
Technical Report: Partial Dependence without Model Predictions through Stratification

Terence Parr
University of San Francisco
parrt@cs.usfca.edu

James D. Wilson
University of San Francisco
jdwilson4@usfca.edu

Abstract

1 Introduction

Partial dependence, the isolated effect of a specific variable or variables on the response variable, y , is important to researchers and practitioners in many disparate fields such as medicine, business, and the social sciences. For example, in medicine, researchers are interested in the relationship between an individual’s demographics or clinical features and their susceptibility to illness. Business analysts at a car manufacturer might need to know how changes in their supply chain are affecting defect rates. Climate scientists are interested in how different atmospheric carbon levels affect temperature.

For an explanatory matrix, \mathbf{X} , with a single variable, x_1 , a plot of the y against x_1 visualizes the marginal effect of feature x_1 on y exactly. Given two or more features, one can similarly plot the marginal effects of each feature separately, however, the analysis is complicated by the interactions of the variables. Variable interactions, codependencies between features, result in marginal plots that do not isolate the specific contribution of a feature of interest to the target. For example, a marginal plot of sex (male/female) against body weight would likely show that, on average, men are heavier than women. While true, men are also taller than women on average, which likely accounts for most of the difference in average weight. It is unlikely that two “identical” people, differing only in sex, would be appreciably different in weight.

Rather than looking directly at the data, there are several partial dependence techniques that interrogate fitted models provided by the user: Friedman’s original partial dependence (which we will denote FPD) Friedman [2000], Individual Conditional Expectations (ICE) Goldstein et al. [2015], Accumulated Local Effects (ALE) Apley [2016], and most recently SHAP Lundberg and Lee [2017]. Model-based techniques dominate the partial dependence research literature because interpreting the output of a fitted model has several advantages. Models have a tendency to smooth over noise. Models act like analysis preprocessing steps, potentially reducing the computational burden on model-based partial dependence techniques; e.g., ALE is $O(n)$ for the n records of \mathbf{X} . Model-based techniques are typically model-agnostic, though for efficiency, some provide model-specific optimizations, as SHAP does. Partial dependence techniques that interrogate models also provide insight into the models themselves; i.e., how variables affect model behavior. It is also true that, in some cases, a predictive model is the primary goal so creating a suitable model is not an extra burden.

Model-based techniques do have some significant disadvantages, however. As we demonstrate in Section 4 using synthetic and real data sets, model-based techniques vary in their ability

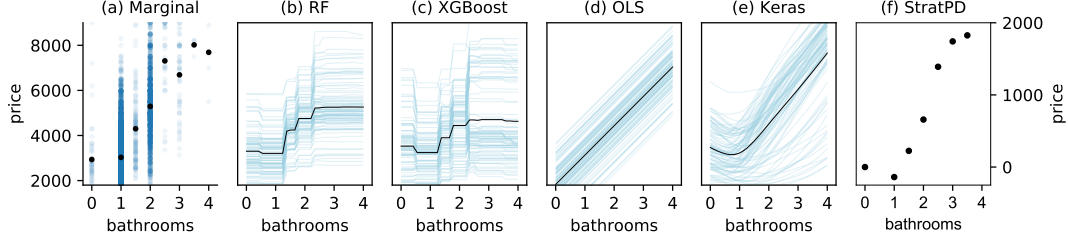


Figure 1: Plots of bathrooms versus rent price using New York City apartment rent data. (a) marginal plot, (b) PD/ICE plot derived from random forest, (c) PD/ICE plot derived from gradient boosted machine, and (d) PD/ICE plot derived from ordinary least squares regression; sample size is 10,000 observations of ~50k. The PD/ICE plots are different for the same data set, depending on the chosen user model. `X[‘bathrooms’].unique()` shows `(array([0. , 1. , 1.5, 2. , 2.5, 3. , 3.5, 4.]), array([54, 8151, 140, 1539, 39, 67, 3, 7]))`. STRATPD has missing last value, not enough data. what are R^2 values. how tuned

to tease apart the effect of codependent features on the response. Also, recall that there are vast armies of business analysts and scientists at work that need to analyze data, in a manner akin to exploratory data analysis (EDA), that have no intention of creating a predictive model. Either they have no need, perhaps needing only partial dependence plots, or they do not have the expertise to choose, tune, and assess models (or write software at all).

Even in the case where a machine learning practitioner is available to create a fitted model for the analyst, hazards exist. First, if a fitted model is unable to accurately capture the relationship between features and y accurately, for whatever reason, then partial dependence does not provide any useful information to the user. To make interpretation more challenging, there is no definition of “accurate enough.” Second, given an accurate fitted model, business analysts and scientists are still peering at the data through the lens of the model, which can distort partial dependence curves. Separating visual artifacts of the model from real effects present in the data requires expertise in model behavior (and optimally in the implementation of model fitting algorithms).

Consider the combined FPD/ICE plots shown in Figure 1 derived from several models (random forest, gradient boosting, linear regression, deep learning) fitted to the same New York City rent data set Kaggle [2017]. The subplots in Figure 1(b)-(e) present starkly different partial dependence relationships and it is unclear which, if any, is correct. The marginal plot, (a), drawn directly from the data shows a roughly linear growth in price for a rise in the number of bathrooms, but this relationship is biased because of the dependence of bathrooms on other variables, such as the number of bedrooms. (e.g., five bathroom, one bedroom apartments are unlikely.) For real data sets with codependent features, the true relationship is unknown so it is hard to evaluate the correctness of the plots. (Humans are unreliable estimators, which is why we need data analysis algorithms in the first place.) Nonetheless, having the same algorithm, operating on the same data, give meaningfully different partial dependences is undesirable and makes one question their validity.

Experts are often able to quickly recognize model artifacts, such as the staircase phenomenon inherent to the decision tree-based methods in Figure 1(b) and (c). In this case, though, the staircase is more accurate than the linear relationship in (d) and (e) because the number of bathrooms is discrete (except for “half baths”). The point is that interpreting model-based partial dependence plots can be misleading, even for experts.

An accurate mechanism to compute partial dependences that did not peer through fitted models would be most welcome. Such partial dependence curves would be accessible to users, like business analysts, who lack the expertise to create suitable models and would also reduce the chance of plot misinterpretation due to model artifacts. The curves could also help machine learning practitioners to choose appropriate models based upon relationships exposed in the data.

In this paper, we propose a strategy, called STRATified Partial Dependence (STRATPD), that (i) computes partial dependences directly from training data (\mathbf{X}, \mathbf{y}) , rather than through the predictions of a fitted model, and (ii) does not presume mutually-independent features. As an example, Figure 1(f) shows the partial dependence plot computed by STRATPD. The

technique depends on the notion of an idealized partial dependence: integration over the partial derivative of y with respect to the variable of interest for the smooth function that generated (\mathbf{X}, y) . As that function is unknown, we estimate the partial derivatives from the data non-parametrically. Colloquially, the approach examines changes in y across x_j while holding $\mathbf{X}_{\setminus j}$ constant or very similar ($\mathbf{X}_{\setminus j}$ denotes all variables except x_j). A similar stratification approach works for categorical variables (CATSTRATPD). Both STRATPD and CATSTRATPD have $O(n^2)$ worst-case time complexity (like FPD), but STRATPD behaves linearly on real data sets. Our prototype is currently limited to regression, isolates only single-variable partial dependence, and cannot identify interaction effects (as ICE can). The software is available via Python package `stratx` with source code at [github](#), including the code to regenerate images in this paper.

We begin by describing and providing algorithms for the proposed stratification approach in Section 2 then compare STRATPD to related (model-based) work in Section 3. In Section 4, we present partial dependence curves generated by STRATPD and CATSTRATPD on real data sets, contrast the plots with those of existing methods, and use synthetic data to highlight biases in some model-based methods.

2 Partial dependence without model predictions

To estimate partial derivatives, STRATPD stratifies $\mathbf{X}_{\setminus j}$ feature space into disjoint regions of observations where all $\mathbf{X}_{\setminus j}$ variables are approximately matched across the observations in that region. Within each $\mathbf{X}_{\setminus j}$ region, any fluctuations in the response variable are likely due to the variable of interest, x_j . Estimates of the partial derivative within a region are computed discretely via the changes in y values between unique x_j positions. The overall partial derivative at $x_j = z$ is the average of all slopes, found in any region, whose x_j range spans z . Stratification occurs through the use of a decision tree but only for the purpose of partitioning feature space; STRATPD never uses predictions from any model.

Definition 1 The *idealized partial dependence* of y on feature x_j for smooth generator function $f : \mathbb{R}^p \rightarrow \mathbb{R}$ evaluated at $x_j = z$ is the cumulative sum up to z :

$$PD_j(z) = \int_{\min(x_j)}^z \frac{\partial y}{\partial x_j} dx_j \quad (1)$$

$PD_j(z)$ is the value contributed to y by x_j at $x_j = z$ and $PD_j(\min(x_j)) = 0$. The advantages of this partial dependence definition are that it does not depend on predictions from a fitted model and is insensitive to collinear or otherwise codependent features, unlike the Friedman’s original definition that he points out is less accurate for codependent data sets. We will denote Friedman’s as FPD_j to distinguish it from this ideal, PD_j .

For example, consider quadratic equation $y = x_1^2 + x_2 + 100$ as a generator of data in $[0, 3]$. The partial derivatives are $\frac{\partial y}{\partial x_1} = 2x_1$ and $\frac{\partial y}{\partial x_2} = 1$, giving $PD_1 = x_1^2$ and $PD_2 = x_2$.

The obvious disadvantage of this feature impact definition is that function f , from which PD_j is derived, is unknown in practice, so symbolically computing the partial derivatives is not possible. But, if we could compute accurate partial dependence curves by some other method, then this definition would still represent a viable means to obtain feature impacts.

STRATPD stratifies a data set into groups of observations that are similar, except in the variable of interest, x_j , through the use of a single decision tree. Any fluctuation of the response variable within a group (decision tree leaf) is likely due to x_j . The β_1 coefficient of a simple local linear regression fit to the (x_j, y) values within a group provides an estimate of $\frac{\partial y}{\partial x_j}$ in that group’s x_j range. Averaging the partial derivative estimates across all such groups yields the overall $\frac{\partial y}{\partial x_j}$ partial derivative approximation. The cumulative sum of the estimated partial derivative yields the partial dependence curve.

StratPD

Fit tree regressor to all but `x_c` with hyper parameter `min_slopes_per_x`

```

For each leaf:
  y bar = Group leaf samples by x_c, computing average y per unique x_c
  dx = discrete difference between adjacent unique x_c
  dy = discrete difference between adjacent average y bar
  add (x[i], x[i+1], dy[i]/dx[i]) for each unique x_c to list D

for each x in unique x_c from X:
  slopes = [slope for (a, b, slope) in D if x >= a and x < b]
  count[x] = |slopes|
  dydx[x] = mean(slopes)

Drop slope estimates computed using fewer than min_slopes_per_x values
pdx = discrete difference between adjacent unique x_c
pdy = cumulative sum of dydx * pdx
return pdx, [0]+pdy // insert 0 for pdx[0] since sum contributed from beyond left is 0

```

CatStratPD

```

Fit tree regressor to all but x_c with hyper parameter min_slopes_per_x
For each leaf:
  y bar = Group leaf samples by categories of x_c, computing average y per unique category x_c
  Compute unique categories and counts per category
  refcat is randomly chosen category from x_c
  For each unique category x in leaf:
    delta[cat,leaf] = Subtract y for refcat from all y bar (refcat delta will be 0)
end
Let Avg[cat] be vector with running sum mapping category to count
work = set of leaf indexes
while more work and something changed and less than max iterations:
  for each leaf in leaves:
    if cat in delta[:,leaf] intersects with Avg:
      j = random category in intersection
      adjust delta[:,leaf] to be relative to j so delta[j,leaf]==0 then add Avg[j] so comparable
      merge into Avg
  work -= all j merged this iteration

```

3 Related work

FPD

ICE

ALE

SHAP

cold start, counting execution time and number of hyper parameters. particularly deep learning

The techniques differ in algorithm simplicity, performance, and ability to isolate codependent variables. a nonparametric technique could also inform which machine learning model to use if a model is desired.

SHAP is mean centered FPD for independent variables, proof in supplemental material.

4 Experimental results

what if X,y relationship is very weak? models would get low accuracy. what happens to us? I think we would simply show low partial dependence curves.

5 Discussion and future work

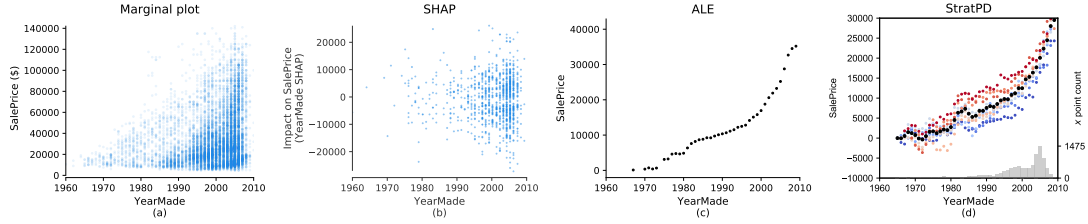


Figure 2: (a) Marginal plot of bulldozer **YearMade** versus **SalePrice** using subsample of 20k observations, (b) partial dependence drawn by SHAP interrogating an RF with 40 trees and explaining 1000 values with 100 observations as background data, (c) STRATPD partial dependence.

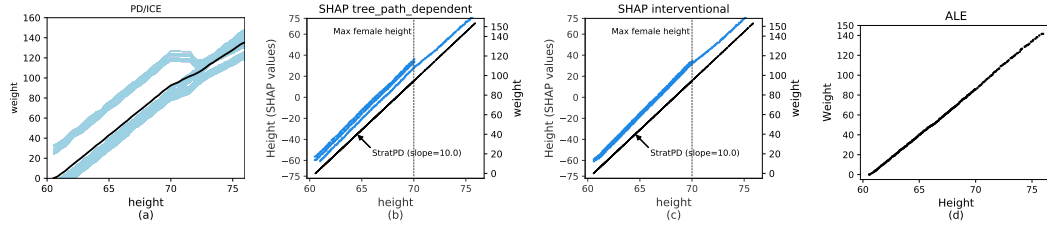


Figure 3: SHAP partial dependence plots of response body weight on feature **height** using 2000 synthetic observations from Equation (??). SHAP interrogated an RF with 40 trees and explained all 2000 samples; the interventional case used 100 observations as background data.

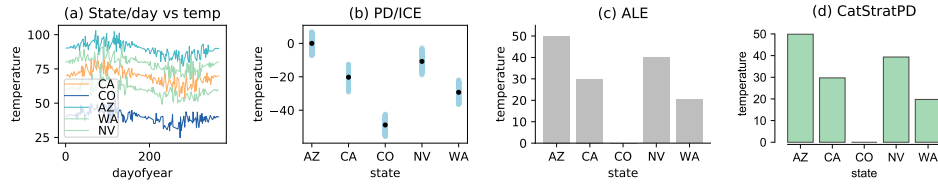


Figure 4: foo.

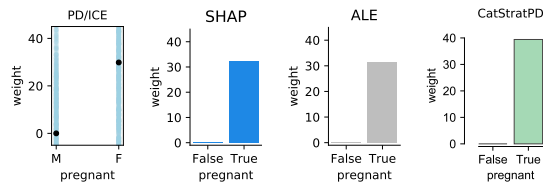


Figure 5: foo.

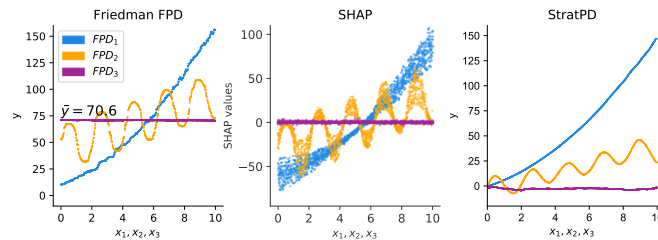


Figure 6: $y = x_1^2 + x_1x_2 + 5x_1\sin(3x_2) + 10$ where $x_1, x_2, x_3 \sim U(0, 10)$ and x_3 does not affect y . No noise added.

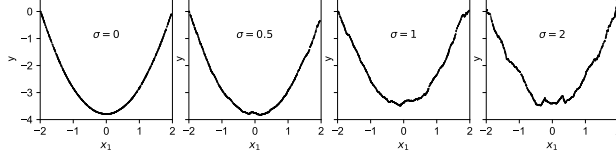


Figure 7: $y = x_1^2 + x_1 + 10 + N(0, \sigma)$ where $x_1, x_2 \sim U(-2, 2)$ and $\sigma \in [0, 0.5, 1, 2]$.

6 Appendix

Algorithm: *StratPD*

Input: $\mathbf{X}, \mathbf{y}, c, \text{min_samples_leaf}, \text{min_slopes_per_x}$

Output: pdx, pdy : Unique x_c , partial dependence values across x_c

$T :=$ Decision tree regressor fit to $(\mathbf{X}_{\bar{c}}, \mathbf{y})$ with hyper-parameter: min_samples_leaf

for each leaf $l \in T$ **do**

$(\mathbf{x}_l, \mathbf{y}_l) = \{(x_c^{(i)}, y^{(i)})\}_{i \in l}$ // Get leaf samples

$\mathbf{ux} := \text{unique}(\mathbf{x}_l)$

$\bar{\mathbf{y}} :=$ Group leaf records $(\mathbf{x}_l, \mathbf{y}_l)$ by value of \mathbf{x}_l , computing \bar{y} per unique value

$\mathbf{dx} := \mathbf{ux}^{(i+1)} - \mathbf{ux}_{i=1..|\mathbf{ux}|-1}^{(i)}$ // Discrete difference

$\mathbf{dy} := \bar{\mathbf{y}}^{(i+1)} - \bar{\mathbf{y}}_{i=1..|\mathbf{ux}|-1}^{(i)}$

 Add tuples $(\mathbf{ux}^{(i)}, \mathbf{ux}^{(i+1)}, \mathbf{dy}^{(i)}/\mathbf{dx}^{(i)})_{i=1..|\mathbf{ux}|-1}$ to list \mathbf{d}

end

$\mathbf{ux} := \text{unique}(\{x_c^{(i)}\}_{i=1..n})$

for each $x \in \mathbf{ux}$ // Counts slopes, compute average slope per unique x_c value

do

$\text{slopes} := [\text{slope for } (a, b, \text{slope}) \in \mathbf{d} \text{ if } x \geq a \text{ and } x < b]$

$\mathbf{c}_x := |\text{slopes}|$

$\text{dydx}_x := \overline{\text{slopes}}$

end

$\text{dydx} := \text{dydx}[\mathbf{c} \geq \text{min_slopes_per_x}]$ // Drop slope estimates computed from too few

$\mathbf{ux} := \mathbf{ux}[\mathbf{c} \geq \text{min_slopes_per_x}]$

$\text{pdx} := \mathbf{ux}^{(i+1)} - \mathbf{ux}_{i=1..|\mathbf{ux}|-1}^{(i)}$

$\text{pdy} := [0] + \text{cumulative_sum}(\text{dydx} * \text{pdx})$ // integrate, inserting 0 for leftmost x_c

return pdx, pdy

References

- Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.
- Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015. doi: 10.1080/10618600.2014.907095. URL <https://doi.org/10.1080/10618600.2014.907095>.
- Daniel W Apley. Visualizing the effects of predictor variables in black box supervised learning models. *arXiv preprint arXiv:1612.08468*, 2016.
- Scott M Lundberg and Su-In 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, editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc., 2017. URL <http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf>.
- Kaggle. Two sigma connect: Rental listing inquiries. <https://www.kaggle.com/c/two-sigma-connect-rental-listing-inquiries>, 2017. Accessed: 2019-06-15.

Algorithm: *CatStratPD*

Input: $\mathbf{X}, \mathbf{y}, c, \text{min_samples_leaf}$

Output: $\Delta^{(k)}$ = category k 's effect on y where $\text{mean}(\Delta^{(k)}) = 0$

$n^{(k)}$ = number of supported observations per category k

T := Decision tree regressor fit to $(\mathbf{X}_{\bar{c}}, \mathbf{y})$ with hyper-parameter: *min_samples_leaf*

// Get average y delta relative to random ref category for each sample in each leaf

Let $\Delta_{x,l}$ be dictionary mapping (category, leaf) to delta from ref category

Let $\text{Count}_{x,l}$ be dictionary mapping (category, leaf) to count

for each leaf $l \in T$ **do**

$(\mathbf{x}_l, \mathbf{y}_l) = \{(x_c^{(i)}, y^{(i)})\}_{i \in l}$ // Get leaf samples

$\mathbf{ux}, \mathbf{cx} := \text{unique}(\mathbf{x}_l)$ // Get unique categories, counts from leaf samples

$\bar{\mathbf{y}} :=$ Group leaf records $(\mathbf{x}_l, \mathbf{y}_l)$ by categories of \mathbf{x}_l , computing \bar{y} per unique category

$\text{refcat}_l :=$ random category from \mathbf{y}

for each $x \in \mathbf{ux}$ **do**

$\text{Count}_{x,l} := \mathbf{cx}_x$

$\Delta_{x,l} := \bar{\mathbf{y}} - y[\text{refcat}_l]$

end

end

work := 1 .. $|\text{uniq_refcats}|$

Let Avg_x be vector with running sum mapping category to count

while $\text{len}(\text{work}) > 0$ and $\text{len}(\text{completed}) > 0$ and $\text{iteration} \leq \text{max_iter}$ **do**

end