Technical Report: A Stratification Approach to Partial Dependence for Codependent Variables

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

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

Abstract

1 Introduction

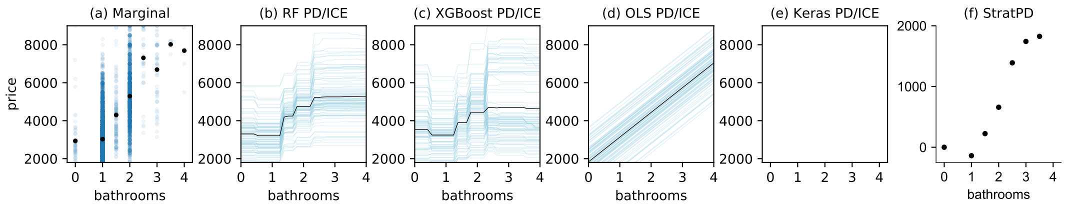

Figure 1: Plots of bathrooms versus rent price using New York City apartment rent data. (a) marginal plot, (b) PD/ICE plot derived from random forest, (c) PD/ICE plot derived from gradient boosted machine, and (d) PD/ICE plot derived from ordinary least squares regression; sample size is 10,000 observations 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.

partial dependence is important because...

Existing techniques, such as FPD, ICE, ALE, SHAP peer through the lens of a model's predictions. For the same data applying the same technique but using different models, we get different answers, which calls into question the validity of the curves.

key is "all else being equal", which implies you don't want curves affected by other variables. Interaction plots are also very useful, such as ICE, but here our goal is the pure partial dependence curve. In the future, we hope to consider extracting interaction between variables like SHAP.

Many analysts do not need a predictive model nor would they know how to choose, tune, and assess a model. Could also be the case that a technique is not available in the desired deployment environment. 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.

we introduce an ideal definition of partial dependence that does not rely on predictions from a fitted model based upon partial derivatives and then estimate partial derivatives nonparametrically to get partial dependence. The technique seems to isolate variables well and has linear behavior for numeric variables and mildly quadratic behavior for categorical variables in practice. The theoretical complexity is $O(n^2)$ like FPD.

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

state up front it only gets pure partial dependence, no interaction and has quadratic theoretical complexity, but it has the advantage that it doesn't require a fitted model. Sometimes there is an advantage to a model, smoothing etc. But, in many cases lack of model increases the accessibility of the tool to analysts and could prevent nonexpert machine learning practitioners from interpretation errors from poorly fit or tuned models.

2 Partial dependence without model predictions

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

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

$PD_j(z)$ 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.

3 Existing work

FPD

ICE

ALE

SHAP

4 Experimental results

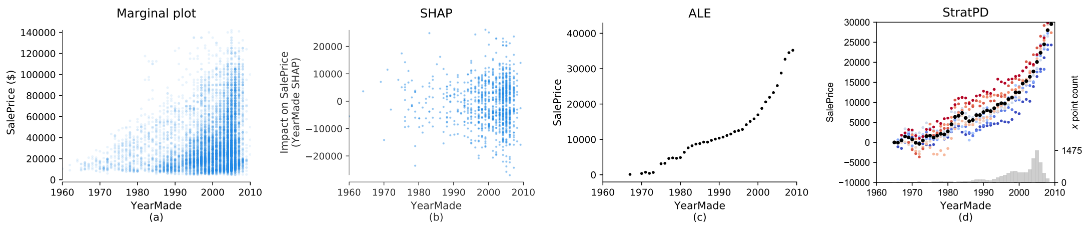

Figure 2: (a) Marginal plot of bulldozer `YearMade` versus `SalePrice` using subsample of 20k observations, (b) partial dependence drawn by SHAP interrogating an RF with 40 trees and explaining 1000 values with 100 observations as background data, (c) StratPD partial dependence.

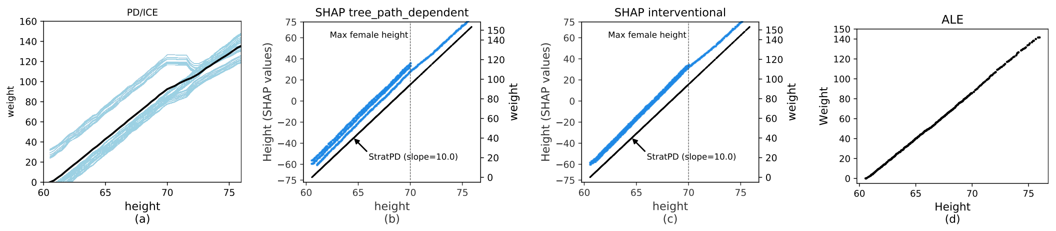

Figure 3: SHAP partial dependence plots of response body weight on feature `height` using 2000 synthetic observations from Equation (**??**). SHAP interrogated an RF with 40 trees and explained all 2000 samples; the interventional case used 100 observations as background data.

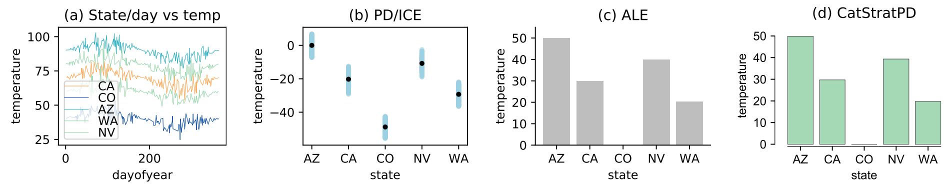

Figure 4: foo.

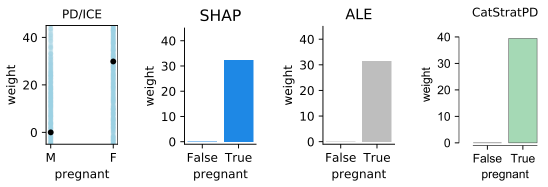

Figure 5: foo.

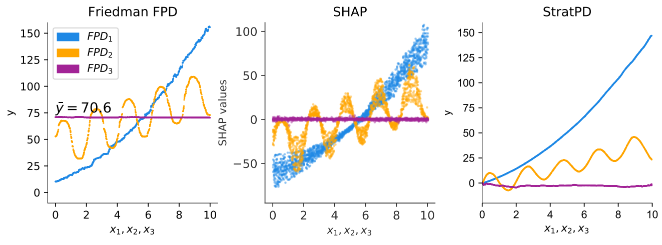

Figure 6: $y = x_1^2 + x_1 x_2 + 5x_1 sin(3x_2) + 10$ where $x_1, x_2, x_3 \sim U(0, 10)$ and x_3 does not affect y. No noise added.

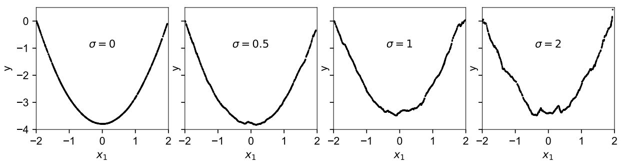

Figure 7: $y = x_1^2 + x_1 + 10 + N(0, \sigma)$ where $x_1, x_2 \sim U(-2, 2)$ and $\sigma \in [0, 0.5, 1, 2]$.

5 Algorithms

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
```

```
Algorithm: StratPD
Input: $\mathbf{X}$, $\mathbf{y}$, c, min_samples_leaf, min_slopes_per_x
Output: $\mathbf{pdx}, \mathbf{pdy}$: Unique $x_c$, partial dependence values across $x_c$
$T$ := Decision tree regressor fit to ($\mathbf{X}_{\overline{c}}$, $\mathbf{y}$) 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
    $\mathbf{ux} := unique(\mathbf{x}_l)$
    $\bar{\mathbf{y}}$ := Group leaf records $(\mathbf{x}_l, \mathbf{y}_l)$ by value of $\mathbf{x}_l$, computing $\bar{y}$ per unique value
    $\mathbf{dx} := \mathbf{ux}^{(i+1)} - \mathbf{ux}^{(i)}_{i=1..|\mathbf{ux}|-1}$    // Discrete difference
    $\mathbf{dy} := \bar{\mathbf{y}}^{(i+1)} - \bar{\mathbf{y}}^{(i)}_{i=1..|\mathbf{ux}|-1}$
    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
$\mathbf{ux} := unique(\{x_c^{(i)}\}_{i=1..n})$
for each $x \in \mathbf{ux}$    // Counts slopes, compute average slope per unique $x_c$ value
do
    $slopes := [slope$ for $(a, b, slope) \in \mathbf{d}$ if $x \geq a$ and $x < b]$
    $\mathbf{c}_x := |slopes|$
    $\mathbf{dydx}_x := \overline{slopes}$
end
$\mathbf{dydx} := \mathbf{dydx}[\mathbf{c} \geq min\_slopes\_per\_x]$    // Drop slope estimates computed from too few
$\mathbf{ux} := \mathbf{ux}[\mathbf{c} \geq min\_slopes\_per\_x]$
$\mathbf{pdx} := \mathbf{ux}^{(i+1)} - \mathbf{ux}^{(i)}_{i=1..|\mathbf{ux}|-1}$
$\mathbf{pdy} := [0]$ + cumulative_sum$(\mathbf{dydx} * \mathbf{pdx})$    // integrate, inserting 0 for leftmost $x_c$
return $\mathbf{pdx}, \mathbf{pdy}$
```

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
```

```
Algorithm: CatStratPD
Input: $\mathbf{X}, \mathbf{y}, c, min\_samples\_leaf$
Output: $\Delta^{(k)}$ = category $k$'s effect on $y$ where $mean(\Delta^{(k)}) = 0$
        $n^{(k)}$ = number of supported observations per category $k$
$T$ := Decision tree regressor fit to $(\mathbf{X}_{\overline{c}}, \mathbf{y})$ 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 $Count_{x,l}$ 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
    $\mathbf{ux}$, $\mathbf{cx}$ := $unique(\mathbf{x}_l)$    // 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
    $refcat_l$ := random category from $\mathbf{y}$
    for each $x \in \mathbf{ux}$ do
        $Count_{x,l}$ := $\mathbf{cx}_x$
        $\Delta_{x,l}$ := $\bar{\mathbf{y}} - y[refcat_l]$
    end
end
work := 1 .. $|uniq\_refcats|$
Let $Avg_x$ be vector with running sum mapping category to count
while len(work) > 0 and len(completed)>0 and iteration<=max_iter do
end
```

References