959. URL http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf.

229. **Snoek, J.**, **O. Rippel**, **K. Swersky**, **R. Kiros**, **N. Satish**, **N. Sundaram**, **M. M. A. Patwary**, **P. Prabhat**, and **R. P. Adams**, Scalable bayesian optimization using deep neural networks. *In Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045349.

230. **Song, J.**, **L. Yu**, **W. Neiswanger**, and **S. Ermon**, A general recipe for likelihood-free bayesian optimization. *In International Conference on Machine Learning*. 2022.

231. **Souza, A.**, **L. Nardi**, **L. B. Oliveira**, **K. Olukotun**, **M. Lindauer**, and **F. Hutter**, Bayesian optimization with a prior for the optimum. *In* **N. Oliver**, **F. Pérez-Cruz**, **S. Kramer**, **J. Read**, and **J. A. Lozano** (eds.), *Machine Learning and Knowledge Discovery in Databases. Research Track*. Springer International Publishing, Cham, 2021. ISBN 978-3-030-86523-8.

232. **Springenberg, J. T.**, **A. Klein**, **S. Falkner**, and **F. Hutter**, Bayesian optimization with robust bayesian neural networks. *In* **D. Lee**, **M. Sugiyama**, **U. Luxburg**, **I. Guyon**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper/2016/file/a96d3afec184766bfeca7a9f989fc7e7-Paper.pdf.

233. **Sturges, H. A.** (1926). The choice of a class interval. *Journal of the American Statistical Association*, **21**(153), 65–66. URL https://doi.org/10.1080/01621459.1926.10502161.

234. **Teh, Y. W.** (2010). Dirichlet process. https://www.stats.ox.ac.uk/~teh/research/npbayes/Teh2010a.pdf.

235. **Tenney, I.**, **J. Wexler**, **J. Bastings**, **T. Bolukbasi**, **A. Coenen**, **S. Gehrmann**, **E. Jiang**, **M. Pushkarna**, **C. Radebaugh**, **E. Reif**, and **A. Yuan**, The language interpretability tool: Extensible, interactive visualizations and analysis for NLP models. *In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 2020. URL https://www.aclweb.org/anthology/2020.emnlp-demos.15.

236. **Thrun, S.** and **T. M. Mitchell**, Learning one more thing. *In IJCAI*. 1994.

237. **Tiao, L. C.**, **A. Klein**, **M. W. Seeger**, **E. V. Bonilla**, **C. Archambeau**, and **F. Ramos**, Bore: Bayesian optimization by density-ratio estimation. *In* **M. Meila** and **T. Zhang** (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*. PMLR, 2021. URL https://proceedings.mlr.press/v139/tiao21a.html.

238. **Tibshirani, R.** (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, **58**(1), 267–288. ISSN 00359246. URL http://www.jstor.org/stable/2346178.

239. **Tjoa, E.** and **C. Guan** (2020). A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE Transactions on Neural Networks and Learning Systems*, 1–21.

240. **Torrey, L.** and **J. W. Shavlik**, Handbook of research on machine learning applications (Chapter 11). IGI Global, 2009.

241. **Turner, R.**, **D. Eriksson**, **M. McCourt**, **J. Kiili**, **E. Laaksonen**, **Z. Xu**, and **I. Guyon** (2021). Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *CoRR*, **abs/2104.10201**. URL https://arxiv.org/abs/2104.10201.

242. **Ustun, B.** and **C. Rudin** (2016). Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, **102**(3), 349–391. ISSN 1573-0565. URL https://doi.org/10.1007/s10994-015-5528-6.

243. **Uzilov, A. V.**, **J. M. Keegan**, and **D. H. Mathews** (2006). Detection of non-coding rnas on the basis of predicted secondary structure formation free energy change. *BMC bioinformatics*, **7**, 173–173. ISSN 1471-2105. URL https://www.ncbi.nlm.nih.gov/pubmed/16566836. 16566836[pmid].

244. **Vaswani, A.**, **N. Shazeer**, **N. Parmar**, **J. Uszkoreit**, **L. Jones**, **A. N. Gomez**, **u. Kaiser**, and **I. Polosukhin**, Attention is all you need. *In Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17. Curran Associates Inc., Red Hook, NY, USA, 2017. ISBN 9781510860964.

245. **Vig, J.**, **A. Madani**, **L. R. Varshney**, **C. Xiong**, **richard socher**, and **N. Rajani**, {BERT}ology meets biology: Interpreting attention in protein language models. *In International Conference on Learning Representations*. 2021. URL https://openreview.net/forum?id=YWtLZvLmud7.

246. **Wallace, E.**, **J. Tuyls**, **J. Wang**, **S. Subramanian**, **M. Gardner**, and **S. Singh**, AllenNLP Interpret: A framework for explaining predictions of NLP models. *In Empirical Methods in Natural Language Processing*. 2019. URL https://allennlp.org/interpret.

247. **Wang, A.**, **Y. Pruksachatkun**, **N. Nangia**, **A. Singh**, **J. Michael**, **F. Hill**, **O. Levy**, and **S. Bowman**, Superglue: A stickier benchmark for general-purpose language understanding systems. *In* **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/4496bf24afe7fab6f046bf4923da8de6-Paper.pdf.

248. **Wang, A.**, **A. Singh**, **J. Michael**, **F. Hill**, **O. Levy**, and **S. Bowman**, GLUE: A multi-task benchmark and analysis platform for natural language understanding. *In Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*. Association for Computational Linguistics, Brussels, Belgium, 2018*a*. URL https://www.aclweb.org/anthology/W18-5446.

249. **Wang, C.-C.**, **K. L. Tan**, **C.-T. Chen**, **Y.-H. Lin**, **S. S. Keerthi**, **D. Mahajan**, **S. Sundararajan**, and **C.-J. Lin** (2018*b*). Distributed newton methods for deep neural networks. *Neural Comput.*, **30**(6), 1673–1724. ISSN 0899-7667. URL https://doi.org/10.1162/neco_a_01088.

250. **Wang, T.**, Multi-value rule sets for interpretable classification with feature-efficient representations. *In* **S. Bengio**, **H. Wallach**, **H. Larochelle**, **K. Grauman**, **N. Cesa-Bianchi**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, 10835–10845. URL http://papers.nips.cc/paper/8281-multi-value-rule-sets-for-interpretable-classification-with-feature-efficient-representations.pdf.

251. **Wang, Z.**, **B. Shakibi**, **L. Jin**, and **N. Freitas**, Bayesian Multi-Scale Optimistic Optimization. *In* **S. Kaski** and **J. Corander** (eds.), *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, volume 33 of *Proceedings of Machine Learning Research*. PMLR, Reykjavik, Iceland, 2014. URL https://proceedings.mlr.press/v33/wang14d.html.

252. **Wang, Z.**, **M. Zoghi**, **F. Hutter**, **D. Matheson**, and **N. De Freitas**, Bayesian optimization in high dimensions via random embeddings. *In Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13. AAAI Press, 2013. ISBN 9781577356332.

253. **Wang, Z. J.**, **A. Kale**, **H. Nori**, **P. Stella**, **M. Nunnally**, **D. H. Chau**, **M. Vorvoreanu**, **J. W. Vaughan**, and **R. Caruana** (2021). Gam changer: Editing generalized additive models with interactive visualization. URL https://arxiv.org/abs/2112.03245.

254. **Weiss, K.**, **T. M. Khoshgoftaar**, and **D. Wang** (2016). A survey of transfer learning. *Journal of Big Data*, **3**(1), 9. ISSN 2196-1115. URL https://doi.org/10.1186/s40537-016-0043-6.

255. **White, C.**, **W. Neiswanger**, and **Y. Savani**, Bananas: Bayesian optimization with neural architectures for neural architecture search. *In Proceedings of the AAAI Conference on Artificial Intelligence*. 2021.

256. **Wilcoxon, F.** (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, **1**(6), 80–83. ISSN 00994987. URL http://www.jstor.org/stable/3001968.

257. **Wu, Z.** and **M. Palmer**, Verbs semantics and lexical selection. *In Proceedings of the 32Nd Annual Meeting on Association for Computational Linguistics*, ACL '94. Association for Computational Linguistics, Stroudsburg, PA, USA, 1994. URL https://doi.org/10.3115/981732.981751.

258. **Yang, Y.** and **M. Loog** (2018). A benchmark and comparison of active learning for logistic regression. *Pattern Recognit.*, **83**, 401–415. URL https://doi.org/10.1016/j.patcog.2018.06.004.

259. **Yeh, C.-K.**, **C.-Y. Hsieh**, **A. Suggala**, **D. I. Inouye**, and **P. K. Ravikumar**, On the (in)fidelity and sensitivity of explanations. *In* **H. Wallach**, **H. Larochelle**, **A. Beygelzimer**, **F. d'Alché-Buc**, **E. Fox**, and **R. Garnett** (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/a7471fdc77b3435276507cc8f2dc2569-Paper.pdf.

260. **Yeo, I.-K.** and **R. A. Johnson** (2000). A new family of power transformations to improve normality or symmetry. *Biometrika*, **87**(4), 954–959. ISSN 00063444. URL http://www.jstor.org/stable/2673623.

261. **Yoon, J.**, **J. Jordon**, and **M. van der Schaar**, INVASE: Instance-wise variable selection using neural networks. *In International Conference on Learning Representations*. 2019. URL https://openreview.net/forum?id=BJg_roAcK7.

262. **Yoon, J.**, **W. R. Zame**, **A. Banerjee**, **M. Cadeiras**, **A. M. Alaa**, and **M. van der Schaar** (2018). Personalized survival predictions via trees of predictors: An application to cardiac transplantation. *PLOS ONE*, **13**(3), 1–19. URL https://doi.org/10.1371/journal.pone.0194985.

263. **Zhang, H.**, **Z. Hu**, **W. Qin**, **M. Xu**, and **M. Wang** (2021). Adversarial co-distillation learning for image recognition. *Pattern Recognition*, **111**, 107659. ISSN 0031-3203. URL https://www.sciencedirect.com/science/article/pii/S0031320320304623.

264. **Zhang, L. A.**, **J. Xu**, **D. Gold**, **J. Hagen**, **A. K. Kochhar**, **A. J. Lohn**, and **O. A. Osoba**, *Air Dominance Through Machine Learning: A Preliminary Exploration of Artificial Intelligence?Assisted Mission Planning*. RAND Corporation, Santa Monica, CA, 2020*a*.

265. **Zhang, W.**, **X. Chen**, **Y. Liu**, and **Q. Xi** (2020*b*). A distributed storage and computation k-nearest neighbor algorithm based cloud-edge computing for cyber-physical-social systems. *IEEE Access*, **8**, 50118–50130.

LIST OF PAPERS BASED ON THESIS

1. Ghose, A., and Ravindran, B. (2020) *Interpretability with Accurate Small Models*. In Frontiers in Artificial Intelligence, section: Machine Learning and Artificial Intelligence, Vol 3, February 2020. DOI: 10.3389/frai.2020.00003, https://www.frontiersin.org/articles/10.3389/frai.2020.00003/pdf.

2. Ghose, Abhishek, and Balaraman Ravindran. *Learning Interpretable Models Using an Oracle.* ArXiv:1906.06852 [Cs, Stat], Jan. 2022. arXiv.org, http://arxiv.org/abs/1906.06852.

3. Ghose, Abhishek. *Accurate Small Models using Adaptive Sampling.* ArXiv:2210.03921 [cs.LG], Oct. 2022. arXiv.org, https://arxiv.org/abs/2210.03921

Software(s): *compactem* (Ghose, 2020), https://compactem.readthedocs.io/en/latest/.

DOCTORAL COMMITTEE

Chairperson: Dr. P. Sreenivasa Kumar
Professor
Computer Science & Engineering
Indian Institute of Technology Madras

Research Advisor: Dr. Balaraman Ravindran
Professor
Computer Science & Engineering
Indian Institute of Technology Madras

Members: Dr. Deepak Khemani
Professor
Computer Science & Engineering
Indian Institute of Technology Madras

Dr. Mitesh Khapra
Associate Professor
Computer Science & Engineering
Indian Institute of Technology Madras

Dr.Palaniappan Ramu
Associate Professor
Department of Engineering Design
Indian Institute of Technology Madras

CURRICULUM VITAE

1. NAME : Abhishek Ghose

2. DATE OF BIRTH : 29^{th} June, 1982

3. PERMANENT ADDRESS : S-18, Mana Jardin
HN Halli Lake Road, Off Sarjapur Main Road,
Bengaluru - 560035,
Karnataka
Email: abhishek.ghose.82@gmail.com
Phone: +91-9686809809

4. EDUCATIONAL QUALIFICATIONS

Master of Science, Engineering (MS)

Year of Completion	:	2012
Institution	:	Indian Institute of Technology, Madras
Specialization	:	Computer Science and Engineering

Bachelor of Science, Engineering (B.Sc. Engg.)

Year of Completion	:	2004
Institution	:	National Institute of Technology, Jamshedpur
Specialization	:	Mechanical Engineering