Monte-Carlo Methods for Prediction Functions

by Zachary M. Jones

Abstract An abstract of less than 150 words.

Many methods for estimating prediction functions produce estimated functions which are not directly human-interpretable because of their complexity: they may include high-dimensional interactions and/or complex nonlinearities. While a learning method's capacity to automatically learn interactions and interactions is attractive when the goal is prediction, there are many cases where users want good predictions *and* the ability to understand how predictions depend on the features. `mmpf` implements general methods for interpreting prediction functions using Monte-Carlo methods. These methods so that any function which generates predictions can be interpreted. `mmpf` is currently used in other packages for machine learning like `edarf` and `mlr` (Jones and Linder, 2016; Bischl et al., 2016).

Marginalizing Prediction Functions

The core function of `mmpf`, `marginalPrediction`, allows marginalization of a prediction function so that it depends on a subset of the features. Say the matrix of features $\mathbf{X}$ is partitioned into two subsets, $\mathbf{X}_u$ and $\mathbf{X}_{-u}$, where the former is of primarty interest. A prediction function f which in the regression case maps $\mathbf{X} \rightarrow \mathbf{y}$, where $\mathbf{y}$ is a real-valued vector, can be estimated from $(\mathbf{X}, \mathbf{y})$ giving $\hat{f}$, which might not be additive or linear in the columns of $\mathbf{X}_u$. If we knew the joint distribution $\mathbb{P}(\mathbf{X}, \mathbf{y})$ we could marginalize f using the joint distribution giving f_u directly.

$$f_u(\mathbf{X}_u) = \int f(\mathbf{X}_u, \mathbf{X}_{-u}) \mathbb{P}(\mathbf{X}) d\mathbf{X}_{-\mathbf{u}}$$

However, we instead have $\hat{f}$ and also don't know the joint distribution of $(\mathbf{X}, \mathbf{y})$. If we assume that $\mathbb{P}(\mathbf{X}, \mathbf{y}) = \mathbb{P}(\mathbf{X}_u, \mathbf{y}) \mathbb{P}(\mathbf{X}_{-u}, \mathbf{y})$, then we can instead compute

$$f_u(X_u) = \int f(\mathbf{X}_u, \mathbf{X}_{-u}) \mathbb{P}(\mathbf{X}_{-u}) d\mathbf{X}_{-\mathbf{u}}$$

which relies only on knowing the marginal distribution of $\mathbf{X}_{-u}$.

f_u can be approximated using $\hat{f}$ and the empirical marginal distribution of $\mathbf{X}_u$ using Monte-Carlo integration.

$$\hat{f}_u(\mathbf{X}_u) = \sum_{i=1}^{N} \hat{f}(\mathbf{X}_u, \mathbf{X}_{-u}^{(i)})$$

Partial dependence is the result of this type of procedure; it is the expected value of the marginalized prediction function $\hat{f}_u$ evaluated at values of $\mathbf{X}_u$ Friedman (2001). `marginalPrediction` allows users to compute partial dependence and any other function of this marginalized function easily.

```
library(mmpf)
library(randomForest)

data(iris)
fit = randomForest(Species ~ ., data = iris)

mp = marginalPrediction(data = iris[, -ncol(iris)],
  vars = "Petal.Width",
  n = c(10, nrow(iris)), model = fit, uniform = TRUE,
  predict.fun = function(object, newdata) predict(object, newdata, type = "prob"))
print(mp)
##     Petal.Width    setosa versicolor virginica
##  1:   0.1000000 0.6374133  0.2337733 0.1288133
##  2:   0.3666667 0.6374133  0.2337733 0.1288133
##  3:   0.6333333 0.6356267  0.2350533 0.1293200
##  4:   0.9000000 0.1707200  0.5997333 0.2295467
##  5:   1.1666667 0.1688267  0.6016267 0.2295467
##  6:   1.4333333 0.1688133  0.5880800 0.2431067
```

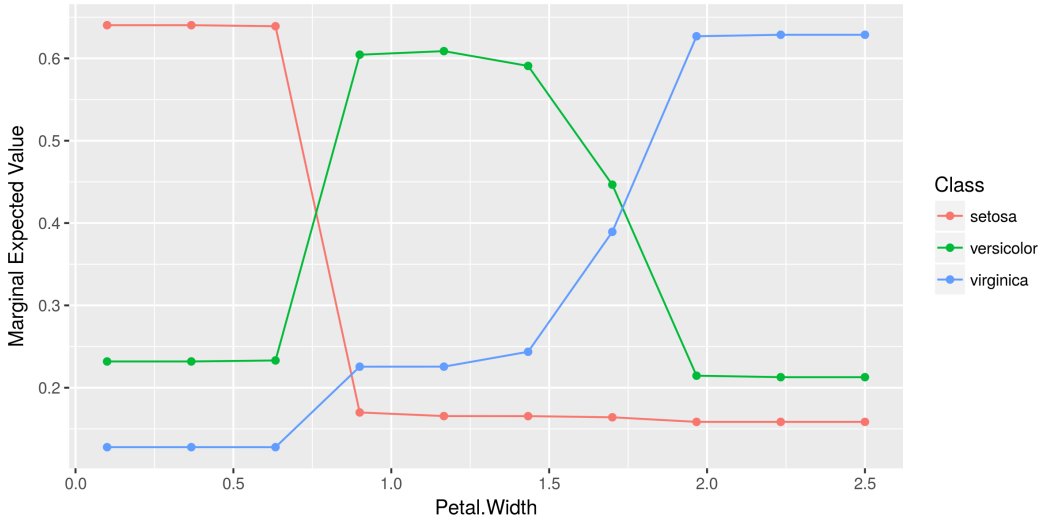

Figure 1: The expected value of $\hat{f}$ estimated by a random forest and marginalized by Monte-Carlo integration to depend only on "Petal.Width."

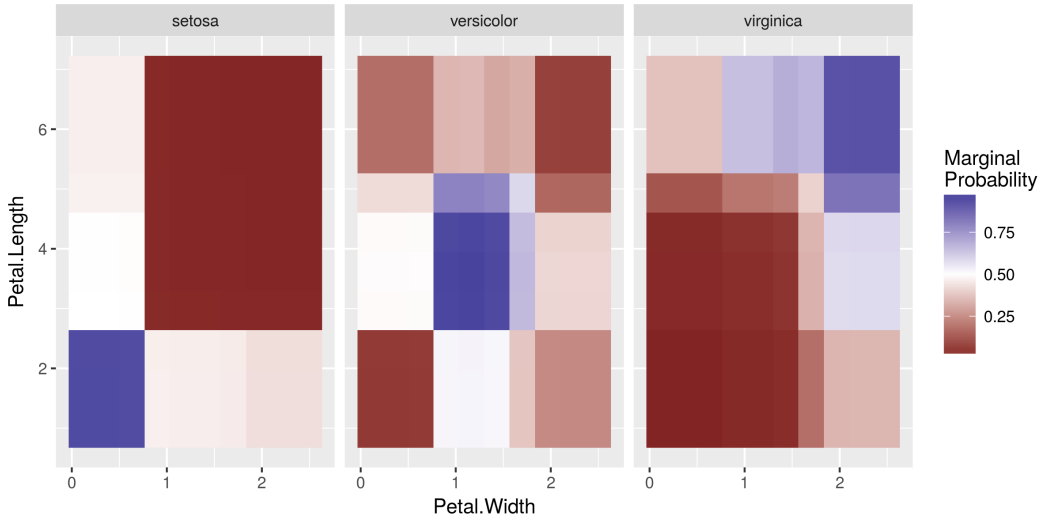

Figure 2: The expected value of $\hat{f}$ estimated by a random forest and marginalized by Monte-Carlo integration to depend only on "Petal.Width" and "Petal.Length."

```
##  7:   1.7000000 0.1640400  0.4242800 0.4116800
##  8:   1.9666667 0.1619867  0.2066667 0.6313467
##  9:   2.2333333 0.1619867  0.2047867 0.6332267
## 10:   2.5000000 0.1619867  0.2047867 0.6332267
```

If the prediction function is marginalized so that it depends on a low dimensional subset of the features, for example one or two dimensions, then it can be easily visualized, as in Figure 1.

As mentioned earlier, *any* function of the marginalized function can be computed, including vector-valued functions. For example the expectation and variance of the marginalized function can be simultaneously computed, the results of which are shown in Figures 2 and 3. Computing the variance of a marginalized function can be used for detecting interactions between $\mathbf{X}_u$ and $\mathbf{X}_{-u}$ (...).

```
mp.int = marginalPrediction(data = iris.features,
  vars = c("Petal.Width", "Petal.Length"),
  n = c(10, nrow(iris)), model = fit, uniform = TRUE,
  predict.fun = function(object, newdata) predict(object, newdata, type = "prob"),
  aggregate.fun = function(x) list("mean" = mean(x), "variance" = var(x)))
```

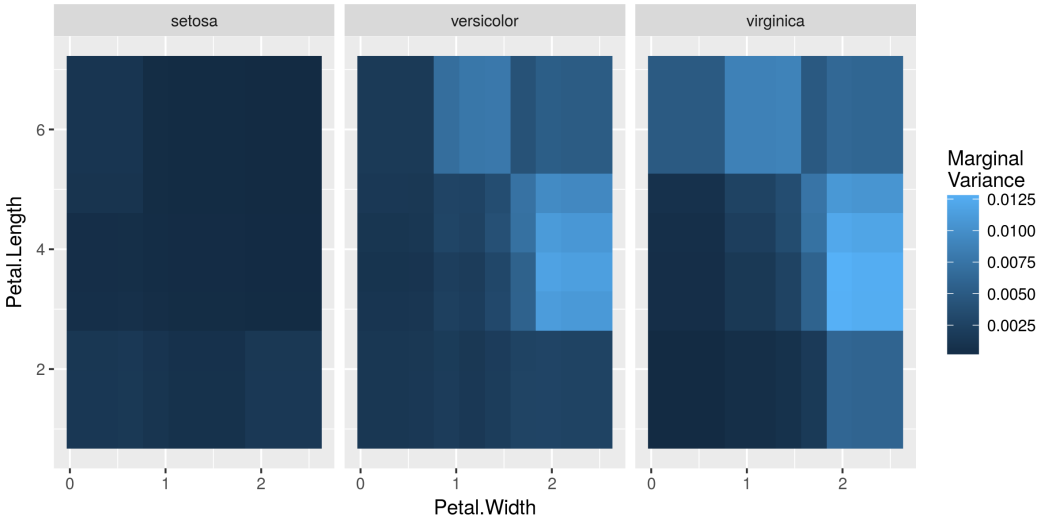

Figure 3: The variance of $\hat{f}$ estimated by a random forest and marginalized by Monte-Carlo integration to depend only on "Petal.Width" and "Petal.Length." Non-constant variance indicates interaction between these variables and those marginalized out of $\hat{f}$.

Monte-Carlo Integration on Non-Uniform Measures

`marginalPrediction` can also perform Monte-Carlo integration against non-uniform probability measures which are constructed from the training data. This can help avoid problems from the product distribution assumption. When $\mathbb{P}(\mathbf{X}) \neq \mathbb{P}(\mathbf{X}_u)\mathbb{P}(\mathbf{X}_{-u})$ assuming that this is the case can result in evaluating in a $\hat{f}_u$ which is constructed from model behavior in regions of the of the feature space which occur with low probability under $\mathbb{P}(\mathbf{X})$, making $\hat{f}_u$ depend heavily on the behavior of the method used to learn $\hat{f}$.

As a simple example consider the case of data from a deterministic function which is linear across its support. Specifically, $\mathbf{y} = f(\mathbf{X}) = \mathbf{X}\mathbf{w}$ where $\mathbf{X}$ is a two dimensional matrix of features each of which is drawn from an independent uniform distribution on $0, 1$ and $\mathbf{w}$ is a vector of real-valued weights. Suppose that there is an outlier which is extreme in both X_1 and y, but perfectly normal in X_2: $X_1 = 1.5$, $X_2 = .5$, and, $y = X_1^2 + X_2$. Figure 4 shows the empirical distribution of $\mathbf{X}$. Some methods for constructing $\hat{f}$ will strongly depend on this outlying point, as is the case in this example. Using support vector regression and a polynomial kernel results in dramatic nonlinearity when $X_1 > 1$. If we compute unweighted partial dependence assume a discrete uniform distribution on X_1, which includes the outlying point at $X_1 = 1.5$.

Although this outlier is easy to see, because this problem is low-dimensional, in a higher dimensional space outliers are difficult to detect visually. In these cases Monte-Carlo methods for interpreting $\hat{f}$ will also depend heavily on these sorts of points. Constructing a weight function, which takes points of the same dimension as $\mathbf{X}$ and outputs a scalar which encodes some notion of "closeness" to the training data can help alleviate this problem, as shown in Figure 5.

With observed outlying points model behavior can be controlled by choosing a loss function which does not heavily weight outlying points. However, this is not always possible or desirable. Outliers can also be generated by the aforementioned product distribution assumption. This can be especially problematic when features are highly correlated, making their joint distribution far from uniform.

Bibliography

B. Bischl, M. Lang, L. Kotthoff, J. Schiffner, J. Richter, E. Studerus, G. Casalicchio, and Z. M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html. [p]

J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001. [p]

Z. M. Jones and F. J. Linder. edarf: Exploratory data analysis using random forests. *The Journal of Open Source Software*, 1(6), oct 2016. doi: 10.21105/joss.00092. URL http://dx.doi.org/10.21105/joss.00092. [p]

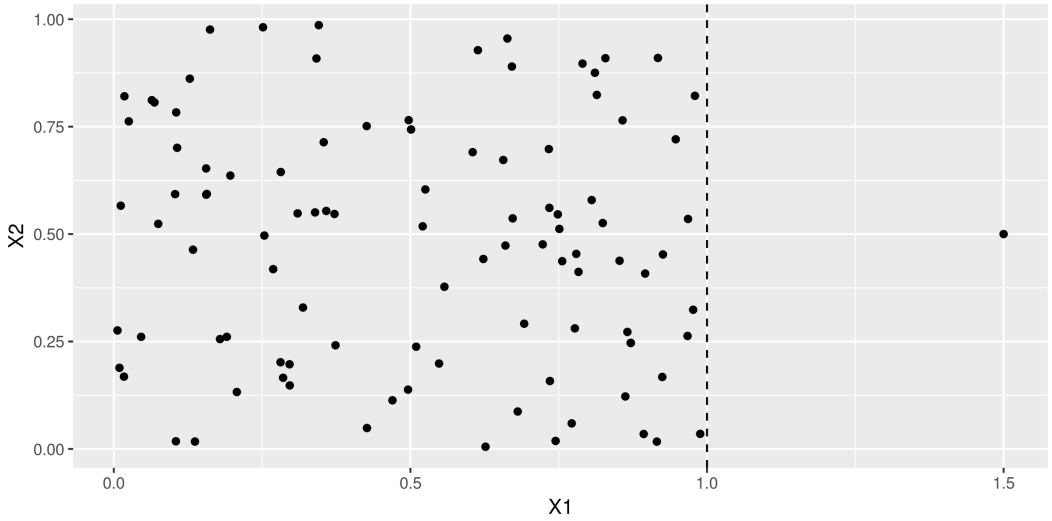

Figure 4: The joint distribution of the features, both of which are independent uniform draws, with the except of one point at $X_1 = 1.5, X_2 = .5$, which lies outside the support and is an outlier.

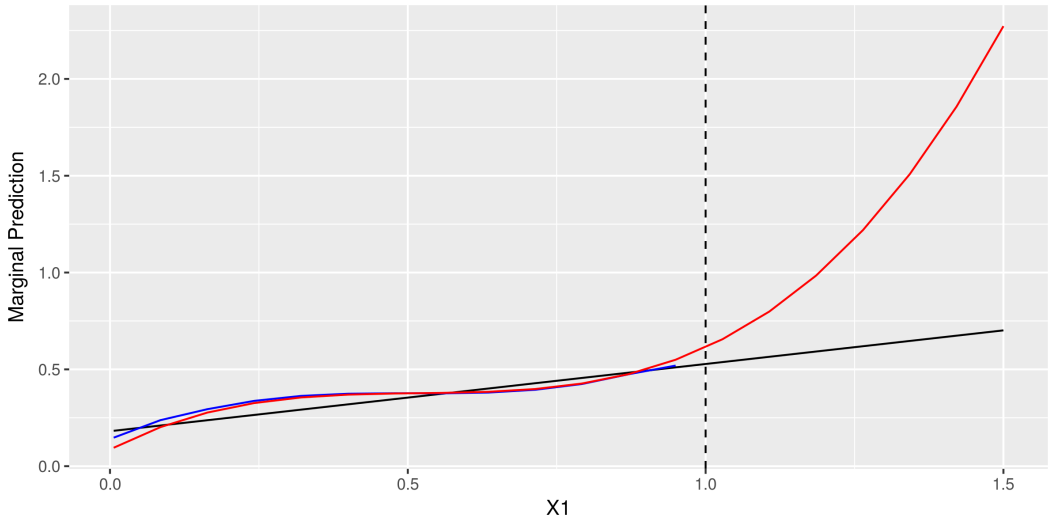

Figure 5: $\hat{f}$ marginalized to depend only on $\mathbf{X}_1$ without weighting, shown in red, or with weighting based on Tukey's half-space depth, shown in blue. The true marginal relationship is shown in black and the dashed line shows the true bounds of the support of X_1.

Zachary M. Jones
Pennsylvania State University
University Park, Pennsylvania
United States
zmj@zmjones.com