Distilling Interpretable Models into Human-Readable Code

Walker Ravina*
walkerravina@google.com
Google

Ethan Sterling
esterling@google.com
Google

Olexiy Oryeshko
olexiy@google.com
Google

Nathan Bell
nathanbell@google.com
Google

Honglei Zhuang
hlz@google.com
Google

Xuanhui Wang
xuanhui@google.com
Google

Yonghui Wu
yonghui@google.com
Google

Alexander Grushetsky
grushetsky@google.com
Google

ABSTRACT

The goal of model distillation is to faithfully transfer teacher model knowledge to a model which is faster, more generalizable, more interpretable, or possesses other desirable characteristics. Human-readability is an important and desirable standard for machine-learned model interpretability. Readable models are transparent and can be reviewed, manipulated, and deployed like traditional source code. As a result, such models can be improved outside the context of machine learning and manually edited if desired. Given that directly training such models is difficult, we propose to train interpretable models using conventional methods, and then distill them into concise, human-readable code.

The proposed distillation methodology approximates a model's univariate numerical functions with piecewise-linear curves in a localized manner. The resulting curve model representations are accurate, concise, human-readable, and well-regularized by construction. We describe a piecewise-linear curve-fitting algorithm that produces high-quality results efficiently and reliably across a broad range of use cases. We demonstrate the effectiveness of the overall distillation technique and our curve-fitting algorithm using three publicly available datasets COMPAS, FICO, and MSLR-WEB30K.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning approaches**.

KEYWORDS

Model distillation, human readable, piecewise-linear curves

1 INTRODUCTION

Univariate functions are widely used in interpretable models. For example, in Generalized Additive Models (GAMs) [11] the model is a sum of univariate shape functions,

$$M = f_0 + f_1(x_1) + f_2(x_2) + f_3(x_3) + \cdots + f_n(x_n).$$

Lou *et al.* [18] showed that adding a limited number of pairwise feature interactions allows GAM style additive models to capture a significant fraction of the accuracy of a fully-interacting model. In many cases of interest, such feature interactions are intuitively captured with products of univariate functions,

$$g_1(c_1) \cdot f_1(x_1) + g_2(c_2) \cdot f_2(x_2) + \ldots,$$

or products of groups of features,

$$(g_{1,1}(c_1) + g_{1,2}(c_2)) \cdot f_1(x_1) + (g_{2,1}(c_1) + g_{2,2}(c_2)) \cdot f_2(x_2) + \ldots,$$

where the magnitude of one feature (i.e. x_i) is modulated by a function of another "context" feature (i.e. c_i) [31]. In other cases, the interaction amongst features is adequately approximated by additive models of univariate functions nested within univariate functions,

$$f(x_1, x_2, x_3) \approx g_1(f_{1,1}(x_1) + f_{1,2}(x_2) + f_{1,3}(x_3)),$$

or

$$f(x_1, x_2, x_3) \approx g_1(f_{1,1}(x_1) + f_{1,2}(x_2) + f_{1,3}(x_3)) + g_2(f_{2,1}(x_1) + f_{2,2}(x_2) + f_{2,3}(x_3)) + \ldots,$$

where the outer function captures nonlinear behavior [7]. Indeed, the Kolmogorov–Arnold representation theorem [16, 26] guarantees that every continuous multivariate function of n inputs can be represented as

$$f(x_1, \ldots, x_n) = \sum_{i=0}^{2n} g_i \left(\sum_{j=1}^{n} f_{i,j}(x_j) \right).$$

In practice a single outer function is often sufficient, yielding an interpretable model.

In principle one can directly optimize concise, human-readable function representations (e.g. piecewise-linear curves or splines) containing a small number of variables. However, directly optimizing such representation often yields less accurate models than alternative model representations. For example, Lou et al [17] showed that learning spline GAMs is less accurate than learning bagged boosted decision forest GAMs. Our experiments show similar results for directly optimizing GAMs composed of piecewise-linear curves using Stochastic Gradient Descent (SGD) methods. Broadly speaking, the model representations that are practical to use during model optimization are not human-readable. This holds even when there exists a simpler model with a concise, human-readable form that provides comparable accuracy. This is similar to how relatively small decision forests or neural networks can match the accuracy of much larger ensembles when distilled from them, but

*Corresponding author.

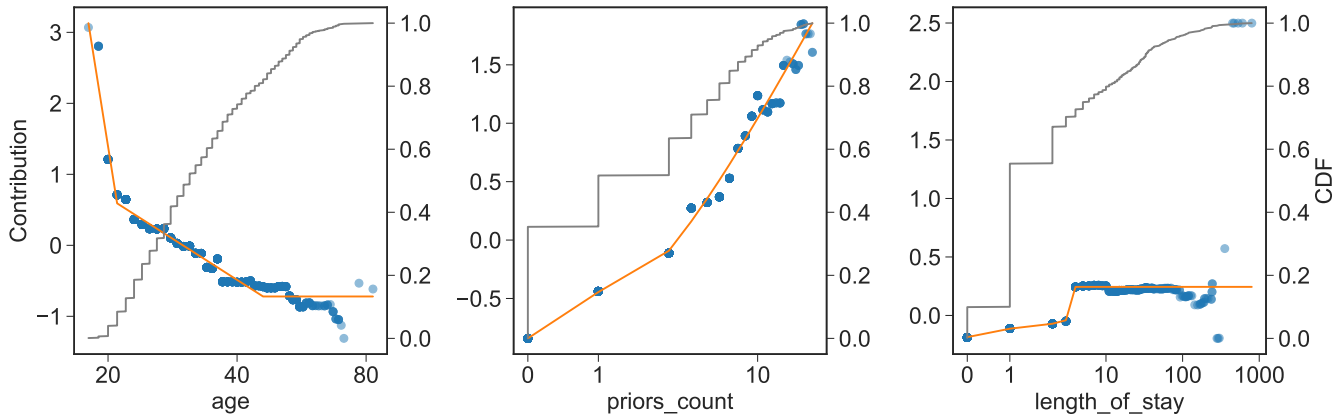

Figure 1.1: Shape plots for numerical features from models learned on the COMPAS dataset. The GAM forest model is shown in blue dots, while its distillation into the two-segment curve model is shown in orange lines (both map to the left Y axis). Cumulative distribution functions of the corresponding signals are shown in grey (right Y axis).

Listing 1.1: Code for a distilled COMPAS model

```
score = sum([
  PWLCurve("age", [(18, 3.13), (21, 0.5914),
    (46, -0.7206)], fx="log"),
  PWLCurve("priors_count", [(0, -0.8415), (1, -0.4452),
      (38, 2.146)], fx="log1p"),
  PWLCurve("length_of_stay", [(0, -0.1855),
    (3, -0.04099), (4, 0.2443)], fx="log1p"),
  EnumCurve("c_charge_degree", {1: 0.0198, 2: -0.0384}),
  ## ... other features ...
])
```

not when trained directly [5, 12]. To address this issue, we propose to distill models into readable representations in a separate process after model optimization, decoupling the initial, learned model representation from the final, published model representation. For example, the proposed distillation methodology can be applied to additive models trained using bagged boosted decision trees, as demonstrated in our results. Similarly, the technique can be applied to additive neural nets [2], as demonstrated by Zhuang et al [31]. Listing 1.1 and Figure 1.1 show textual and graphical representations of a model obtained by applying our approach to a decision forest GAM learned from the COMPAS dataset. In this paper we describe a technique for distilling models composed of univariate components into human readable representations. From here on, we will use "curves" to refer to piecewise-linear curves, "curve models" to refer to models where each component is a curve, and "code" to refer to the textual representations of curve models or curves. The rest of this paper is structured as follows.

First we motivate the benefits of using curve models. We then describe the localized distillation process and piecewise-linear approximation algorithm, sometimes referred to as segmented regression [29], for creating curve models. Lastly, we present experimental results on three publicly available datasets.

- The COMPAS dataset[1] is the result of a ProPublica investigation [4] into possible racial bias of the proprietary COMPAS model score for defendants in Broward county, Florida. The dataset has been studied extensively in the context of bias, fairness, and interpretability [8, 9, 14, 22].
- The FICO dataset [1] is composed of real-world anonymized credit applications along with risk scores.
- The MSLR-WEB30K dataset [19] is a widely used learning-to-rank benchmark dataset.

Both FICO and MSLR-WEB30K have been previously studied in the context of interpretability [2, 7, 18, 31]. In each case, we distill a decision forest GAM and evaluate the accuracy of the distilled curve models. The COMPAS and FICO datasets represent high-stakes domains [20] in which the benefits of curve models, discussed below, are particularly compelling. The MSLR-WEB30K is significantly larger both in number of features (136) and number of training examples (2,270,296). We use MSLR-WEB30K to compare our curve approximation algorithm to `pwlf` [13], a publicly available piecewise-linear curve-fitting library, on the basis of accuracy, robustness and efficiency. Furthermore the results from MSLR-WEB30K demonstrate that the accuracy of this approach is not limited to small datasets.

2 PIECEWISE-LINEAR CURVES

A piecewise linear curve (`PWLCurve`) is defined by a list of (x, y) control points through which the curve must pass. Between control points, output y values are determined by performing linear interpolation between neighboring control points. Beyond the leftmost or rightmost control points, output values are capped to the y-value of the neighboring control point. In most cases of interest 5 or 6 control points, defining 4 or 5 interior segments, is sufficient to capture the desired behavior.

We allow for an optional x-transformation, specified with the `fx` argument, to fit curves to data with different scales. When an x-transformation is present it is applied to the input value and x-values of all the control points, and then linear interpolation is performed in the transformed space. We support `identity` (default), `log`, `log1p` and `symlog1p` transformations. Here `symlog1p` is defined as

[1] https://github.com/propublica/compas-analysis

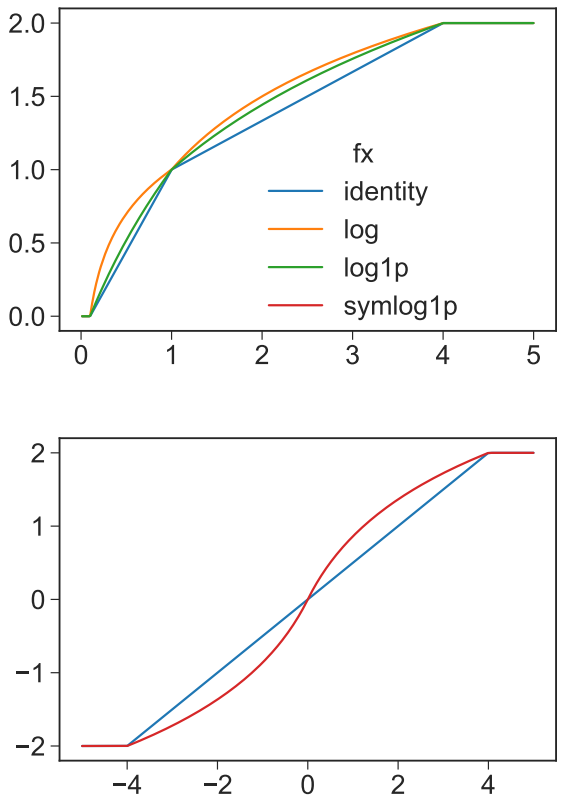

Figure 2.1: `PWLCurve` interpolation over the same control points using different x-transformations.

`sgn(x) * log1p(abs(x))` and is suitable for highly-variable features that take on both positive and negative values. Figure 2.1 illustrates two sets of curves with the same control points but different transformations.

Univariate categorical functions are represented by `EnumCurve`, which directly maps input values to outputs using a discrete mapping. In practice we round the floating point values of all `PWLCurve` coordinates and `EnumCurve` output values to a smaller number of digits for brevity. This rarely has a meaningful impact on accuracy, but produces more concise code.

3 BACKGROUND & MOTIVATION

Interpretable models are critical for high-stakes decisions [20] and provide many advantages over more complex model structures [6, 10]. In this section we explain how distilling interpretable models into curve models reinforces these benefits and addresses a variety of real-world engineering challenges. Here, one underlying theme is that distilling models into human-readable source code *reduces a novel machine learning problem to an established software engineering problem with an abundance of existing solutions.*

3.1 Greater Transparency

A model is transparent if it provides a textual or graphical representation that enables its behavior to be understood comprehensively [23]. One way in which the proposed method provides greater transparency is by simplifying graphical depictions of a model while retaining its essential characteristics. It is often argued, implicitly or explicitly, that the shape plots of an interpretable model are an *exact description* of the model and therefore provide a reliable way to understand the model. While this claim is narrowly true, it is misleading in general. Unless given specific guidance, humans will naturally discount certain fine-grained details of the plots when developing an understanding of the model. By distilling interpretable models to a concise representation, we discard extraneous characteristics and reduce the mental effort necessary to understand the model. For example, it is not immediately obvious what understanding an individual should derive from the shape plots of the `MSinceOldestTradeOpen`, `NumTotalTrades` and `PercentTradesNeverDelq` features in the initially-learned FICO model, shown in Figure 3.1. Indeed, different individuals may derive qualitatively different understandings from these graphical depictions. However, given the additional knowledge that the distilled curve model represented by the overlaid curves in Figure 3.1 has nearly identical accuracy, an observer can make much stronger inferences about the model's essential characteristics. Interpretability can be increased even further by imposing monotonicity constraints. We discuss the effect of such constraints in Section 6.3.

Clearly when distillation yields a simpler model with comparable accuracy we would say the distillation process has succeeded. However, instances where distillation yields a model with inferior accuracy warrant further investigation because the apparent "failure" can often be attributed to essential characteristics of the teacher model that were not successfully transferred to the student *because they violate a prescribed notion of human-interpretability.* While a complete discussion of this phenomenon is beyond the scope of this paper, the idea can be viewed as an extension of the use of structural constraints to define "interpretable" models, just now applied to the structure of individual functions in the model. Under this policy, if the accuracy of a candidate model cannot be reproduced using a predefined class of expressive, "human-scale" functions (e.g. curves with a small number of truncated control points) its transparency would be called into question.

3.2 Constructive Regularization

The proposed method can also be viewed as a post-hoc regularization process that is completely compatible with, and complementary to, optimization-based regularization techniques (e.g. L1/L2 penalties or monotonicity constraints). In the context of regularization, our emphasis on conciseness is aligned with the minimum description length principle [27] for model selection. Ustun and Rudin [23] applied similar reasoning to motivate linear models with small, integer-valued weights. The constrained description length of curves provides limited capacity for capturing idiosyncratic behavior. As a result, curve distillation successfully removes aberrations from teacher model functions. This regularization effect can be seen in Figure 1.1 and Figure 3.1. The fewer segments the greater the effect. To find the most concise curve model we can

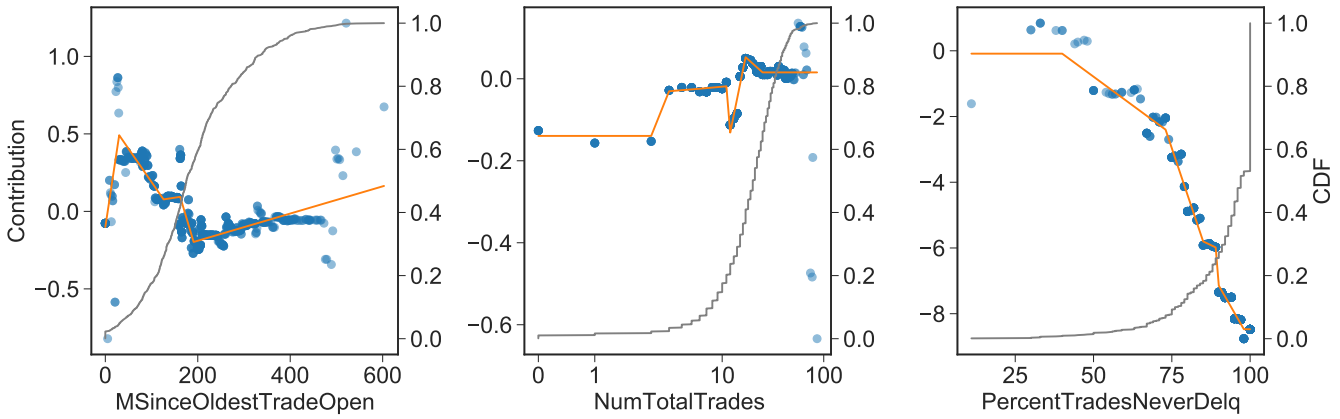

Figure 3.1: Shape plots for a GAM forest model (in blue dots) and 5 segment curve distillation (in orange lines) for the FICO dataset.

repeatedly apply the proposed method with decreasing number of control points. Naturally, the optimality of this approach is subject to the limitations and characteristics of our localized distillation methodology (see Section 4) and curve approximation algorithm (see Section 5). While it is difficult to directly compare models with different functional representations, comparing the length and readability of their corresponding code is instructive.

One practical advantage of curve-based regularization is that regularity is enforced by construction and the complexity of individual curves is readily apparent and quantifiable. Therefore, organizations that adopt curve models can set objective guidelines about model complexity that developers can anticipate when submitting model candidates for approval. Such guidelines can specify the maximum number of curve segments, maximum number of significant digits per curve control point, or monotonicity of the curve. Similar to the use of nothing-up-my-sleeve numbers in cryptography [28], curve models enable developers to preemptively address suspicions about potential weaknesses and constructively prove the robustness of a given model candidate. In general, standardizing development around curve models is a straightforward way for organizations to systematically enforce best practices, defend against common mistakes and pitfalls, and expedite model verification and approval. The accessible, readable nature of curve models enables organization members beyond engineers (e.g. executives, product managers, etc.) to participate in this approval process.

3.3 Readable, Editable Code

Code of curve models can be read, reviewed, merged and versioned like conventional source code. An example model for the COMPAS dataset is shown in Listing 1.1. One can understand how a curve model would behave under novel or extremal conditions by mentally "evaluating" the model under hypothetical "what if?" scenarios without the need for additional tools. Subjecting models to a traditional source code review process facilitates a more rigorous examination of the model's characteristics and greater accountability than is possible with non-readable models. Indeed, conducting "model review" through source code review ensures that the candidate model itself - not some separate, potentially inconsistent description or artifact of the model or how it was trained - is the subject of review. In the event that undesirable model behavior is discovered, the model's code may be directly edited to correct such issues. For example, in the case of the COMPAS model a user may wish to deliberately cap the contribution of features such as `priors_count` and `length_of_stay` features for legitimate policy reasons not captured by classification metrics such as AUC-ROC. The contribution of other features can be entirely removed. Agarwal *et al.* [2] discussed how such an approach of training with biased features and then removing them can potentially be better than simply training without biased features. This approach can prevent the model from extracting bias through other features which are correlated with biased ones.

Model transparency is essential in the context of high-stakes decisions [20] arising in criminal justice, finance, health care, and other areas. Providing the complete source of the model in simple, portable, human-readable code makes the models transparent. Compared to human-readable models produced by CORELS [3], which are expressed in universally-understandable if-then language, curve models sacrifice accessibility for greater expressiveness and general-purpose application.

3.4 Collaborative Model Development

Curve distillation is compatible with any algorithm or modeling technique that results in univariate functions. In the experiments section we apply the proposed technique to decision forest GAMs on several datasets. Previous work [31] successfully applied the proposed technique to GAMs learned via neural networks, as well as similar neural networks with limited interactions via multiplicative pairs. Organizing collaborative development around curve models enables engineers to apply a plurality of different tools, techniques, or platforms to optimize components of a (potentially large-scale) model. Engineers are free to choose a modeling approach that maximizes their productivity, similarly to how engineers use multiple IDEs, code formatters, or linters to collaboratively develop software.

Listing 3.1: A COMPAS model as a C++ function

```
double COMPAS(double age, double priors_count,
              double length_of_stay, int charge_degree,
              // ... other features ...
              ) {
  static auto age_curve = PWLCurve({{18, 3.13},
    {21, 0.5914}, {46, -0.7206}}, "log");
  static auto priors_count_curve = PWLCurve(
    {{0, -0.8415}, {1, -0.4452}, {38, 2.146}},"log1p");
  static auto length_of_stay_curve = PWLCurve(
    {{0, -0.1855}, {3, -0.04099}, {4, 0.2443}}, "log1p");
  static auto charge_degree_curve = EnumCurve({{1,
      0.0198}, {2, -0.0384}});
  // ... other features ...
  return (age_curve.Eval(age) +
    priors_count_curve.Eval(priors_count) +
    length_of_stay_curve.Eval(length_of_stay) +
    charge_degree_curve.Eval(charge_degree) +
    // ... other features ...
  );
}
```

Curve distillation can be viewed as a "format conversion" tool that translates an arbitrary and potentially exotic model representation into a fixed, agreed-upon vocabulary of human-readable building blocks.

3.5 Straightforward Deployment

Curve models are fast-to-evaluate and straightforward to deploy. Since evaluation requires minimal computation - just a handful of floating point operations per curve - curve models are well-suited for performance-critical applications. Curves are a portable, platform-agnostic representation that can be natively supported in a variety of languages or systems with little effort. For example, Listing 3.1 shows a C++ implementation of a COMPAS model with 2 segments. In general, curve models are straightforward to deploy because they offer a multitude of integration options. Curves can be embedded in configuration files, passed via CGI parameters, manually embedded into complex applications in a piecemeal fashion, systematically translated to a target representation, or evaluated by existing runtime systems with a few incremental extensions.

4 LOCALIZED DISTILLATION

Our distillation process takes two inputs: a teacher model containing one or more univariate functions, and a representative dataset (generally the training data). Our method differs from conventional distillation techniques in that we (1) distill each univariate function in isolation and (2) optimize for mean squared error (MSE) when approximating each univariate function. Specifically, each univariate function in the teacher model is evaluated on the dataset to produce representative (x, y) example pairs. For discrete categorical features we create a mapping where each unique x is mapped to the mean y. For numerical features, we produce a `PWLCurve` using the approximation algorithm described in Section 5. If the teacher model contains univariate functions nested within other univariate functions we replace the source functions in a bottom-up fashion. Otherwise, all non-nested functions can be approximated in parallel. The final model is constructed by replacing each original univariate function with its approximation.

Conventionally, model distillation involves a global optimization using the same (or at least similar) objective to the original teacher model training. This objective may differ from a point-wise MSE objective. For example ranking objectives often have pair-wise definitions. Why then do we advocate a localized optimization using a MSE objective in all circumstances? The primary answer is that, in the context of interpretable models, there is substantial value in maintaining a strong one-for-one correspondence between each source function and target function. Notably, this allows us to visualize each shape function in the teacher model against its corresponding curve replacement, to attribute distillation failures - data instances where the curve model is less accurate than the teacher model - to specific univariate functions, and to take remedial actions. For example, in the Figure 4.1 we can immediately tell that the shape functions of `AverageMInFile`, and `PercentInstallTrades` were well-approximated by a curve with only 2 segments while the one for `MSinceMostRecentDelq` was not. In the experiments section we show that the meaningful behavior of nearly all shape functions can be accurately captured by curves with three to five segments.

While a global optimization approach (i.e. optimizing the parameters of all curves in the target model simultaneously) using a problem-specific metric might produce a more accurate result, it is computationally more expensive and would lack the same one-to-one correspondence with the teacher model, making distillation failures more difficult to diagnose. Furthermore, if higher accuracy is desired, the output of the proposed distillation process can be used to initialize a global optimization of the curve model's parameters.

5 PIECEWISE-LINEAR CURVE APPROXIMATION

Given a univariate numerical function $f(x) \rightarrow y$, our goal is to produce a `PWLCurve` $c(x) \rightarrow y$ that faithfully approximates $f(x)$ by minimizing the $MSE(c(x), f(x))$ over sample data. Clearly, the accuracy of the overall distillation method depends critically on the accuracy of the individual curve approximations - i.e. how much metric loss is incurred when each $c(x)$ is substituted for the corresponding $f(x)$ in the trained model.

Additionally, the practical success of the methodology also depends on the robustness and efficiency of the approximation algorithm. To enable systematic use of curve distillation in model training pipelines, the approximation algorithm must run with minimal configuration. Complex hyperparameters pose a significant barrier to entry. We have designed `pwlfit`, our piecewise linear approximation algorithm, so that in practice users only need to consider the `num_segments` and `mono` (monotonicity) parameters. While `num_segments=5` segments and `mono=False` is sufficient to get high accuracy (as demonstrated by our experiments), it is desirable to investigate whether the model can be further simplified with fewer segments or with monotonicity restrictions. To facilitate such investigations it is important that distillation runs quickly (e.g. less than 1 second per function) which enables interactive analysis via Jupyter notebooks [15] or other tools. These practical considerations have informed various decisions in the design of `pwlfit`. In particular, we prefer an algorithm which quickly and reliably yields high accuracy

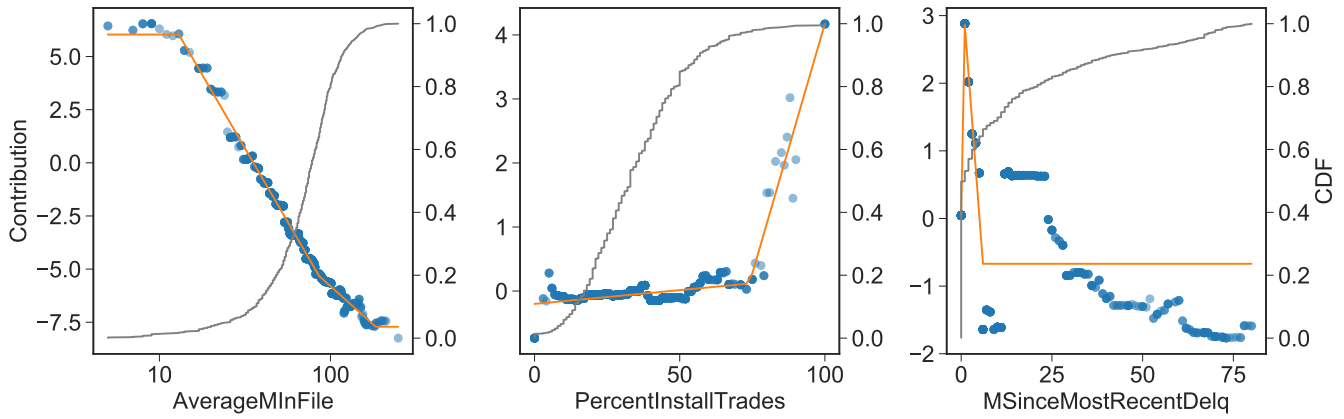

Figure 4.1: Shape plots for a GAM forest model (in blue dots) and 2-segment curve distillation (in orange lines) for the FICO dataset.

results with minimal configuration to one which sacrifices either of these practical considerations for marginal gains in accuracy.

In this section we will describe the salient characteristics and noteworthy features of `pwlfit`. We invite interested readers to consult the publicly-available source code of `pwlfit` [21], for additional details.

5.1 Algorithm

Given a list of $(x, y, weight)$ points and a desired number of segments k, we search for a curve to minimize mean squared error, MSE. A curve with k segments is characterized by its $k + 1$ control points – a set of x-knots and their corresponding y-knots. Given only the x-knots, we can solve a linear least squares expression for the optimal y-knots and the resulting error. Since we don't know the correct x-knots, we search through the space of possible x-knots and solve a least squares expression at each step to calculate the error. Here, `pwlf` [13] implements a similar approach. The provided code implements such a search, with different options and optimizations.

5.1.1 Initial Downsampling. For performance, we randomly downsample large datasets to approximately one million points before fitting. We downsample to reduce the cost of sorting, which dominates the runtime for large data. This downsampling imposes a negligible quality loss, because piecewise linear curves are simple models that don't need so many data points.

5.1.2 Knot Discretization. To further reduce runtime, we discretize the search space for x-knots. We choose `num_samples` x-values from the data, spaced equally by cumulative weight, and search over the combinations of x-knots from that sampled set of candidates. Using the default 100 samples, our candidates are the x-values at $(0\%, 1.01\%, \ldots, 98.9\%, 100\%)$ of the cumulative weight.

For data with many repeated x-values, some of our candidates will be duplicates. For example, 55% of the values in the `length_of_stay` feature in the COMPAS data set are 0 or 1. In such cases, we iteratively resample at higher rates (such as $0\%, 0.505\%, 1.01\%$, etc.) until we collect a suitable number of distinct candidates, never exceeding the specified `num_samples` parameter.

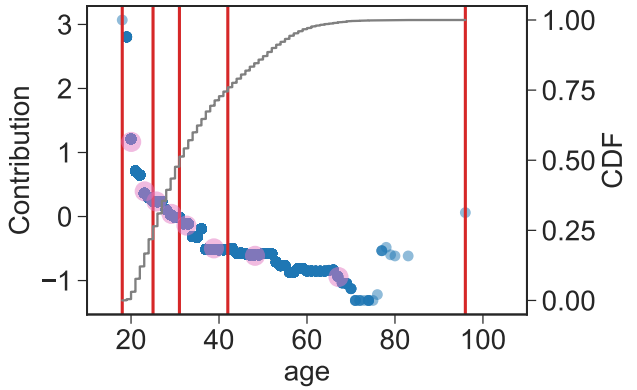

Figure 5.1: Candidate x-knots (red vertical lines) and derived condensed points (pink large dots) on the `age` piece of a COMPAS GAM forest model (blue dots). For visual clarity, this illustration considers only five x-knot candidates.

5.1.3 Condensation. To minimize the cost of each linear least squares step, we condense the data using a novel technique described in Appendix B. Given `num_samples` candidate knots, we condense the full data into two synthetic points per adjacent pair of candidates, for a total of `2 * (num_samples - 1)` synthetic points. For any function that's linear between each adjacent pair of candidate x-knots, which is guaranteed by our choice of discrete candidate x-knots, these condensed points perfectly recreate the loss of that function over the full data set. We run our linear least squares solver on the condensed points instead of the full data set, which reduces our cost per solve from $O(\texttt{num\_points})$ to $O(\texttt{num\_samples})$. This is purely a performance optimization, with no quality implications.

5.1.4 Global Optimization via Greedy Search. After discretization, the solution space consists of $\binom{\texttt{num\_samples}}{\texttt{num\_segments}+1}$ x-knot combinations, which is still too large for an exhaustive search. To make the search tractable we use a greedy search heuristic that optimizes one x-knot at a time. Specifically, at each step of the process we evaluate the error associated with each candidate x-knot, and keep the candidate that yields the least error.

With this approach, we optimize in two stages. We begin with a single x-knot as our solution, and greedily add the best remaining candidate x-knot until our solution consists of (`num_segments` + 1) x-knots. Then we cycle through our solution, removing one x-knot at a time and replacing that x-knot with the best remaining candidate x-knot, which could be the same x-knot that we just removed. We continue this cycle of iterative improvements until our solution converges, or until we've exceeded the maximum number of runs (defaulting to 10 runs).

5.1.5 Slope Constraints & Monotonicity. `pwlfit` can impose a minimum and/or maximum slope on the solution via bounded least squares. Instead of solving the least squares expression directly for the y-knots, we solve it for the deltas between adjacent y-knots. Then we impose a min/max slope by bounding the deltas. Slope restrictions can be used to limit the spikiness of curves, but we primarily use them to impose monotonicity. For example, specifying `min_slope=0` restricts to monotonically non-decreasing functions while `max_slope=0` restricts to monotonically non-increasing functions. Specifying a `min_slope` greater than 0 or a `max_slope` less than 0 restricts to strictly increasing or decreasing functions, respectively.

`pwlfit` can deduce the direction of monotonicity by applying isotonic regression [25] to the condensed points. We fit an increasing and a decreasing isotonic regression, and use the direction that minimizes mean squared error. The user can override this behavior by specifying the direction explicitly or by disabling monotonicity entirely.

5.1.6 Input Transformations. `pwlfit` can also interpolate in a transformed x-coordinate space instead of the original space, as a simple form of feature engineering. `pwlfit` transforms the x-values before learning the curve. Specifically, `pwlfit` will choose a candidate x-transformation among `log`, `log1p`, or `symlog1p` based on the range of the x-values and then proceed with that transformation if it increases the Pearson correlation between $f(x)$ and y by a noticeable amount over the identity transformation. Alternatively, the user can specify any strictly increasing 1D transform or specify the identity transform to disable transformation. Representative `PWLCurve` examples with each of these transforms are shown in Figure 2.1.

6 EXPERIMENTS

6.1 Distillation Accuracy

Table 6.1 and Figure 6.1 show the results obtained from experiments on the different datasets. The results of applying our distillation technique with our piecewise-linear approximation algorithm are presented as `pwlfit`. We present results from using various numbers of segments without a monotonicity restriction and otherwise default parameters. In all cases we truncated the control points to four significant digits. We also present several additional reference points to provide context.

- **SGD**: We directly learn the curves with the Adadelta[30] optimizer. We initialize the y values of the control points as zeros. For the x values of the control points we use the quantiles for numerical features (e.g. 0%, 50%, 100% for a three point, two segment curve) or all unique values for categorical features. We then apply Adadelta to optimize the y values. Simultaneously optimizing x and y values was also attempted, but the results were always worse than optimizing y values alone.
- **NAM**: Neural Additive Models (NAMs) [2] is another method for learning interpretable models proposed by Agarwal et al. We present their result for reference where applicable.
- **Interacting forest**: We train a bagged, boosted decision forest allowing feature interactions to demonstrate the accuracy of a non-interpretable, high-complexity "black box" model.
- **GAM forest**: We train a bagged boosted decision forest GAM by restricting each tree to use only one feature. This model is also the source model for our distillation technique.
- `pwlf`: We apply our distillation technique using an alternative piecewise-linear approximation algorithm `pwlf`[13].

We used three different metrics to evaluate accuracy. Further details on our experimentation setup can be found in the Appendix.

- **COMPAS**: Labels are whether recidivism occurred for an individual within a time period. We use area under the receiver operating characteristic curve (AUC-ROC) to measure classifier accuracy. Five fold cross validation was performed with the same folds as used by the NAM paper [2]. We present the mean across folds with a 95% confidence interval.
- **FICO**: Labels are a risk score for an individual. We use root mean square error (RMSE) to measure regressor accuracy. We again use five fold cross validation with the same folds as the NAM paper [2] and present the mean across folds with a 95% confidence interval.
- **MSLR-WEB30K**: Labels are per document relevance judgements. We use exponential normalized discounted cumulative gain at $k = 5$ (NDCG@5) to measure ranker accuracy. Cross validation was not performed given the large size of the data - only the first of the predefined folds was used. Here we present the metric mean and 95% confidence interval on the predefined test set.

Our results show that applying our distillation technique with 4-5 segments with `pwlfit` always produces models which are as accurate as both the source GAM forest and NAM models. In the case of the COMPAS dataset these models are as accurate as full complexity models. Applying our technique with `pwlf` produces competitive results, albeit less accurate on the MSLR-WEB30k dataset. By contrast, the results show that learning curves directly via SGD is less general. On the FICO dataset more segments are required to achieve accuracy comparable to the NAM and GAM forest models. On the MSLR-WEB30k dataset the accuracy is inferior even with many more segments.

The consistent accuracy of applying our distillation approach with `pwlfit` on these three datasets and three separate tasks (classification, regression, learning to rank) demonstrates that the process is not sensitive to either the specific data or the top level objective being used.

Table 6.1: Metrics across datasets with 95% confidence intervals. For AUC-ROC and NDCG@5 higher is better, for RMSE lower is better. MSLR-WEB30k was not used by the NAM paper and is omitted from that row.

Model	COMPAS (AUC-ROC)	FICO (RMSE)	MSLR-WEB30K (NDCG@5)
Interacting forest	0.742 ± 0.014	3.128 ± 0.114	0.485 ± 0.006
GAM forest	0.741 ± 0.016	3.495 ± 0.136	0.442 ± 0.006
NAM	0.741 ± 0.025	3.490 ± 0.225	
`pwlfit num_segments=1`	0.740 ± 0.010	3.782 ± 0.129	0.431 ± 0.006
`pwlfit num_segments=2`	0.741 ± 0.014	3.617 ± 0.124	0.437 ± 0.006
`pwlfit num_segments=3`	0.743 ± 0.013	3.535 ± 0.124	0.438 ± 0.006
`pwlfit num_segments=4`	0.743 ± 0.014	3.504 ± 0.117	0.440 ± 0.006
`pwlfit num_segments=5`	0.743 ± 0.015	3.493 ± 0.121	0.440 ± 0.006
`pwlf num_segments=2`	0.741 ± 0.019	3.724 ± 0.110	0.426 ± 0.006
`pwlf num_segments=3`	0.744 ± 0.014	3.568 ± 0.127	0.429 ± 0.006
`pwlf num_segments=4`	0.744 ± 0.015	3.500 ± 0.111	0.430 ± 0.006
`pwlf num_segments=5`	0.743 ± 0.015	3.525 ± 0.108	0.432 ± 0.006
SGD num_segments=1	0.728 ± 0.010	4.349 ± 0.074	0.352 ± 0.006
SGD num_segments=2	0.734 ± 0.008	4.028 ± 0.118	0.382 ± 0.006
SGD num_segments=3	0.742 ± 0.010	3.887 ± 0.100	0.393 ± 0.006
SGD num_segments=4	0.742 ± 0.012	3.742 ± 0.137	0.401 ± 0.006
SGD num_segments=5	0.741 ± 0.013	3.643 ± 0.121	0.404 ± 0.006
SGD num_segments=15	0.740 ± 0.014	3.499 ± 0.126	0.419 ± 0.006
SGD num_segments=20	0.738 ± 0.014	3.499 ± 0.145	0.418 ± 0.006

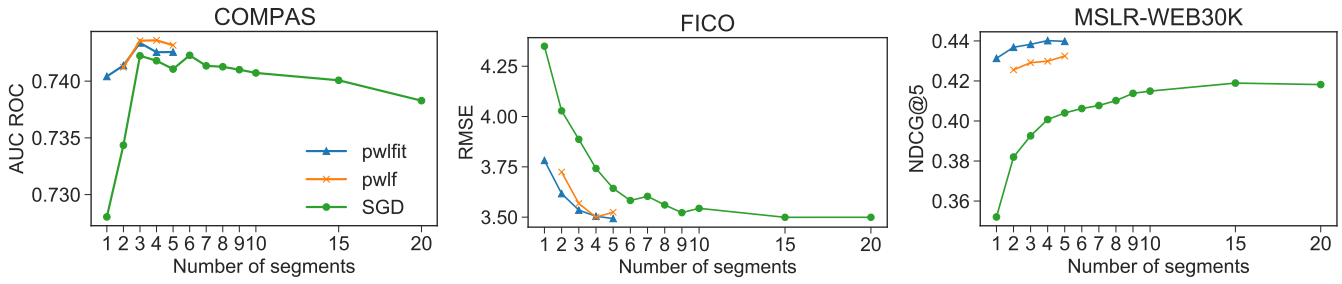

Figure 6.1: Metrics across datasets. For AUC-ROC and NDCG@5 higher is better, for RMSE lower is better.

6.2 Efficiency & Robustness

The experiments of the previous section showed that `pwlfit` more accurately distills the source model across datsets than `pwlf`. We also found on the MSLR-WEB30k dataset that `pwlfit` is more efficient and robust than `pwlf`. Figure 6.2 shows per fit metrics on the MSLR-WEB30K dataset as the number of segments varies. The top plot shows the time in seconds, as measured on a ThinkStation P520, to fit each of the 136 submodels of the source GAM forest. We find that `pwlfit` is faster in the average case as the number of segments increases, and has a narrower distribution. The bottom plot shows the RMSE of each fit against its source for the 136 submodels of the GAM forest. We again find that `pwlfit` performs favorably in the average case with a narrower distribution.

It's worth noting that `pwlf` by default does not perform any downsampling. For the MSLR-WEB30k dataset running `pwlf` without any downsampling was prohibitively expensive. For all of our experiments we ran `pwlf` with a pre-processing downsample to a random 1000 examples. We found this to be a fair point for balancing speed and quality when comparing to `pwlfit`. It is of course possible with both algorithms to modify the number of samples used to strike a different tradeoff between run time and accuracy.

6.3 Monotonicity

As discussed in Section 5, `pwlfit` can fit monotonic curves with automatic direction detection. Figure 6.3 compares curve models fit with and without monotonicity constraints (automatically inferring the direction) across datasets. For the COMPAS dataset monotonic and non monotonic models are comparably accurate, while for FICO and MSLR-WEB30K non-monotonic models are more accurate.

Monotonicity with respect to appropriate features is a desirable, if not required, property for interpretable models. In these cases a monotonic model may be preferable to a non-monotonic one, even if it is less accurate. For example, Figure 6.4 compares monotonic and non-monotonic 5 segment curve models on the FICO dataset for the `MSinceMostRecentDelq`, `PercentTradesWBalance`, and `NumSatisfactoryTrades` features. In each case, given the semantic meaning of the feature, it

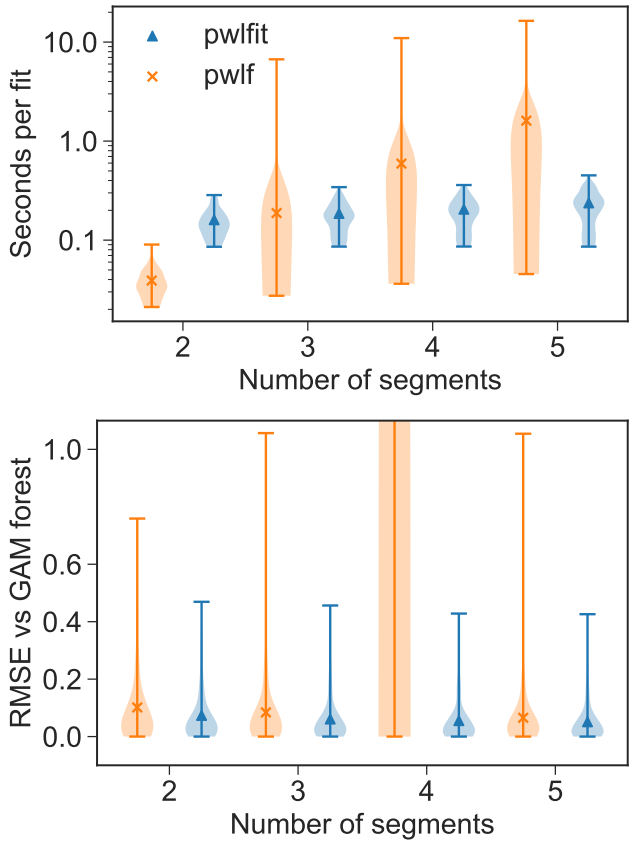

Figure 6.2: Time in seconds to fit, and RMSE vs source submodel for the MSLR-WEB30K dataset. Note that for 4 segments `pwlf` has extreme outliers for RMSE vs the source submodel.

is desirable for the models output to be monotonic with respect to it. This is especially true for certain considerations such as transparency. For a transparent risk score model, users would be shocked to find that going from 1 month without delinquency to 12 months (`MSinceMostRecentDelq`) harms their risk score.

7 CONCLUSION

We have introduced a novel method for distilling interpretable models into human-readable code using piecewise-linear curves and demonstrated its efficacy on three datasets. We have shown that curve models match or outperform the accuracy achieved by other additive models. On smaller datasets, curve models match the accuracy of more complex models, like interacting decision forests. Our localized distillation methodology is applicable to any model containing univariate numerical functions and is straightforward to implement using the publicly-available `pwlfit`[21] library.

We have explained how curve model distillation reinforces interpretability and addresses a variety of real-world engineering challenges. Curve models are 1) transparent, 2) well-regularized, 3) easy to analyze for presence of biases or other fairness issues, and 4) can be directly edited or improved outside the context of machine learning to fix the aforementioned fairness issues. Distilling models into human-readable code allows one to address novel machine learning problems using well-established software engineering methods. Curve models can be improved by multiple contributors in parallel, reviewed, and made to systematically follow best practices. Curve models are well-suited for production applications, since they can be natively supported in many languages, are easy to deploy, and fast to evaluate.

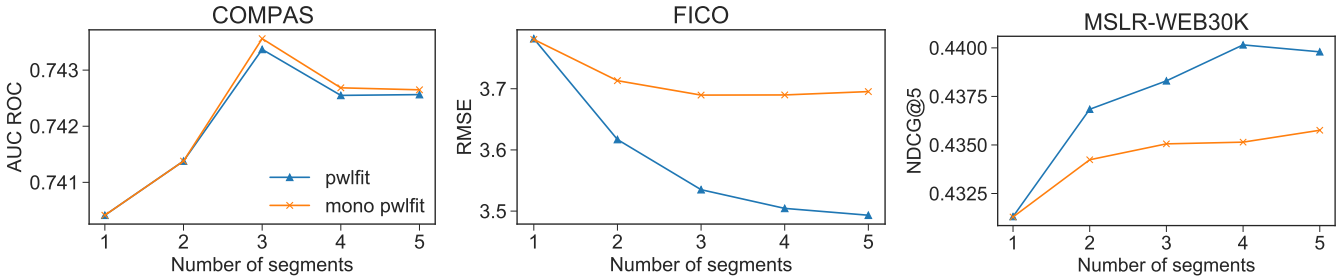

Figure 6.3: Metrics across datasets using monotonic curves. For AUC-ROC and NDCG@5 higher is better, for RMSE lower is better.

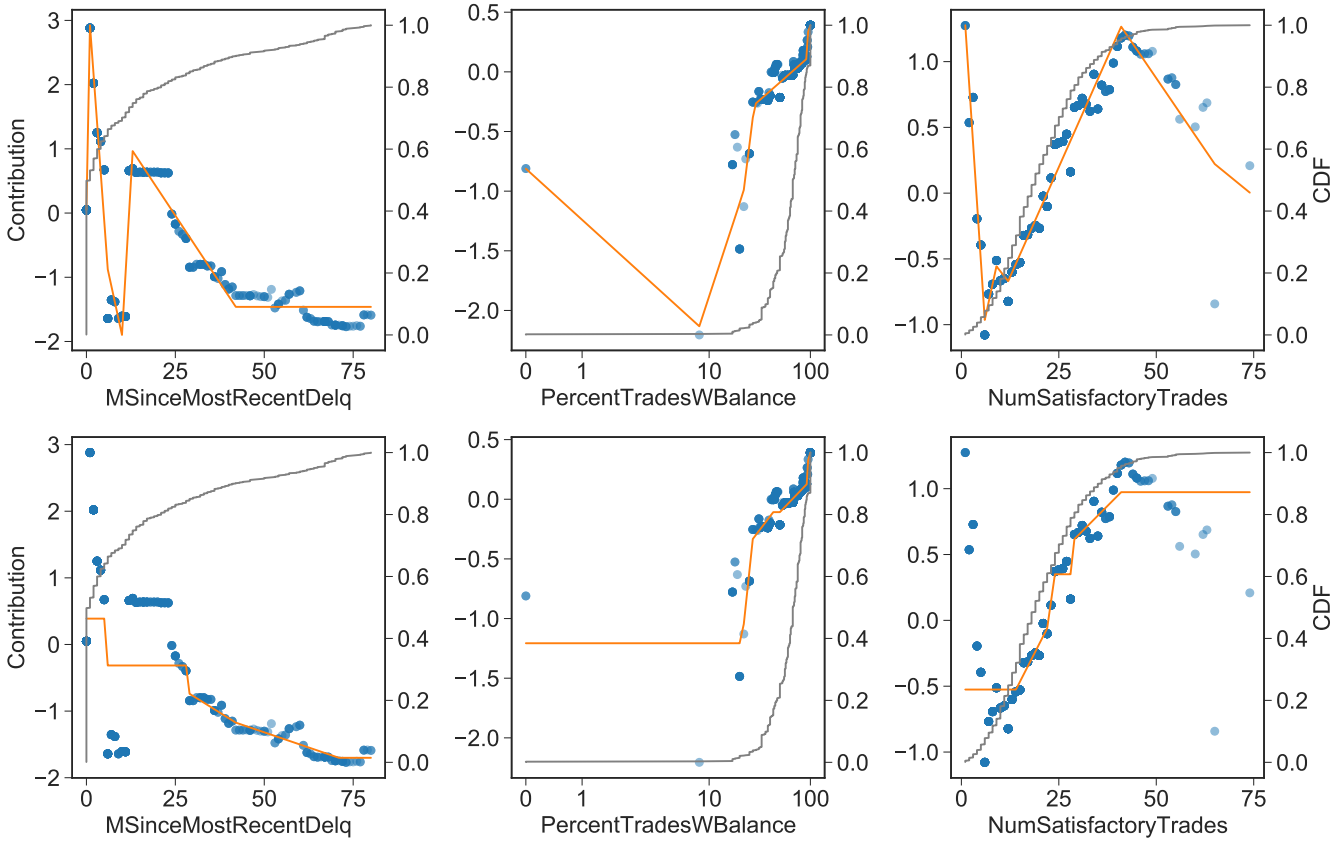

Figure 6.4: Models fit on the FICO dataset using 5 segments without monotonicity (top) and with monotonicity (bottom).

ACKNOWLEDGMENTS

We thank Vytenis Sakenas, Jaime Fernandez del Rio, Benoit Zhong, and Petr Mitrichev for their support and providing the algorithms and optimization infrastructure used in our experiments. We also thank Paul Heymann, Diego Federici, Mike Bendersky, Paul Haahr, and Petr Mitrichev for their helpful feedback and detailed reviews.

REFERENCES

[1] 2018. FICO Explainable Machine Learning Challenge. https://community.fico.com/s/explainable-machine-learning-challenge.

[2] Rishabh Agarwal, Nicholas Frosst, Xuezhou Zhang, Rich Caruana, and Geoffrey E. Hinton. 2020. Neural Additive Models: Interpretable Machine Learning with Neural Nets. arXiv:cs.LG/2004.13912

[3] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. 2017. Learning Certifiably Optimal Rule Lists for Categorical Data. arXiv:stat.ML/1704.01701

[4] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine bias: There's software used across the country to predict future criminals. *And it's biased against blacks. ProPublica* 23 (2016).

[5] Cristian Buciluundefined, Rich Caruana, and Alexandru Niculescu-Mizil. 2006. 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, 535–541. https://doi.org/10.1145/1150402.1150464

[6] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. 2015. Intelligible Models for HealthCare: Predicting Pneumonia Risk and Hospital 30-Day Readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. Association for Computing Machinery, New York, NY, USA, 1721–1730. https://doi.org/10.1145/2783258.2788613

[7] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. 2018. An Interpretable Model with Globally Consistent Explanations for Credit Risk. arXiv:cs.LG/1811.12615

[8] Alexandra Chouldechova. 2017. Fair Prediction with Disparate Impact: A Study of Bias in Recidivism Prediction Instruments. *Big Data* 5, 2 (Jun 2017), 153–163. https://doi.org/10.1089/big.2016.0047

[9] Julia Dressel and Hany Farid. 2018. The accuracy, fairness, and limits of predicting recidivism. *Science Advances* 4, 1 (2018). https://doi.org/10.1126/sciadv.aao5580 arXiv:https://advances.sciencemag.org/content/4/1/eaao5580.full.pdf

[10] Mengnan Du, Ninghao Liu, and Xia Hu. 2019. Techniques for interpretable machine learning. *Commun. ACM* 63, 1 (Dec 2019), 68–77. https://doi.org/10.1145/3359786

[11] Trevor Hastie and Robert Tibshirani. 1986. Generalized Additive Models. *Statist. Sci.* 1, 3 (08 1986), 297–310. https://doi.org/10.1214/ss/1177013604

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the Knowledge in a Neural Network. arXiv:stat.ML/1503.02531

[13] Charles F. Jekel and Gerhard Venter. 2019. *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*. https://github.com/cjekel/piecewise_linear_fit_py

[14] Jon Kleinberg, Sendhil Mullainathan, and Manish Raghavan. 2016. Inherent Trade-Offs in the Fair Determination of Risk Scores. arXiv:cs.LG/1609.05807

[15] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, F. Loizides and B. Schmidt (Eds.). IOS Press, 87 – 90.

[16] A. K. Kolmogorov. 1957. On the Representation of Continuous Functions of Several Variables by Superposition of Continuous Functions of One Variable and Addition. *Doklady Akademii Nauk SSSR* 114 (1957), 369–373.

[17] Yin Lou, Rich Caruana, and Johannes Gehrke. 2012. Intelligible Models for Classification and Regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. Association for Computing Machinery, New York, NY, USA, 150–158. https://doi.org/10.1145/2339530.2339556

[18] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. 2013. Accurate Intelligible Models with Pairwise Interactions. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '13)*. Association for Computing Machinery, New York, NY, USA, 623–631. https://doi.org/10.1145/2487575.2487579

[19] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[20] Cynthia Rudin. 2018. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. arXiv:stat.ML/1811.10154

[21] Ethan Sterling and Walker Ravina. 2019. *pwlfit: A Piecewise-Linear Curve Fitting Library*. https://github.com/google/pwlfit

[22] Sarah Tan, Rich Caruana, Giles Hooker, and Yin Lou. 2018. Distill-and-Compare. *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society* (Feb 2018). https://doi.org/10.1145/3278721.3278725

[23] Berk Ustun and Cynthia Rudin. 2014. Methods and Models for Interpretable Linear Classification. arXiv:stat.ME/1405.4047

[24] Wikipedia. 2020. Bhatia–Davis inequality. http://en.wikipedia.org/w/index.php?title=Bhatia%E2%80%93Davis%20inequality&oldid=875899600. [Online; accessed 03-September-2020].

[25] Wikipedia. 2020. Isotonic regression. http://en.wikipedia.org/w/index.php?title=Isotonic%20regression&oldid=989717822. [Online; accessed 30-November-2020].

[26] Wikipedia. 2020. Kolmogorov–Arnold representation theorem. http://en.wikipedia.org/w/index.php?title=Kolmogorov%E2%80%93Arnold%20representation%20theorem&oldid=964097101. [Online; accessed 10-August-2020].
[27] Wikipedia. 2020. Minimum description length. http://en.wikipedia.org/w/index.php?title=Minimum%20description%20length&oldid=965620302. [Online; accessed 12-August-2020].
[28] Wikipedia. 2020. Nothing-up-my-sleeve number. http://en.wikipedia.org/w/index.php?title=Nothing-up-my-sleeve%20number&oldid=972510276. [Online; accessed 12-August-2020].
[29] Wikipedia. 2020. Segmented regression. http://en.wikipedia.org/w/index.php?title=Segmented%20regression&oldid=910888930. [Online; accessed 10-August-2020].
[30] Matthew D. Zeiler. 2012. ADADELTA: An Adaptive Learning Rate Method. *CoRR* abs/1212.5701 (2012). arXiv:1212.5701 http://arxiv.org/abs/1212.5701
[31] Honglei Zhuang, Xuanhui Wang, Michael Bendersky, Alexander Grushetsky, Yonghui Wu, Petr Mitrichev, Ethan Sterling, Nathan Bell, Walker Ravina, and Hai Qian. 2020. Interpretable Learning-to-Rank with Generalized Additive Models. arXiv:cs.IR/2005.02553

A EXPERIMENTAL DETAILS

A.1 Cross Validation

K fold cross validation was performed on the COMPAS, and FICO datasets with 5 folds. Each fold was used once as a test set (20%) with the remaining folds as the training set (80%). We used the same random folds as in the NAM paper [2]. No validation set was used in the case of cross validation and instead out of bag evaluation was used (see below).

A.2 Ensemble Learning

For both SGD and tree training we trained ensembles with 56 bags using a bag frac of $\frac{7}{8}$ to produce the random subsets. For MSLR-WEB30K queries were randomly divided into bags. For the other datasets individual examples were randomly divided into bags. When applying our distillation technique we distilled the ensembles into a single curve per feature. When learning the curves directly via SGD we averaged the learned y-coordinate values across bags to obtain the final model.

A.3 Loss Functions

We trained SGD and tree models using log-loss for the COMPAS dataset, mean squared error (MSE) for the FICO dataset, and ranking loss (approximate NDCG@5) for the MSLR-WEB30K dataset.

A.4 Hyper-parameters

For the COMPAS, and FICO datasets hyper-parameters were tuned using out of bag evaluation on the training set of the first fold. For MSLR-WEB30k we used the predefined validation set of the first fold.

- **SGD** We tuned the batch size in {128, 256, 512, 1024, 4096}. We used the Adadelta [30] optimizer and tuned a sufficient maximum number of steps for convergence. No other parameters were tuned.
- **Interacting forest** We trained depth 5 trees using an internal boosted forest algorithm. We tuned a sufficient maximum number of steps for convergence. No other parameters were tuned.
- **GAM forest** We trained depth 3 trees restricted to using a single feature with an internal boosted forest algorithm. We tuned a sufficient maximum number of steps for convergence. No other parameters were tuned.

In all cases we trained models for the tuned maximum number of steps and then truncated models after training. Truncation used out of bag evaluation and a confidence based truncation algorithm which attempts to select the earliest step for which no later step provides a confident win.

A.5 Code

The GitHub repository for `pwlfit` [21] contains several Jupyter notebooks applying our distillation technique and performing the analyses shown in this paper. Please reference the v0.1.0 release to get the accompanying data files and appropriate snapshot of the source code and notebooks.

B LINEAR CONDENSE

Linear condensing is a data optimization designed to reduce the runtime complexity of our piecewise-linear curve fitting.

B.1 Motivation/Overview

`pwlfit` picks a set of candidate x-knots and searches through combinations of those x-knots. For each combination considered, it solves a linear least squares expression for the ideal y-knots, calculates the resulting squared error, and prefers the combination that yields the lowest error.

Each solve is linear in the size of input, which is slow for large data. We could downsample to save compute at the cost of accuracy. Instead, we introduce a technique to save compute at no cost in accuracy: we distill the data into $O(\#candidates)$ synthetic points that perfectly capture the squared error of every candidate `PWLCurve` over the full data. Then we optimize over the synthetic points instead of the real points.

This is possible because we know the candidate x-knots ahead of time. A `PWLCurve` defined on those x-knots will always be linear between any adjacent x-knots in the set of candidates. As we show in the theorem, we can distill arbitrarily many points down to two points such that linear fits are the same on those two points as on the full set. In the corollary, we apply this process separately between each pair of candidate x-knots, producing two points between each pair. Together, the squared error of such a `PWLCurve` is the same on those synthetic points as it is on the full data set. (Up to a constant that we safely ignore because it's the same for each `PWLCurve`.)

B.2 Definitions

For convenience, we take standard definitions and specialize them for weighted 2D points.

Definition B.1. Let a 'point' refer to a real-valued triple of the form $(x, y, weight)$ with $weight > 0$.

Definition B.2. Let a 'line' refer to a function of the form $f(x) = mx + b$ for $m, b, x \in \mathbb{R}$.

Definition B.3. For any function $f : \mathbb{R} \to \mathbb{R}$ and finite point set P, define the squared error $SE(f, P)$ as the sum of $(f(x) - y)^2 \cdot weight$ for each point in P. If P is empty, we consider the squared error to be 0.

Definition B.4. For any finite point set P, define the 'best fit line' $bestfitline(P)$ as the line L that minimizes $SE(L, P)$. In the degenerate case where multiple lines minimize SE, let the best fit line be the solution with zero slope, and if multiple solutions have zero slope, let the best fit line be the solution with a zero y-intercept.

There are two degenerate cases that require tie-breaking. If the point set is empty, every line has the same squared error, so our definition chooses $f(x) = 0$ as the best fit line. If the point set is nonempty but all its points have the same x, then any line with the correct value at x will minimize the squared error, so our definition choose the horizontal line.

B.3 Theorem

Theorem B.5. *Given a set of points P, we can construct a set P' of two or fewer points such that*

1. $min_x(P) <= min_x(P') <= max_x(P') <= max_x(P)$, *and*
2. *For any line L, $SE(L, P) = SE(L, P') + SE(bestfitline(P), P)$.*

Remark. These properties are desirable because (2) allows us to compute the squared error of M lines over a data set of N points in $O(N + M)$ instead of the naive $O(NM)$, and (1) allows us to extend this property from lines to a useful class of piecewise-linear curves in the corollary.

Note that the points in P' are constructed, rather than chosen from P. The construction of P' is implemented in `pwlfit` [21] as `linear_condense.linear_condense`.

Proof. Let X, Y, and W represent the x, y, and $weight$ values of P, respectively. We dismiss the trivial case where P is empty; in that case, an empty P' satisfies the requirements. Likewise, we dismiss the case where $min(X) = max(X)$ since $P' = \{Centroid(P)\}$ fulfills our desired properties. With those cases resolved, we assume for the rest of this proof that $min(X) < max(X)$.

B.3.1 Reframe the Coordinate System. To begin, we reframe the coordinate system such that the origin is the centroid of P and $y = 0$ is the best fit line. (This simplifies the math.) We ensure that the shift of coordinates is reversible and preserves the squared error.

$Centroid(P) = (X \cdot W/sum(W), Y \cdot W/sum(W))$. We translate the coordinate frame by this centroid so that, under the new coordinates, $Centroid(P) = (0, 0)$. After translation, $X \cdot W = 0$ and $Y \cdot W = 0$.

Additionally, we skew the coordinate system by the slope of the best fit line: we replace Y with $Y - X \cdot slope(bestfitline(P))$. With the centroid at the origin, the slope of the best fit line is $Covariance(X, Y, W)/Variance(X, W) = (XY \cdot W)/(XX \cdot W)$. After skewing this slope to 0, $XY \cdot W = 0$.

Under the new coordinate frame, $SE(bestfitline(P), P) = SE(y = 0, P) = Y^2 \cdot W$.

We will determine P' under this new coordinate system. Afterwards, we can easily convert P' back to the original coordinate system by reversing the skew and the translation.

B.3.2 Squared Error of an arbitrary line. We will express $SE(line, P)$ as $SE(bestfitline(P), P)$ plus leftover terms. From that, we will derive a P' such that $SE(line, P')$ equals those leftover terms.

For an arbitrary line $y = mx + b$,

$SE(y = mx + b, P) = (mX + b - Y)^2 \cdot W = (m^2X^2 + 2bmX - 2mXY + b^2 - 2bY + Y^2) \cdot W$.

In our coordinate frame, $X \cdot W = 0$, $Y \cdot W = 0$, and $XY \cdot W = 0$. So $SE(y = mx + b, P) = (m^2X^2 + b^2 + Y^2) \cdot W$.

$Y^2 \cdot W = SE(bestfitline(P), P)$. Therefore,

$$SE(y = mx + b, P) = m^2X^2 \cdot W + b^2 \cdot W + SE(bestfitline(P), P).$$

$$m^2X^2 \cdot W + b^2 \cdot W = SE(y = mx + b, P) - SE(bestfitline(P), P).$$

B.3.3 Squared error over P'.

$$SE(y = mx + b, P') = SE(y = mx + b, P) - SE(bestfitline(P), P)$$

for all lines $y = mx+b \iff (mX'+b-Y')^2 \cdot W' = m^2X^2 \cdot W + b^2 \cdot W$ for all lines $y = mx + b$.

The above equation can be viewed as a quadratic polynomial in the two variables m and b. To hold for all values of m and b, the coefficients of each $m^c b^d$ must be equal on both sides of the equation. Then the equation holds iff:

1. $X'^2 \cdot W' = X^2 \cdot W$, and
2. $X' \cdot W' = 0$, and
3. $sum(W) = sum(W')$, and
4. $Y' \cdot W' = 0$, and
5. $Y'^2 \cdot W' = 0$, and
6. $X'Y' \cdot W' = 0$.

(5) $\iff$ $Y' = 0$, which also guarantees (4) and (6). We will use 1-3 to derive a satisfactory X' and W'.

B.3.4 Deriving X' and W'. We've determined that $Y' = 0$.

Let $X' := (x_1, x_2)$ and $W' := (w_1, w_2)$. Without loss of generality, let $x_1 <= x_2$. Then, to satisfy our squared error expression, it's necessary and sufficient that:

1. $x_1^2 w_1 + x_2^2 w_2 = X^2 \cdot W$, and
2. $x_1 w_1 + x_2 w_2 = 0$, and
3. $w_1 + w_2 = sum(W)$.

Because we have three equations in four unknowns, we cannot directly solve for x_1, x_2, w_1, w_2. To produce a fourth equation, we choose the constraint that $x_1/x_2 = min(X)/max(X)$. This choice will simplify the math, and will ensure that $min(X) <= x_1 <= x_2 <= max(X)$.

With this fourth equation, we solve the simultaneous equations to produce:

$x_1 = -stddev(X, W)\sqrt{-min(X)/max(X)}$
$x_2 = stddev(X, W)\sqrt{max(X)/-min(X)}$.
$w_1 = sum(W) \cdot max(X)/(max(X) - min(X))$
$w_2 = sum(W) \cdot -min(X)/(max(X) - min(X))$.

Note that, because the centroid is zero, $min(X) < 0 < max(X)$, so these expressions are all defined. (The denominators are never 0 and values beneath the square roots are never negative.)

$P' = (x_1, 0, w_1), (x_2, 0, w_2)$ satisfies our requirements.

B.3.5 Verify that $min(X) <= x_1 <= x_2 <= max(X)$. We wanted P' to satisfy the the squared error expression, which it does, and also have its x-values bounded by the x-values of P, which we prove now. Let $\mu := E(X, W)$, the expected value of X weighted by W, which is equivalent to the x-value of $Centroid(P)$. By the Bhatia–Davis inequality [24],

$stddev(X, W)^2 <= (\mu - min(X))(max(X) - \mu)$. (This inequality is equivalent to the observation that the standard deviation of a

distribution is maximized when all the xs are at the extremes – i.e. equal to min(X) or max(X).)

Since μ is zero for P, $stddev(X,W)^2 <= -min(X)max(X)$.

$x_1^2 = stddev(X,W)^2 \cdot (-min(X)/max(X)) <= -min(X)max(X) \cdot (-min(x)/max(X)) = min(X)^2$.

$x_1 < 0$ and $min(X) < 0$, so $x_1^2 <= min(X)^2 \implies min(X) <= x_1$. The proof that $x_2 <= max(X)$ is similar.

Therefore $min(X) <= x_1 <= x_2 <= max(X)$, as desired. □

B.4 Corollary

Corollary B.6. *Given a set of points P and a set of x-knots K, we can construct a set of points P' with $|P'| <= 2(|K|-1)$ such that, for any* `PWLCurve` *C whose x-knots are elements of K, $SE(C,P) = SE(C,P') + c$, where c is a constant determined exclusively by P and K that's the same for every C.*

Note that the points in P' are constructed, rather than chosen from P. The construction of P' is implemented in `pwlfit` [21] as `linear_condense.condense_around_knots`.

B.4.1 Preprocess P by clamping. Let $k := |K|$, and consider K in sorted order. Piecewise-linear curves are constant for input values that exceed the range of their x-knots, so C is constant for $x <= min(K) = K[0]$ and for $x >= max(K) = K[k-1]$.

Therefore we can clamp the x-values of P to $[K[0], K[k-1]]$ without altering $SE(C,P)$. We do so as a preprocess.

B.4.2 Partition P by K. To construct P' from P, we first partition P by K into $k+1$ disjoint pieces labeled $P_0, P_1, ..., P_k$.

- $P_0 := \{x \in P | x < K[0]\}$.
- $P_i := \{x \in P | K[i-1] <= x < K[i]\}$ for $1 <= i <= k-2$.
- $P_{k-1} := \{x \in P | K[k-2] <= x <= K[k-1]\}$.
- $P_k := \{x \in P | K[k-1] < x\}$.

Because we clamped P, P_0 and P_k are empty. Therefore $\bigcup_{i=1}^{k-1} P_i = P$.

A `PWLCurve` is linear between consecutive control points, so C is linear over each P_i.

B.4.3 Convert each partition into two points. From the theorem, for each P_i, we can produce a two-point set P_i' with $min_x(P_i) <= min_x(P_i') <= max_x(P_i') <= max_x(P_i)$, such that for any line L,

$SE(L,P_i) = SE(L,P_i') + SE(bestfitline(P_i), P_i)$. C is linear over each P_i, so

$SE(C,P_i) = SE(C,P_i') + SE(bestfitline(P_i), P_i)$.

B.4.4 Recombine partitions. Let $P' := \bigcup_{i=1}^{k-1} P_i'$. Each P_i' consists of two points, so P' consists of $2(|K|-1)$ points.

$$\begin{aligned} SE(C,P) &= \sum_{i=1}^{k-1} SE(C,P_i) \\ &= \sum_{i=1}^{k-1} (SE(C,P_i') + SE(bestfitline(P_i), P_i)) \\ &= SE(C,P') + \sum_{i=1}^{k-1} SE(bestfitline(P_i), P_i). \end{aligned}$$

$\sum_{i=1}^{k-1} SE(bestfitline(P_i), P_i)$ is determined by P and K, and is therefore the same for every C. Therefore we've proven the corollary.