Variable Importance Plots: An Introduction to the vip Package

by Brandon M. Greenwell and Bradley C. Boehmke

Abstract In the era of big data, it is becoming more of a challenge to not only build state-of-the-art predictive models, but also gain an understanding of whats really going on in the data. For example, it is often of interest to know which, if any, of the predictors in a fitted model are relatively influential on the predicted outcome. Some modern algorithmslike random forests and gradient boosted decision treeshave a natural way of quantifying the importance or relative influence of each feature. Other algorithmslike naive Bayes classifiers and support vector machinesare not capable of doing so and model-agnostic approaches are generally used to measure each predictors importance. Enter **vip**, an R package for constructing variable importance scores/plots for many types of supervised learning algorithms using model-based and novel model-agnostic approaches. We'll also discuss a novel way to display both feature importance and feature effects together using sparklines.

Introduction

Variable importance (VI).

For illustration, we use one of the regression problems described in Friedman (1991) and Breiman (1996). These data are available in the **mlbench package** (Leisch and Dimitriadou, 2012). The inputs consist of 10 independent variables uniformly distributed on the interval $[0, 1]$; however, only 5 out of these 10 are actually used in the true model. Outputs are created according to the formula described in ?mlbench::mlbench.friedman1. The code chunk below simulates 500 observations from the model default standard deviation.

```
# Simulate training data
set.seed(101)  # for reproducibility
trn <- as.data.frame(mlbench::mlbench.friedman1(500))  # ?mlbench.friedman1

# Inspect data
tibble::as_tibble(trn)
#> Warning: `as.tibble()` is deprecated, use `as_tibble()` (but mind the new semantics).
#> This warning is displayed once per session.
#> # A tibble: 500 x 11
#>       x.1   x.2   x.3   x.4    x.5     x.6   x.7   x.8   x.9  x.10     y
#>     <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#>  1 0.372  0.406 0.102 0.322 0.693   0.758  0.518 0.530 0.878 0.763 14.9
#>  2 0.0438 0.602 0.602 0.999 0.776   0.533  0.509 0.487 0.118 0.176 15.3
#>  3 0.710  0.362 0.254 0.548 0.0180  0.765  0.715 0.844 0.334 0.118 15.1
#>  4 0.658  0.291 0.542 0.327 0.230   0.301  0.177 0.346 0.474 0.283 10.7
#>  5 0.250  0.794 0.383 0.947 0.462   0.00487 0.270 0.114 0.489 0.311 17.6
#>  6 0.300  0.701 0.992 0.386 0.666   0.198  0.924 0.775 0.736 0.974 18.3
#>  7 0.585  0.365 0.283 0.488 0.845   0.466  0.715 0.202 0.905 0.640 14.6
#>  8 0.333  0.552 0.858 0.509 0.697   0.388  0.260 0.355 0.517 0.165 17.0
#>  9 0.622  0.118 0.490 0.390 0.468   0.360  0.572 0.891 0.682 0.717  8.54
#> 10 0.546  0.150 0.476 0.706 0.829   0.373  0.192 0.873 0.456 0.694 15.0
#> #  with 490 more rows
```

Model-specific VI

Some machine learning algorithms have their own way of quantifying variable Importance. We describe some of these in the subsection that follow. The issue with model-specific VI scores is that they are not necessarily comparable across different types of models. For example, directly computing the impurity-based VI scores from tree-based models to the -statistic from linear models.

Trees and tree ensembles

Decision trees probably offer the most natural model-based approach to quantifying the importance of each feature. In a binary decision tree, at each node , a single predictor is used to partition the data

into two homogeneous groups. The chosen predictor is the one that maximizes some measure of improvement . The relative importance of predictor is the sum of the squared improvements over all internal nodes of the tree for which was chosen as the partitioning variable; see Breiman, Friedman, and Charles J. Stone (1984) for details. This idea also extends to ensembles of decision trees, such as RFs and GBMs. In ensembles, the improvement score for each predictor is averaged across all the trees in the ensemble. Fortunately, due to the stabilizing effect of averaging, the improvement-based VI metric is often more reliable in large ensembles (see Hastie, Tibshirani, and Friedman 2009, pg. 368). RFs offer an additional method for computing VI scores. The idea is to use the leftover out-of-bag (OOB) data to construct validation-set errors for each tree. Then, each predictor is randomly shuffled in the OOB data and the error is computed again. The idea is that if variable is important, then the validation error will go up when is perturbed in the OOB data. The difference in the two errors is recorded for the OOB data then averaged across all trees in the forest.

To illustrate, we fit a CART-like regression tree, RF, and GBM to the simulated training data. (Note: there are a number of different packages available for fitting these types of models, we just picked popular and efficient implementations for illustration.)

```
# Load required packages
library(rpart)         # for fitting CART-like decision trees
library(randomForest)  # for fitting random forests
library(xgboost)       # for fitting GBMs

# Fit a single regression tree
tree <- rpart(y ~ ., data = trn)

# Fit a random forest
set.seed(101)
rfo <- randomForest(y ~ ., data = trn, importance = TRUE)

# Fit a GBM
set.seed(102)
bst <- xgboost(
  data = data.matrix(subset(trn, select = -y)),
  label = trn$y,
  objective = "reg:linear",
  nrounds = 100,
  max_depth = 5,
  eta = 0.3,
  verbose = 0  # suppress printing
)
```

Each of the above packages include the ability to compute VI scores for all the features in the model; however, the implementation is rather package specific, as shown in the code chunk below. The results are displayed in Figure 1.

```
# Extract VI scores from each model
vi_tree <- tree$variable.importance
vi_rfo <- rfo$variable.importance
vi_bst <- xgb.importance(model = bst)
```
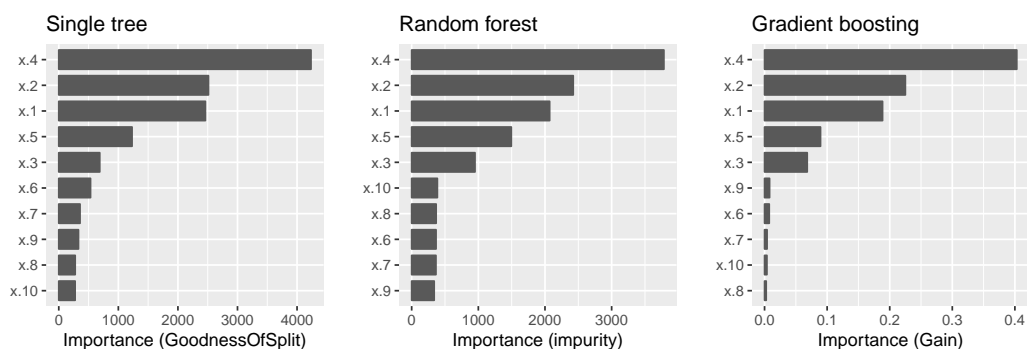

Figure 1: Model-based VIPs for the three different tree-based models.

As we would expect, all three methods rank the variables x.1–x.5 as more important than the others. While this is good news, it is unfortunate that we have to remember the different functions and ways of extracting and plotting VI scores from various model fitting functions. This is where **vip** (**?**) can help...one function to rule them all! Once **vip** is loaded, we can use vi() to extract a tibble of VI scores.

```
library(vip)

vi(tree)  # CART-like decision tree
#> # A tibble: 10 x 2
#>    Variable Importance
#>    <chr>         <dbl>
#>  1 x.4           4234.
#>  2 x.2           2513.
#>  3 x.1           2461.
#>  4 x.5           1230.
#>  5 x.3            688.
#>  6 x.6            533.
#>  7 x.7            357.
#>  8 x.9            331.
#>  9 x.8            276.
#> 10 x.10           275.

vi(rfo)   # RF
#> # A tibble: 10 x 2
#>    Variable Importance
#>    <chr>         <dbl>
#>  1 x.4            74.2
#>  2 x.2            59.9
#>  3 x.1            53.3
#>  4 x.5            37.1
#>  5 x.3            22.5
#>  6 x.9            1.05
#>  7 x.10           0.254
#>  8 x.8           -0.408
#>  9 x.7           -1.56
#> 10 x.6           -2.00

vi(bst)   # GBM
#> # A tibble: 10 x 2
#>    Variable Importance
#>    <chr>         <dbl>
#>  1 x.4           0.403
#>  2 x.2           0.225
#>  3 x.1           0.189
#>  4 x.5           0.0894
#>  5 x.3           0.0682
#>  6 x.9           0.00802
#>  7 x.6           0.00746
#>  8 x.7           0.00400
#>  9 x.10          0.00377
#> 10 x.8           0.00262
```

Notice how the vi() function always returns a tibble with two columns: Variable and Importance. Also, by default, vi() always orders the VI scores from highest to lowest; this, among other options, can be controlled by the user (see ?vip::vi for details). Plotting VI scores with vip() is just as straightforward. For example, the following code can be used to reproduce Figure 1.

```
p1 <- vip(tree) + ggtitle("Single tree")
p2 <- vip(rfo) + ggtitle("Random forest")
p3 <- vip(bst) + ggtitle("Gradient boosting")

# Figure 1
grid.arrange(p1, p2, p3, nrow = 1)
```

Notice how the `vip()` function always returns a `"ggplot"` object (by default, this will be a bar plot). For large models with many features, a dot plot is more effective (in fact, a number of useful plotting options can be fiddles with). Below we call `vip()` and change a few useful options (the resulting plot is displayed in Figure 2).

```
# Load required packages
library(ggplot2)  # for theme_light() function

vip(bst, num_features = 5, bar = FALSE, color, horizontal = FALSE,
    color = "red", shape = 17, size = 4) +
  theme_light()
```
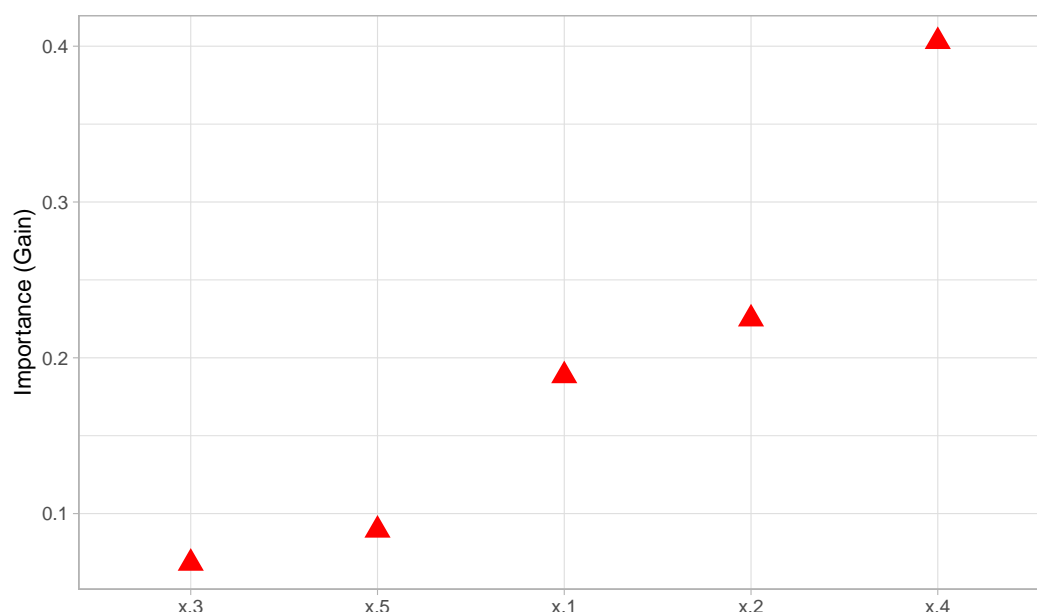

Figure 2: Illustrating various options available in `vip()`.

Linear models

In multiple linear regression, or linear models (LMs), the absolute value of the t-statistic (or some other scaled variant of the estimated coefficients) is commonly used as a measure of VI. The same idea also extends to generalized linear models (GLMs). In the code chunk below, we fit an LM to the simulated `trn` data set allowing for all main and two-way interaction effects, then use the `step()` function to perform backward elimination. The resulting VIP is displayed in Figure 3.

```
# Fit a LM
linmod <- lm(y ~ .^2, data = trn)
backward <- step(linmod, direction = "backward", trace = 0)

# Extract VI scores
vi(backward)
#> # A tibble: 21 x 3
#>    Variable Importance Sign
#>    <chr>         <dbl> <chr>
#> 1 x.4           14.2  POS
#> 2 x.2            7.31 POS
#> 3 x.1            5.63 POS
#> 4 x.5            5.21 POS
#> 5 x.3:x.5        2.46 POS
#> 6 x.1:x.10       2.41 NEG
#> 7 x.2:x.6        2.41 NEG
#> 8 x.1:x.5        2.37 NEG
#> 9 x.10           2.21 POS
```

```
#> 10 x.3:x.4       2.01 NEG
#> #  with 11 more rows

# Plot VI scores
vip(backward, num_features = length(coef(backward)),
    bar = FALSE, horizontal = FALSE)
```
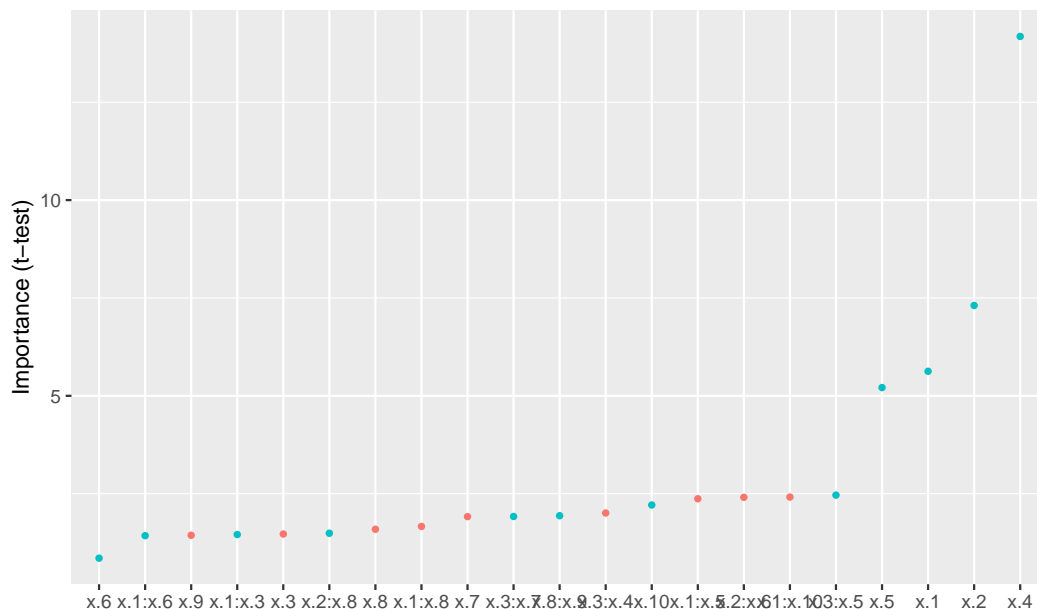

Figure 3: Example VIP from a linear model.

One issue with computing VI scores for LMs using the *t*-statistic approach is that a score is assigned to each term in the model, rather than to just each feature! We can solve this problem using one of the model-agnostic approaches discussed later.

Multivariate adaptive regression splines (MARS), which were introduced in Friedman (1991), is an automatic regression technique which can be seen as a generalization of multiple linear regression and generalized linear models. In the MARS algorithm, the contribution (or VI score) for each predictor is determined using a generalized cross-validation (GCV) statistic. An example using the **earth** package (Milborrow, 2017) is given below (the results are plotted in Figure 4):

```
# Load required packages
library(earth)

# Fit a MARS model
mars <- earth(y ~ ., data = trn, degree = 2, pmethod = "exhaustive")

# Extract VI scores
vi(mars)
#> # A tibble: 10 x 2
#>    Variable Importance
#>    <chr>         <dbl>
#>  1 x.4             100
#>  2 x.1            83.2
#>  3 x.2            83.2
#>  4 x.5            59.3
#>  5 x.3            43.5
#>  6 x.6               0
#>  7 x.7               0
#>  8 x.8               0
#>  9 x.9               0
#> 10 x.10              0
```
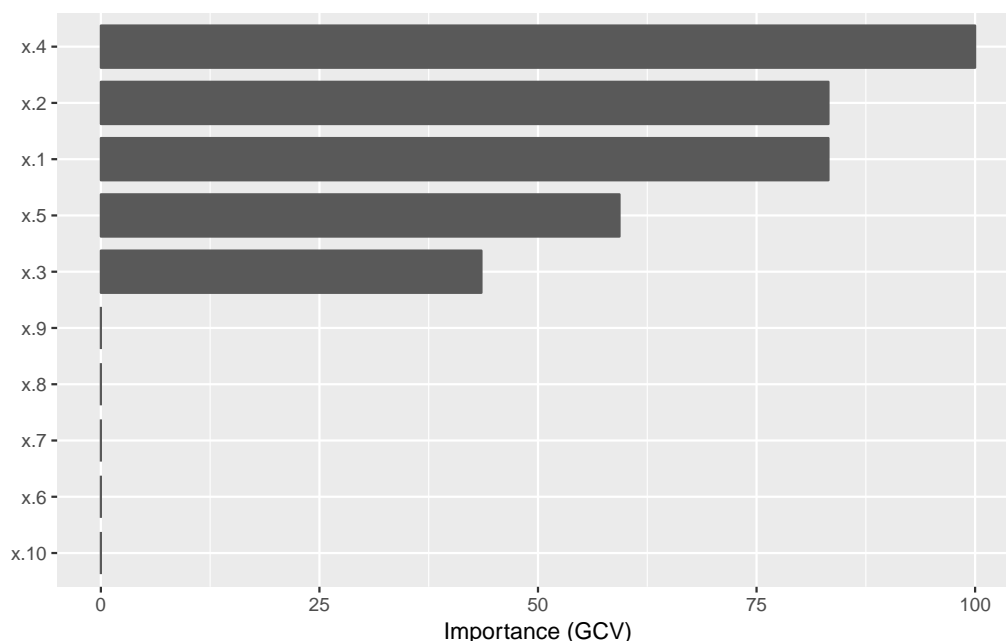
```
# Plot VI scores
vip(mars)
```


Figure 4: Example VIP from a MARS model
.

Neural networks

For neural netwroks (NNs), two popular methods for constructing VI scores are the Garson algorithm (Garson, 1991), later modified by Goh (1995), and the Olden algorithm (Olden et al., 2004). For both algorithms, the basis of these importance scores is the networks connection weights. The Garson algorithm determines VI by identifying all weighted connections between the nodes of interest. Oldens algorithm, on the other hand, uses the product of the raw connection weights between each input and output neuron and sums the product across all hidden neurons. This has been shown to outperform the Garson method in various simulations. For DNNs, a similar method due to Gedeon (1997) considers the weights connecting the input features to the first two hidden layers (for simplicity and speed); but this method can be slow for large networks. We illustrate these two methods below using vip() with the **nnet** package (Venables and Ripley, 2002) (see the results in Figure **??**).

   FIXME: Add vip support for these two methods!

```
# Load required packages
library(nnet)

# Fit a neural network
set.seed(0803)
nn <- nnet(y ~ ., data = trn, size = 7, decay = 0.1, linout = TRUE)

# VIPs
p1 <- vip(nn, method = "garson")
p2 <- vip(nn, method = "olsen")

# Figure X
grid.arrange(p1, p2, nrow = 1)
```

Model-agnostic VI

Model-agnostic interpredibility separates interpretation from the model. Compared to model-specific approaches, model-agnostic VI methods are more flexible (since they can be applied to any supervised learning algorithm). In this section, we discuss model-agnostic methods for quantifying global

feature importance using three different approaches: 1) *partial dependence plots* (PDPs), 2) *individual conditional expectation* (ICE) curves, and 3) permutation-based feature importance. For details on approaches 1)–2), see Greenwell et al. (2018).

Our first model-agnostic approach is based on quantifying the "flatness" of the PDPs of each feature. PDPs help visualize the effect of low cardinality subsets of the feature space on the estimated prediction surface (e.g., main effects and two/three-way interaction effects.). PDPs provide model-agnostic interpretations and can be constructed in the same way for any supervised learning algorithm; for an overview, see Greenwell (2017). Below, we fit a projection pursuit regression (PPR) model and construct PDPs for each feature using the **pdp** package Greenwell (2017). The results are displayed in Figure 5. Notice how the PDPs for the uninformative features are relatively flat compared to the PDPs for features x.1–x.2!

```
# Load required packages
library(pdp)

# Fit a PPR model (nterms was chosen using the caret package with 5 repeats of
# 5-fold cross-validation)
pp <- ppr(y ~ ., data = trn, nterms = 11)

# PDPs for all 10 features
features <- paste0("x.", 1:10)
pdps <- lapply(features, FUN = function(feature) {
  pd <- partial(pp, pred.var = feature)
  autoplot(pd) +
    ylim(range(trn$y)) +
    theme_light()
})
grid.arrange(grobs = pdps, ncol = 5)
```
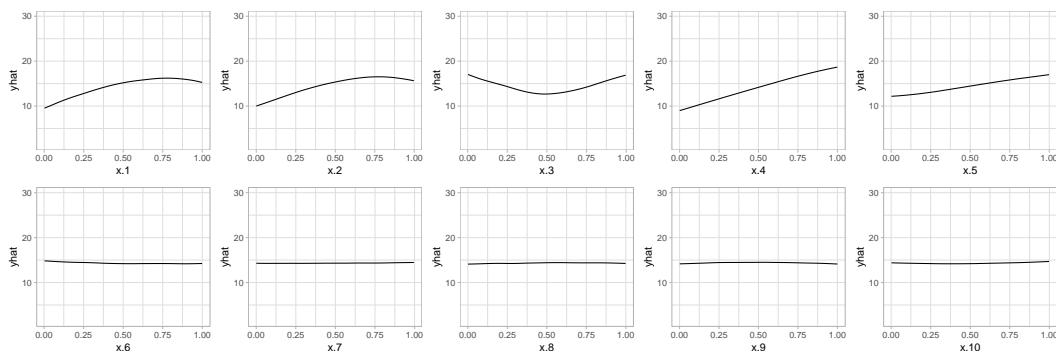

Figure 5: PDPs of main effects in the PPR model.

Next, we compute PDP-based VI scores for the PPR and NN models. The PDP method constructs VI scores that quantify the "flatness" of each PDP (by default, this is defined by computing the standard deviation of the y-axis values for each PDP). To use the PDP method, specify method = "pdp" in the call to vi() or vip():

```
# Fit a PPR model (nterms was chosen using the caret package with 5 repeats of
# 5-fold cross-validation)
pp <- ppr(y ~ ., data = trn, nterms = 11)

# Plot VI scores
p1 <- vip(pp, method = "pdp") + ggtitle("PPR")
p2 <- vip(nn, method = "pdp") + ggtitle("NN")

# Figure X
grid.arrange(p1, p2, ncol = 2)
```

In Figure **??** we display the PDP-based feature importance for the previously obtained PPR and NN models. These VI scores essentially capture the variability in the partial dependence values for each main effect.
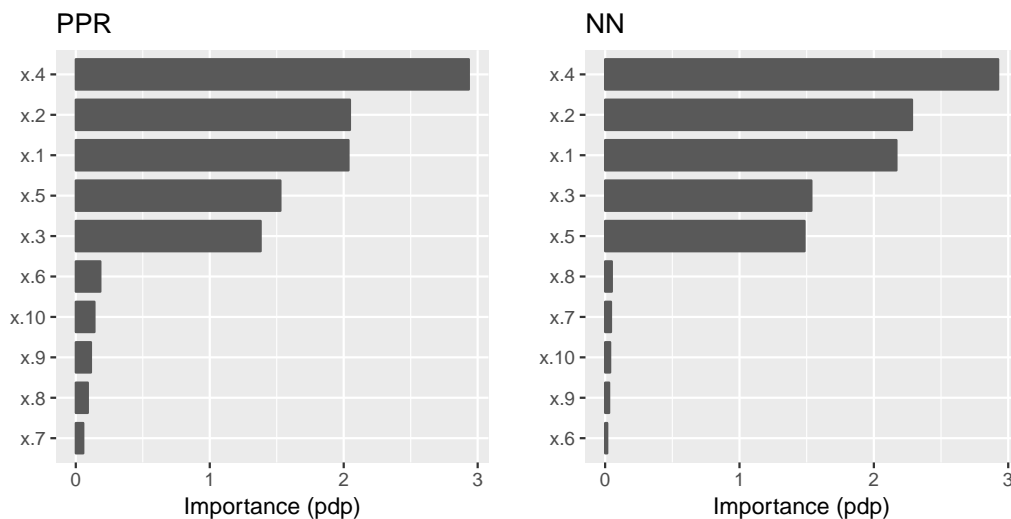
Figure 6: PDP-based feature importance for the PPR and NN models.

ICE curve method

The ICE curve method is similar to the PDP method. The only difference is that we measure the "flatness" of each ICE curve and then aggregate the results (e.g., by averaging). If there are no (substantial) interaction effects, using `method = "ice"` will produce results similar to using `method = "pdp"`. However, if strong interaction effects are present, they can obfuscate the main effects and render the PDP-based approach less useful (since the PDPs for important features can be relatively flat when certain interactions are present; see Goldstein et al. (2015) for details). In fact, it is probably safest to always use `method = "ice"`.

Below, we display the ICE curves for each feature in the PP model using the same y-axis scale; see Figure 7. Again, there is a clear difference between the ICE curves for features x.1–x.5 and x.6–x.10; the later being relatively flat by comparison. Also, notice how the ICE curves within each feature are relatively parallel (if the ICE curves within each feature were perfectly parallel, the standard deviation for each curve would be the same and the results will be identical to the PDP method). In this example, the interaction term between x.1 and x.2 does not obfuscate the PDPs for the main effects and the results are not much different.

```
# ICE curves for all 10 features
ice_curves <- lapply(features, FUN = function(feature) {
  ice <- partial(pp, pred.var = feature, ice = TRUE)
  autoplot(ice, alpha = 0.1) +
    ylim(range(trn$y)) +
    theme_light()
})

# Figure X
grid.arrange(grobs = ice_curves, ncol = 5)
```

Obtaining the ICE-based feature importance scores is also straightforward. The results in Figure 8 are similar to those obtained using the PDP method.

```
# Plot VI scores
p1 <- vip(pp, method = "ice") + ggtitle("PPR")
p2 <- vip(nn, method = "ice") + ggtitle("NN")

# Figure X
grid.arrange(p1, p2, ncol = 2)
```

Permutation method

The permutation method exists in various forms and was made popular in Breiman (2001) for random forests. The permutation approach used in **vip** is quite simple. The idea is that if we randomly
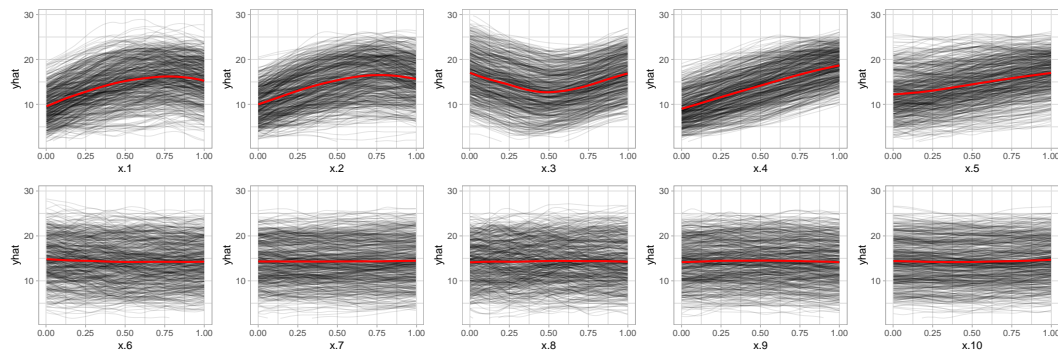
Figure 7: ICE curves for each feature in the PPR model. The red curve represents the PDP (i.e., the averaged ICE curves).
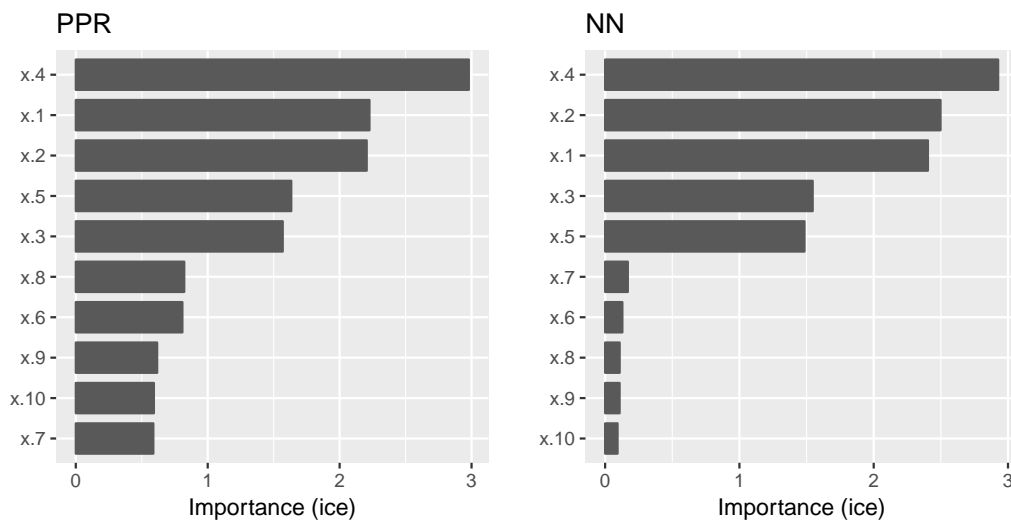

Figure 8: ICE-based feature importance for the PPR and NN models.

permute thse values of an important feature in the training data, the training performance would degrade (since permuting the values of a feature effectively destroys any relationship between that feature and the target variable). This of course assumes that the model has been properly tuned (e.g., using cross-validation) and is not over fitting. The permutation approach uses the difference between some baseline performance measure (e.g., training R^2, AUC, or RMSE) and the same performance measure obtained after permuting the values of a particular feature in the training data (**Note:** the model is NOT refit to the training data after randomly permuting the values of a feature). To use the permutation approach, specify `method = "permute"` in the call to `vi()` or `vip()`. Note that using `method = "permute"` requires specifying a few additional arguments; see `?vi_permute` for details.

An example is given below for the previously fitted PPR and NN models. The result, which are displayed in Figure 9, agree with those obtained using the PDP- and ICE-based methods.

```
# Plot VI scores
set.seed(2021)  # for reproducibility
p1 <- vip(pp, method = "permute", target = "y", metric = "rsquared",
        pred_fun = predict) + ggtitle("PPR")
p2 <- vip(nn, method = "permute", target = "y", metric = "rsquared",
        pred_fun = predict) + ggtitle("NN")

# Figure X
grid.arrange(p1, p2, ncol = 2)
```

Tne permutation approach introduces randomness into the procedure and therefore should be run more than once if computationally feasible. Fortunately, this also allows us tom compute standard errors for the estimated eature importance, as illustaretd in the example below.
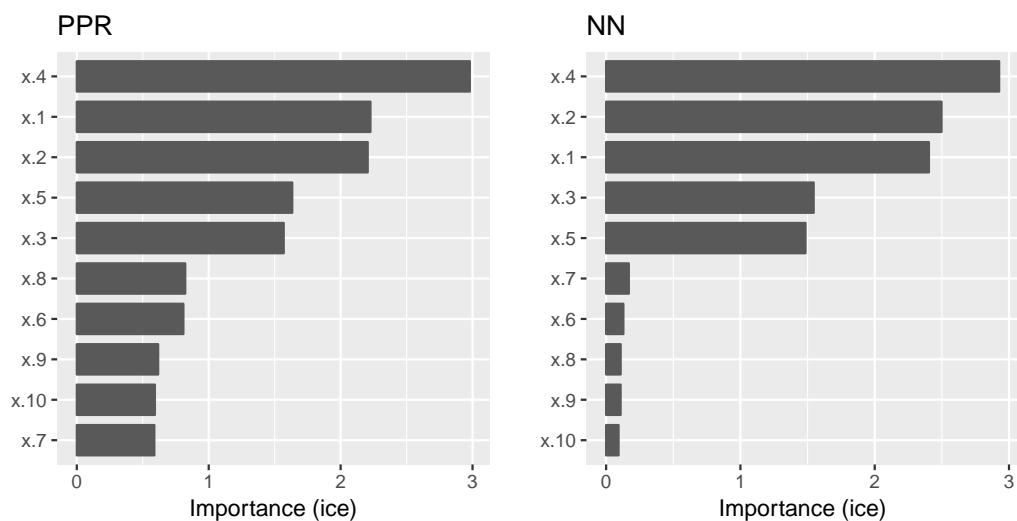
```
# A tibble: 10 x 3
```

Figure 9: Permutation-based feature importance for the PPR and NN models.

```
#>    Variable Importance   StDev
#>    <chr>         <dbl>   <dbl>
#>  1 x.4           0.586  0.0188
#>  2 x.2           0.416  0.0184
#>  3 x.1           0.377  0.0162
#>  4 x.5           0.222  0.0155
#>  5 x.3           0.183  0.00936
#>  6 x.6           0.0620 0.00379
#>  7 x.8           0.0603 0.00378
#>  8 x.9           0.0389 0.00417
#>  9 x.7           0.0373 0.00321
#> 10 x.10          0.0348 0.00290
```

A TensorFlow example

TBD.

Summary

TBD.

Bibliography

L. Breiman. Bagging predictors. *Machine Learning*, 8(2):209–218, 1996. URL https://doi.org/10.1023/A:1018054314350. [p1]

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. URL https://doi.org/10.1023/A:1010933404324. [p8]

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. URL https://doi.org/10.1214/aos/1176347963. [p1, 5]

D. G. Garson. Interpreting neural-network connection weights. *Artificial Intelligence Expert*, 6(4):46–51, 1991. [p6]

T. Gedeon. Data mining of inputs: Analysing magnitude and functional measures. *International Journal of Neural Systems*, 24(2):123–140, 1997. URL https://doi.org/10.1007/s10994-006-6226-1. [p6]

A. Goh. Back-propagation neural networks for modeling complex systems. *Artificial Intelligence in Engineering*, 9(3):143–151, 1995. doi: http://dx.doi.org/10.1016/0954-1810(94)00011-S. [p6]

A. Goldstein, A. Kapelner, J. Bleich, and E. 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. URL https://doi.org/10.1080/10618600.2014.907095. [p8]

B. M. Greenwell. pdp: An r package for constructing partial dependence plots. *The R Journal*, 9(1):421–436, 2017. URL https://journal.r-project.org/archive/2017/RJ-2017-016/index.html. [p7]

B. M. Greenwell, B. C. Boehmke, and A. J. McCarthy. A simple and effective model-based variable importance measure. *arXiv preprint arXiv:1805.04755*, 2018. [p7]

F. Leisch and E. Dimitriadou. *mlbench: Machine Learning Benchmark Problems*, 2012. URL https://CRAN.R-project.org/package=mlbench. R package version 2.1-1. [p1]

S. Milborrow. *earth: Multivariate Adaptive Regression Splines*, 2017. URL https://CRAN.R-project.org/package=earth. R package version 4.5.0. [p5]

J. D. Olden, M. K. Joy, and R. G. Death. An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data. *Ecological Modelling*, 178(3):389–397, 2004. doi: http://dx.doi.org/10.1016/j.ecolmodel.2004.03.013. [p6]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, New York, fourth edition, 2002. URL http://www.stats.ox.ac.uk/pub/MASS4. ISBN 0-387-95457-0. [p6]

Brandon M. Greenwell
Department of Mathematics and Statistics
Wright State University
3640 Colonel Glenn Hwy
Dayton, OH 45435
United States of America
ORCiD: 0000-0002-8120-0084
greenwell.brandon@gmail.com

Bradley C. Boehmke
University of Cincinnati
2925 Campus Green Dr
Cincinnati, OH 45221
United States of America
ORCiD: 0000-0002-3611-8516
bradleyboehmke@gmail.com