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.

TODO:

- Discuss strengths/weaknesses of different approaches.
- Discuss how PDP and ICE curve info is stored as an attribute whenever `method = "pdp"` or `method = "ice"`, respectively.
- Implement and discuss ratio approach in permutation-based VI scores in **vip**?
- Should we discuss interaction detection? Probably...(the H-statistic example, if correct, is motivation enough).
- ~~Discuss other packages (like **DALEX** and **iml**).~~
- ~~Need a good motivating example (e.g., airfoil example from arXiv paper using the **SuperLearner** package).~~
- ~~Better discussion of function arguments, especially for permutation approach.~~
- ~~Parallel examples for PDP-, ICE-, and permutation-based approaches.~~
- ~~Example showing how to supply your own custom metric function.~~
- ~~Add note about permuting features in the train vs. test/new data and a reference to the IML book chapter where this is discussed further.~~
- ~~Discuss layout of the package (e.g., `vi()` vs. `vip()` and workhorse functions like `vi_permute()`.~~

Introduction

Too often machine learning (ML) models are summarized using a single metric (e.g., cross-validated accuracy) and then put into production. Although we often care about the predictions from these models, it is becoming routine to also understand the predictions! Understanding how an ML model makes its predictions helps build trust in the model. Understanding ML models and their predictions is the fundamental idea of the emerging field of *interpretable machine learning* (IML). For an in-depth discussion on IML, see Molnar (2019). In this paper, we focus on global methods for quantifying the importance of features in an ML model. Computing variable importance (VI) and communicating them through variable importance plots (VIPs) is a fundamental component of IML and is the main topic of this paper.

VI scores and VIPs can be constructed for genberal ML models using a number of packages. The **iml** package (Molnar et al., 2018) provides the `FeatureImp()` function which computes feature importance for general prediction models using the permutation approach (discussed later). It is written in **R6** (Chang, 2019) and allows the user to specify a generic loss function or select from a pre-defined list (e.g., `loss = "mse"` for mean squared error). It also allows the user to specify whether importance is measures ad the difference or as the ratio of the original model error and the model error after permutation. The user can also specify the number of repetitions used when permuting each feature to help stabilize the variability in the procedure.

The **DALEX** package (Biecek, 2018) also provides permutation-based variable importance scores through the `variable_importance()` function. Similar to `iml::FeatureImp()`, this function allows the user to specify a loss function and how the importance scores are computed (e.g., using the difference or ratio). It also provides an option to sample the training data before shuffling the data to compute importance (the default is to use `n_sample = 1000`. This can help speed up computation.

The **caret** package (Kuhn, 2017b) includes a general `varImp()` function for computing model-specific and filter-based variable importance scores. Filter-based approaches, which are described in Kuhn and Johnson (2013), do not make use of the fitted model to measure VI. They also do not take into account the other predictors in the model. For regression problems, a popular approach to measuring the VI of a numeric predictor x is to first fit a flexible nonparametric model between x and the target Y; for example, the locally-weighted polynomial regression (LOWESS) method developed by Cleveland (1979). From this fit, a pseudo-R^2 measure can be obtained from the resulting residuals and used as a measure of VI. For categorical predictors, a different method based on standard statistical tests (e.g., t-tests and ANOVAs) can be employed; see Kuhn and Johnson (2013) for details. For classification problems, an area under the ROC curve (AUC) statistic can be used to quantify predictor importance. The AUC statistic is computed by using the predictor x as input to the ROC curve. If x can reasonably separate the classes of Y, that is a clear indicator that x is an important predictor (in terms of class separation) and this is captured in the corresponding AUC statistic. For problems with more than two classes, extensions of the ROC curve or a one-vs-all approach can be used.

If you use the **mlr** interface for fitting ML models (Bischl et al., 2016), then you can use the `getFeatureImportance()` function to extract model-specific VI scores from various tree-based models (e.g., boosted trees and random forests). Unlike **caret**, the model needs to be fit via the **mlr** interface; for instance, you cannot use `getFeatureImportance()` a **gbm** model (Ridgeway, 2017) unless it ws fit using **mlr**.

While the **iml** and **DALEX** packages provide model-agnostic approaches to computing variable importance, **caret** provides model-specific approaches (e.g., using the absolute value of the t-statistic for linear models) as well less accurate filter-based approaches . Furthermore, each package has a completely different interface (e.g., **iml** is written in R6). The **vip** package (Greenwell and Boehmke, 2018) strives to provide a consistent interface to both model-specific and model-agnostic approaches to feature importance that is simple to use with three main functions:

- `vi()` computes VI scores using model-specific or model-agnostic approaches (the results are always returned as a tibble (Müller and Wickham, 2019));
- `vip()` computes VI scores using model-specific or model-agnostic approaches and allow flexible plotting of the results with **ggplot2**-type graphics (Wickham, 2016);
- `add_sparklines()` adds a novel sparkline representation of feature effects (e.g., partial dependence plots) to any VI table produced by `vi()`.

There's also a function called `vint()` (for variable interactions) but is is experimental and will not be discussed here. The interested reader is pointed to Greenwell et al. (2018). Note that `vi()` is actually a wrapper around four main workhorse functions: `vi_model()`, `vi_ice()`, `vi_pdp()`, and `vi_permute()`. These functions compute various types of VI scores. The workhorse function that actually gets called is controlled by the `method` argument in `vi()`; the default is `method = "model"` which corresponds to model-specific variable importance (see `?vi` for details and links to further documentation).

Constructing VIPs in R

We'll illustrate major concepts using one of the regression problems described in Friedman (1991) and Breiman (1996):

$$Y_i = 10\sin\left(\pi X_{1i} X_{2i}\right) + 20\left(X_{3i} - 0.5\right)^2 + 10X_{4i} + 5X_{5i} + \epsilon_i, \quad i = 1, 2, \dots, n, \tag{1}$$

where $\epsilon_i \overset{iid}{\sim} N\left(0, \sigma^2\right)$. Data from this model can be generated using the `mlbench.friedman1()` function 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
```

For these data, it should be clear that features X_1–X_5 are the most important! (The others don't influence Y at all.) Also, based on the form of the model, we'd expect X_4 to be the most important feature, probably followed by X_1 and X_2, both comparably important, with X_5 probably less important. The influence of x_3 is harder to determine due to its quadratic nature, but it seems likely that this nonlinearity will suppress this variable's influence since the total range of this term is from 0–1, the same as the X_5 term.

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 t, 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 i^t. 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)
```

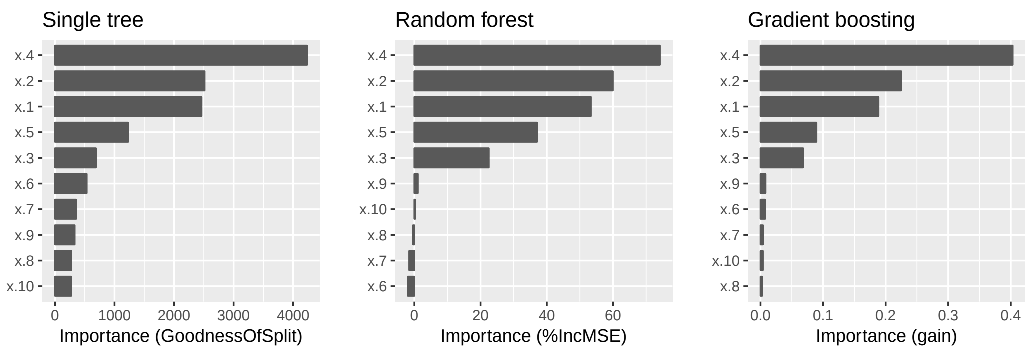


Figure 1: Model-based VIPs for the three different tree-based models fit to the simulated Friedman data..

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 one place 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()
```

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
```

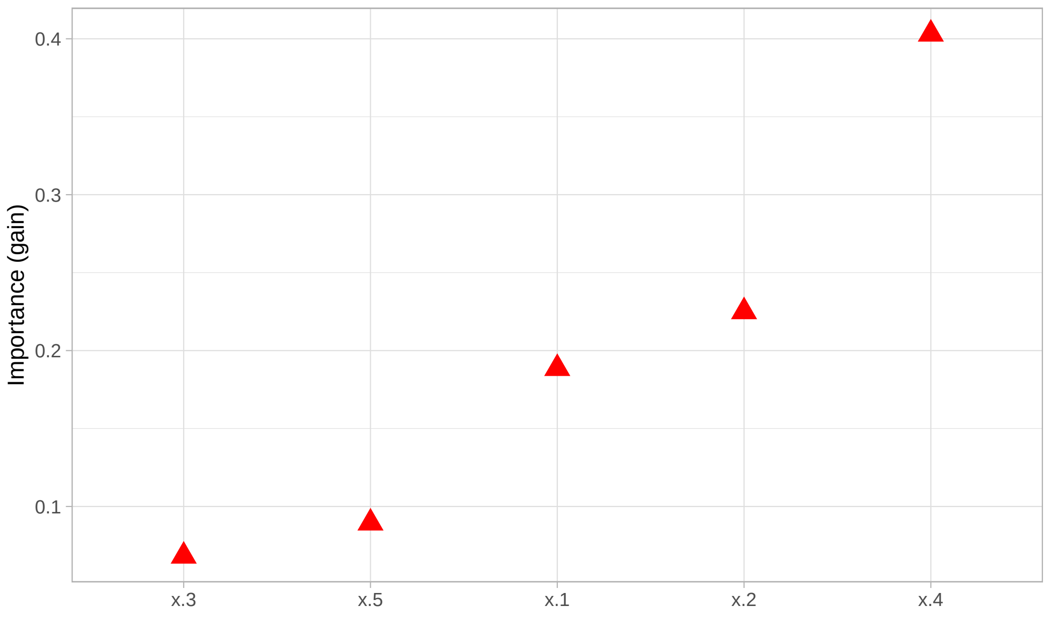


Figure 2: Illustrating various options available in `vip()`.

```
#>    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)
```

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 each individual 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 5):

```
# 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
```

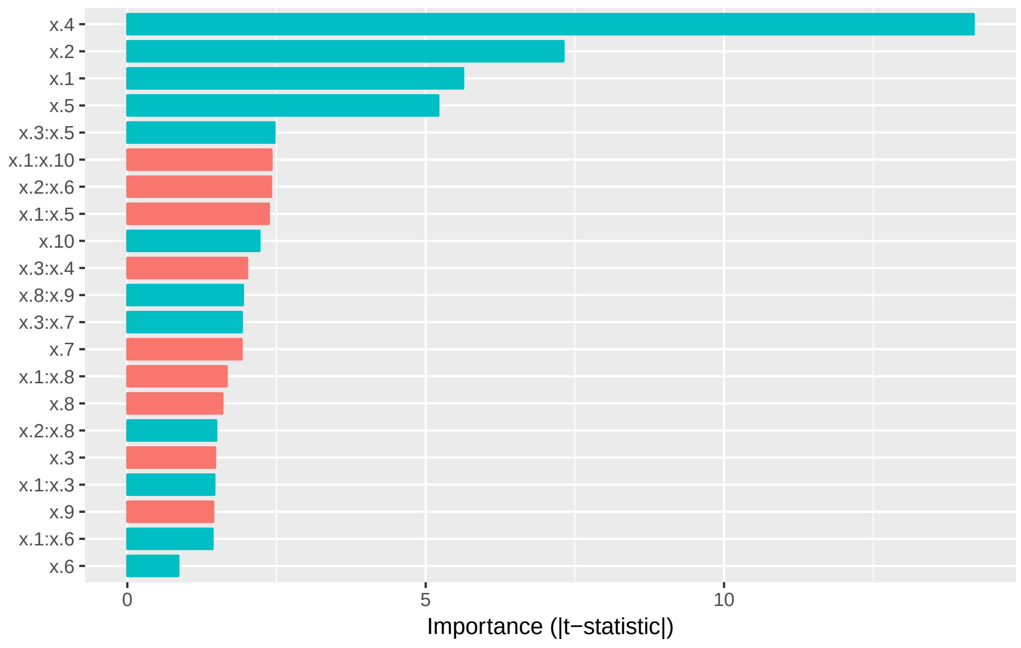


Figure 3: Example VIP from a linear model fit to the simulated Friedman data.

```
#>  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)
```

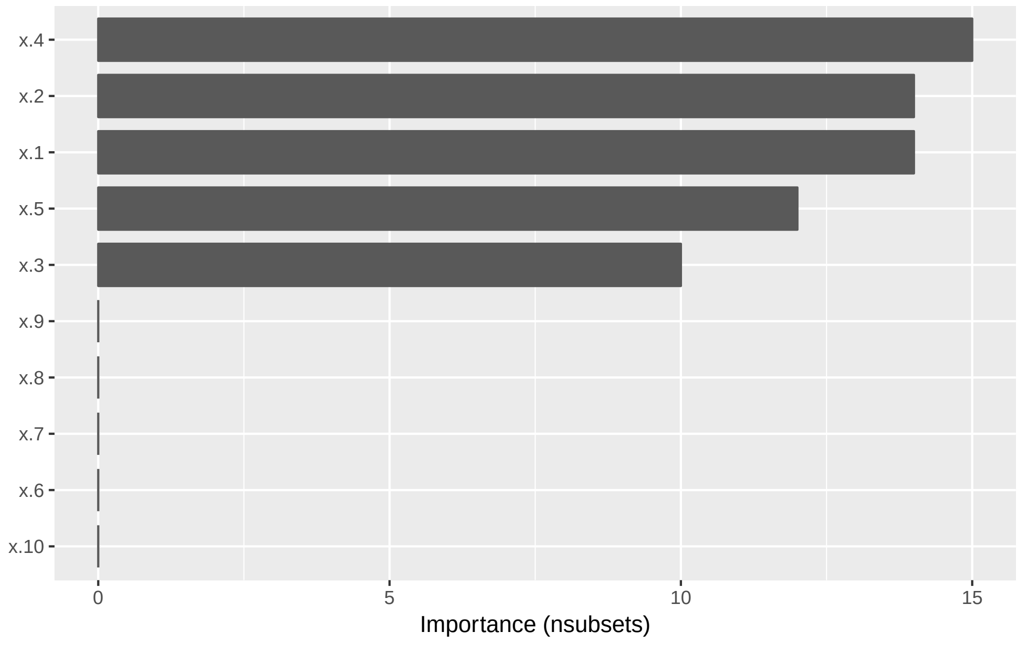


Figure 4: Example VIP from a MARS model fit to the simulated Friedman data.

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 **??**).

```
# 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, type = "garson")
p2 <- vip(nn, type = "olden")

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

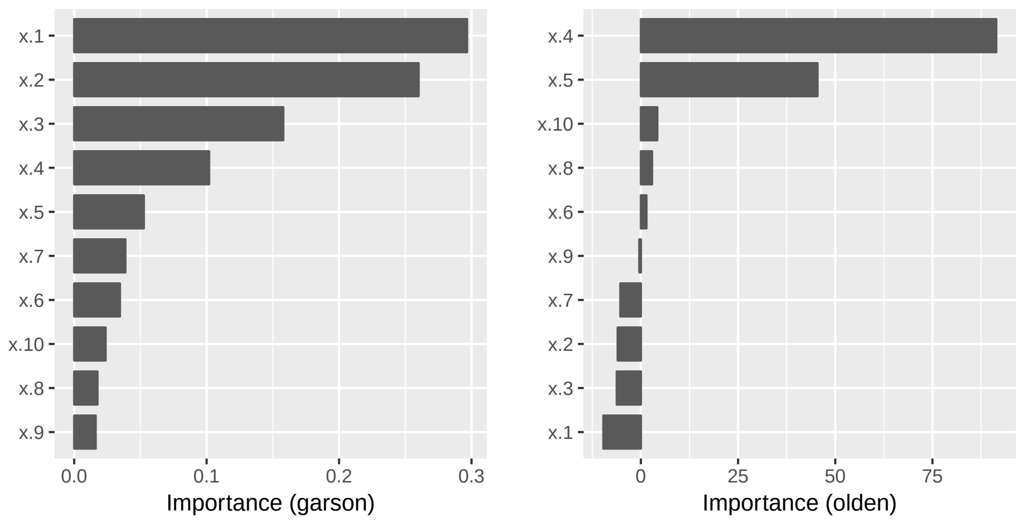


Figure 5: Example VIPs from a single-hidden-layer NN fit to the simulated Friedman data.

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).

PDP method

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 6. 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)
```

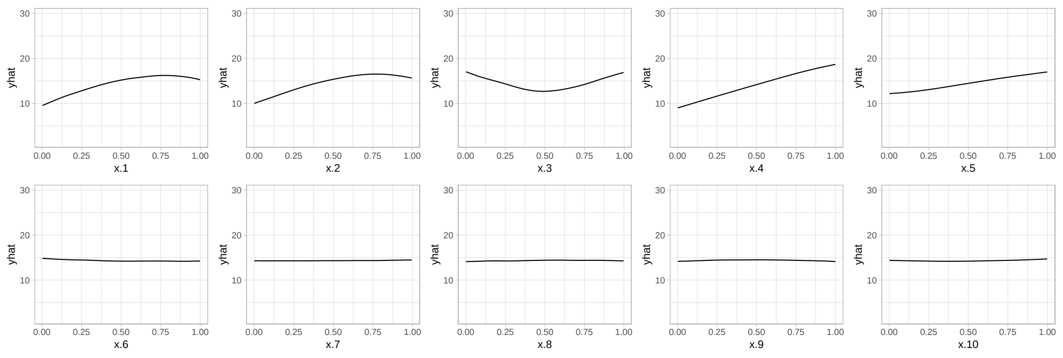


Figure 6: PDPs of main effects in the PPR model fit to the simulated Friedman data..

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.

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

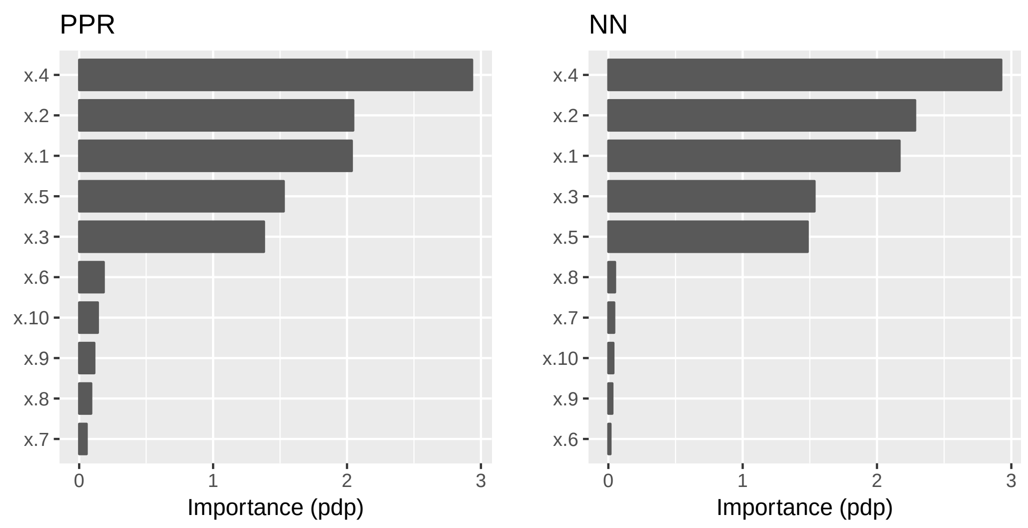


Figure 7: PDP-based feature importance for the PPR and NN models fit to the simulated Friedman data..

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 8. 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)
```

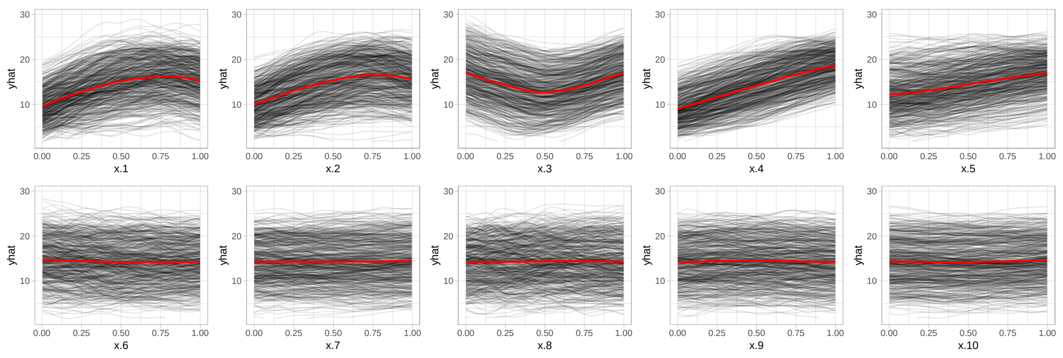


Figure 8: ICE curves for each feature in the PPR model fit to the simulated Friedman data. The red curve represents the PDP (i.e., the averaged ICE curves).

Obtaining the ICE-based feature importance scores is also straightforward. The results in Figure 9 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)
```

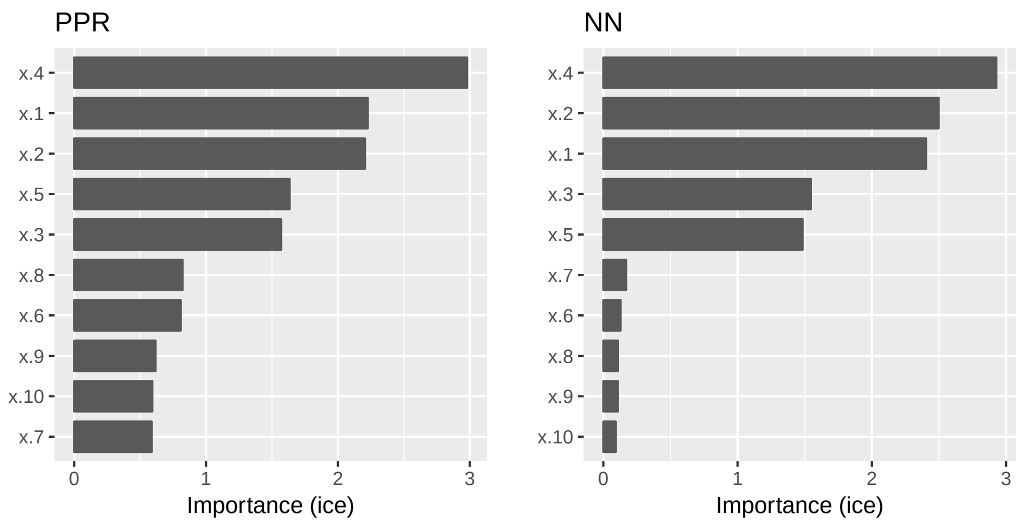


Figure 9: ICE-based feature importance for the PPR and NN models fit to the simulated Friedman data..

Permutation method

The permutation method exists in various forms and was made popular in Breiman (2001) for random forests and generalized and extended in Fisher and Dominici (2018). The permutation approach used in **vip** is quite simple. The idea is that if we randomly permute these 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). It is also important to note that this method may not be appropriate when you havem, for example, highly correlated features (since permuting one feature at a time may lead to unlikely data instances). To use the permutation approach in **vip**, 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 10, 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_wrapper = predict) + ggtitle("PPR")
p2 <- vip(nn, method = "permute", target = "y", metric = "rsquared",
          pred_wrapper = predict) + ggtitle("NN")

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

The permutation approach introduces randomness into the procedure and therefore should be run more than once if computationally feasible. Fortunately, this also allows us to compute standard errors for the estimated feature importance, as illustrated in the example below where we specify `nsim = 10` to request that each feature be permuted 10 times and the results averaged together.

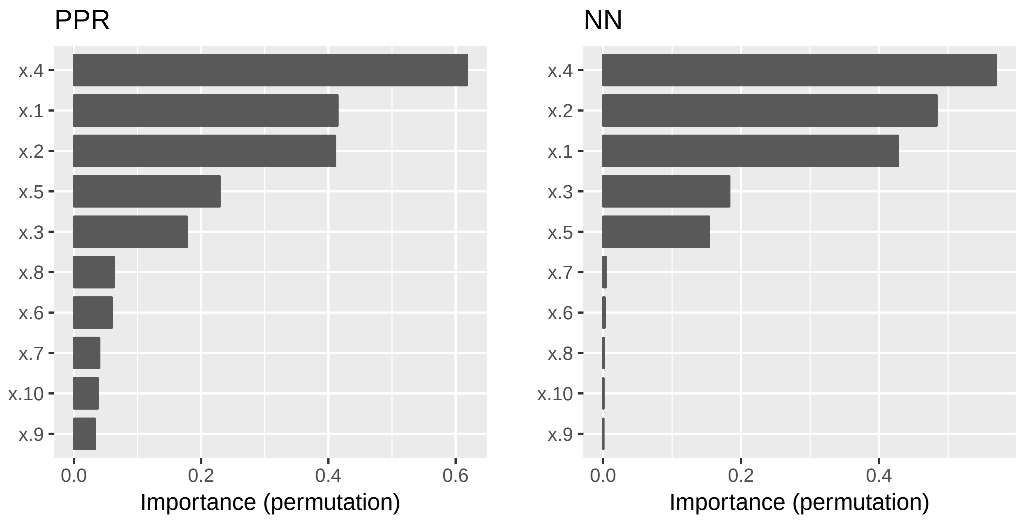


Figure 10: Permutation-based feature importance for the PPR and NN models fit to the simulated Friedman data.

```
# A tibble: 10 x 3
#>    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
```

All available metrics for regression and classification can be listed using the `list_metrics()` function, for example:

```
#>     Metric                      Description                      Task
#> 1      auc           Area under (ROC) curve     Binary classification
#> 2    error          Misclassification error     Binary classification
#> 3  logloss                         Log loss     Binary classification
#> 4     mauc Multiclass area under (ROC) curve Multiclass classification
#> 5 mlogloss               Multiclass log loss Multiclass classification
#> 6      mse               Mean squared error                Regression
#> 7       r2                        R squared                Regression
#> 8 rsquared                        R squared                Regression
#> 9     rmse          Root mean squared error                Regression
```

We can also use a custom metric (i.e., loss function). Suppose for example you want to measure importance using the *mean absolute error* (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|Y_i - \widehat{f}\left(\boldsymbol{X}_i\right)\right|, \tag{2}$$

where $\widehat{f}\left(\boldsymbol{X}_i\right)$ is the predicted value of Y_i. A simple function implementing this metric is given below (note that, according to the documentation in `?vi_permute()`, this function requires two arguments: `actual` and `predicted`).

```
mae <- function(actual, predicted) {
  mean(abs(actual - predicted))
}
```

To use this for computing permutation-based VI scores just pass it to the `metric` argument (be warned, however, that the metric used for computing permutation importance should be the same as the metric used to train and tune the model). Also, since this is a custom metric, we need to specify whether a smaller value indicates better performance by setting `smaller_is_better = TRUE`. The results, which are displayed in Figure 12, are similar to those in Figure 10, albeit a different scale.

```
# Figure X
set.seed(2321)  # for reproducibility
vip(nn, method = "permute", target = "y", metric = mae,
    smaller_is_better = TRUE, pred_wrapper = nnet:::predict.nnet) +
  ggtitle("Custom loss function: MAE")
```

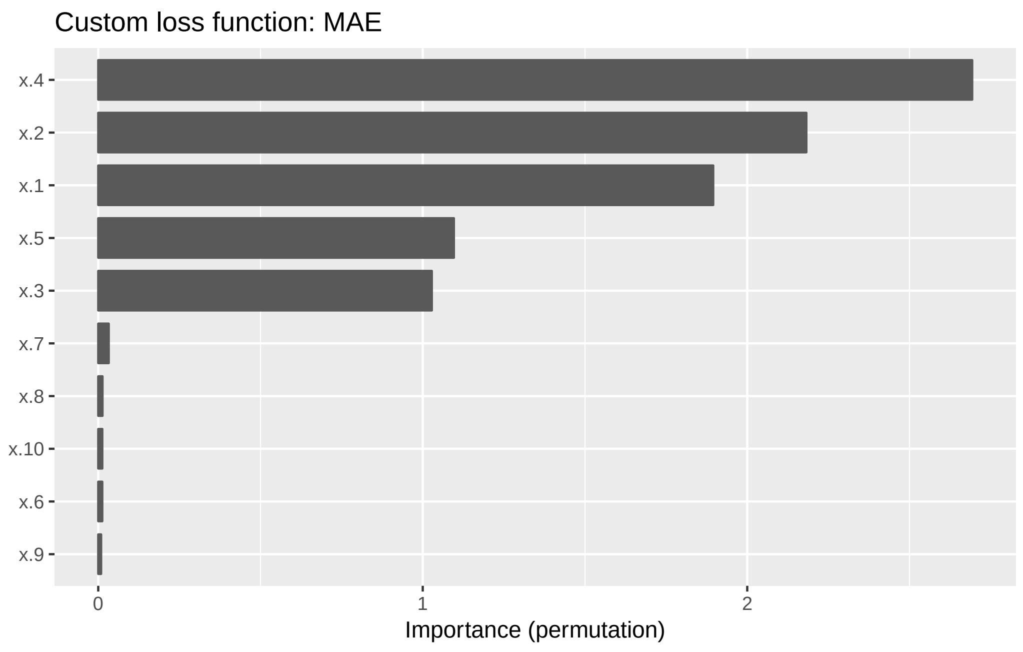


Figure 11: Permutation-based feature importance for the NN model fit to the simulated Friedman data. In this example, permutation importance is based on the MAE metric.

Although permutation importance is most naturally computed on the training data, it may also be useful to do the shuffling and measure performance on new data! This is discussed in depth in Molnar (2019)[Chapter 5, Section 5.2]. For users interested in computing permutation importance using new data, just supply it to the `train` argument in the call to `vi()` or `vi_permute()`. For instance, suppose we wanted to only use a fraction of the original training data to carry out the computations. In this case, we could simply pass the sampled data to the `train` argument as follows:

```
# Figure X
set.seed(2327)  # for reproducibility
vip(nn, method = "permute",
    train = trn[sample(nrow(trn), size = 400), ],  # sample 400 observations
    target = "y", metric = "rmse") +
  ggtitle("Using a random subset of training data")
```

Use sparklines to characterize feature effects

Starting with **vip** 0.1.3, we have included a new function `add_sparklines()` for constructing HTML-based feature importance tables; this feature requires the **DT** package (Xie et al., 2018). The primary difference between `vi()` and `add_sparklines()` is that the latter includes an `Effect` column that displays a sparkline representation of the partial dependence function for each feature. This is a concise way to display both feature importance and feature effect information in a single (interactive) table. See `?vip::add_sparklines` for details. We illustrate the basic use of `add_sparklines()` in the code chunks below.

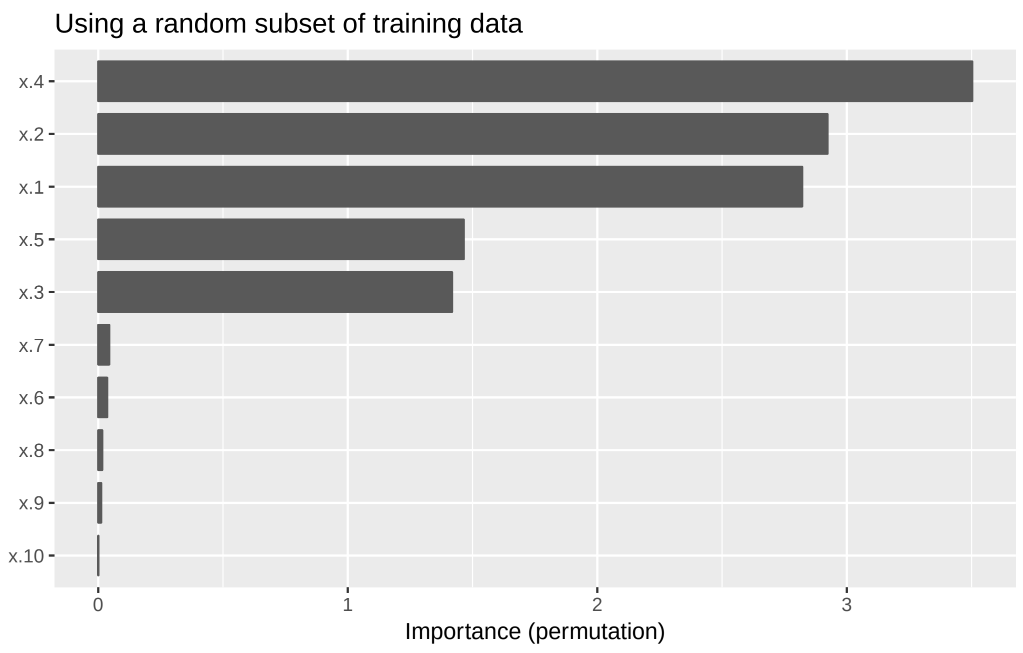


Figure 12: Permutation-based feature importance for the NN model fit to the simulated Friedman data. In this example, permutation importance is based on a random sample of 400 training observations.

```
# First, compute a tibble of variable importance scores using any method
var_imp <- vi(rfo, method = "permute", metric = "rmse", target = "y")

# Next, convert to an html-based data table with sparklines
add_sparklines(var_imp, fit = rfo)
```

Predict the sale price for a home

For illustration, we'll use the Ames housing data set—a more contemporary version of the often cited Boston Housing data set. These data are available in the **AmesHousing** package (Kuhn, 2017a). These data describe the sale of individual residential properties in Ames, Iowa from 2006–2010. The data set contains 2930 observations, 80 features (23 nominal, 23 ordinal, 14 discrete, and 20 continuous), and a continuous target giving the sale price of the home. The version we'll load is a cleaned up version of the original data set and treats all categorical variables as nominal (see `?AmesHousing::make_ames` for details).

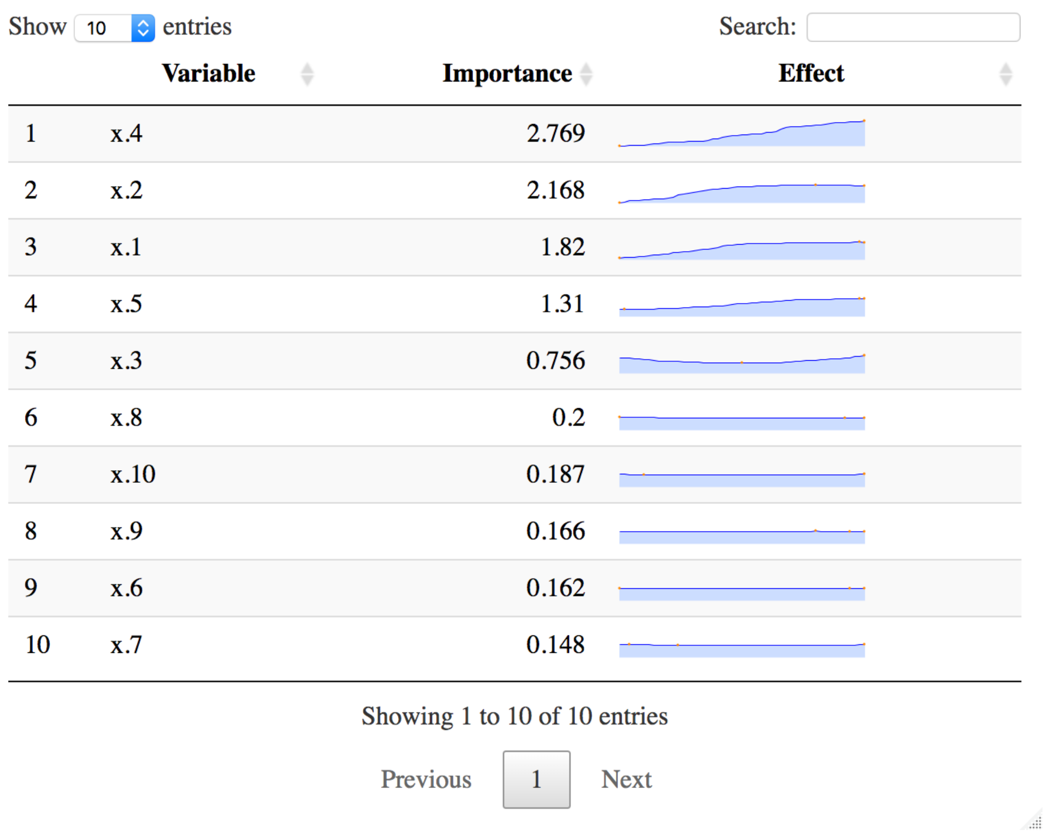
Using the R package **SuperLearner** (Polley et al., 2018), we trained five models using 5-fold cross-validation: a GBM using the **xgboost** package (Chen et al., 2017), a random forest using the **ranger** package (Wright, 2017), a MARS model using the **earth** (Milborrow, 2017) package, a GLMNET model using the **glmnet** package (Friedman et al., 2010), and a support vector regression model using the **kernlab** package (Karatzoglou et al., 2004). The coefficients from the meta learner indicate that the GBM contributes the most to new predictions, followed by the MARS model, random forest, and the GLMNET. The support vector regression model do not contribute to the ensemble.

```
# Load the Ames housing data
ames <- AmesHousing::make_ames()
X <- subset(ames, select = -Sale_Price)
y <- ames$Sale_Price

# Load required packages
library(SuperLearner)

# List of base learners
learners <- c("SL.xgboost", "SL.ranger", "SL.earth", "SL.glmnet", "SL.ksvm")

# Stack models
```



	Variable	Importance	Effect
1	x.4	2.769	
2	x.2	2.168	
3	x.1	1.82	
4	x.5	1.31	
5	x.3	0.756	
6	x.8	0.2	
7	x.10	0.187	
8	x.9	0.166	
9	x.6	0.162	
10	x.7	0.148	

Showing 1 to 10 of 10 entries

Figure 13: VIP with sparkline representation of feature effects fit to the simulated Friedman data.

```
set.seed(840)
ctrl <- SuperLearner.CV.control(V = 5L, shuffle = TRUE)
sl <- SuperLearner(Y = y, X = X, SL.library = learners, verbose = TRUE,
                   cvControl = ctrl)
sl
#>  Call:
#> SuperLearner(Y = y, X = X, SL.library = learners, verbose = TRUE,
#>     cvControl = ctrl)
#>
#>
#>                         Risk        Coef
#> SL.xgboost_All  527111540 0.603160823
#> SL.ranger_All   658707534 0.111650583
#> SL.earth_All    731657510 0.276895806
#> SL.glmnet_All   884821443 0.008292788
#> SL.ksvm_All    6782709110 0.000000000
```

In the code chunks below we request permutation-based VI scores and add sparkline representations of the PDPs for the top ten features. For this we need to define a couple of wrapper functions. One for computing predictions (for the permutation VI scores) and one for computing averaged predictions (for the PDPs):

```
# Prediction wrapper functions
imp_fun <- function(object, newdata) {  # for permutation-based VI scores
  predict(object, newdata = newdata)$pred
}
par_fun <- function(object, newdata) {  # for PDPs
  mean(predict(object, newdata = newdata)$pred)
}
```

To speed up the process, we perform the computations in parallel by setting `parallel = TRUE` in the call to `vi()` and `add_sparklines()`. Note that we first need to set up a parallel backend for this to work. **vip** uses **plyr** (Wickham, 2011) (which relies on **foreach** (Revolution Analytics and

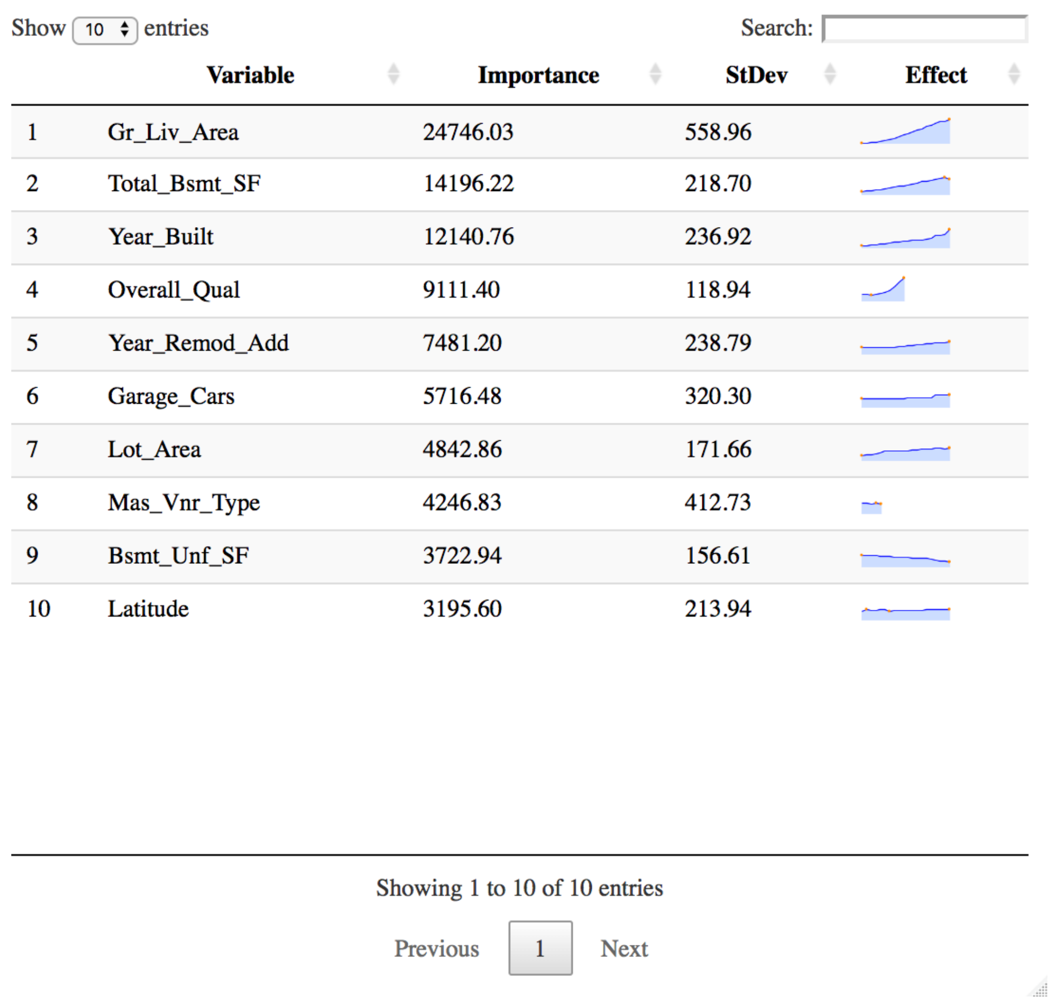
Weston, 2015b)) so any parallel backend supported by the **foreach** package should work. Below we use a *socket* approach with the **doParallel** backend (Revolution Analytics and Weston, 2015a) using a cluster of size five.

```
# Setup parallel backend
library(doParallel) # load the parallel backend
cl <- makeCluster(5) # use 5 workers
registerDoParallel(cl) # register the parallel backend

# Permutation-based feature importance
set.seed(278)
var_imp <- vi(sl, method = "permute", train = X, target = y, metric = "rmse",
              pred_wrapper = imp_fun, nsim = 5, parallel = TRUE)

# Add sparkline representation of feature effects
add_sparklines(var_imp[1L:10L, ], fit = sl, pred.fun = par_fun, train = X,
               digits = 2, verbose = TRUE, trim.outliers = TRUE,
               grid.resolution = 20, parallel = TRUE)

# Shut down cluster
stopCluster(cl)
```


Show 10 entries Search:

	Variable	Importance	StDev	Effect
1	Gr_Liv_Area	24746.03	558.96	
2	Total_Bsmt_SF	14196.22	218.70	
3	Year_Built	12140.76	236.92	
4	Overall_Qual	9111.40	118.94	
5	Year_Remod_Add	7481.20	238.79	
6	Garage_Cars	5716.48	320.30	
7	Lot_Area	4842.86	171.66	
8	Mas_Vnr_Type	4246.83	412.73	
9	Bsmt_Unf_SF	3722.94	156.61	
10	Latitude	3195.60	213.94	

Showing 1 to 10 of 10 entries

Previous 1 Next

Figure 14: VIP with sparkline representation of feature effects for the top ten features from a Super Learner fit to the Ames housing data.

Summary

TBD.

Bibliography

P. Biecek. Dalex: Explainers for complex predictive models in r. *Journal of Machine Learning Research*, 19(84):1–5, 2018. URL `http://jmlr.org/papers/v19/18-416.html`. [p1]

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`. [p2]

L. Breiman. Bagging predictors. *Machine Learning*, 8(2):209–218, 1996. URL `https://doi.org/10.1023/A:1018054314350`. [p2]

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. URL `https://doi.org/10.1023/A:1010933404324`. [p11]

W. Chang. *R6: Encapsulated Classes with Reference Semantics*, 2019. URL `https://CRAN.R-project.org/package=R6`. R package version 2.4.0. [p1]

T. Chen, T. He, and M. Benesty. *xgboost: Extreme Gradient Boosting*, 2017. URL `https://CRAN.R-project.org/package=xgboost`. R package version 0.6-4. [p14]

W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979. doi: 10.1080/01621459.1979.10481038. [p2]

R. C. Fisher, A. and F. Dominici. Model class reliance: Variable importance measures for any machine learning model class, from the "rashomon" perspective. *arXiv preprint arXiv:1801.01489*, 2018. [p11]

J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010. URL `http://www.jstatsoft.org/v33/i01/`. [p14]

J. H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991. URL `https://doi.org/10.1214/aos/1176347963`. [p2, 6]

D. G. Garson. Interpreting neural-network connection weights. *Artificial Intelligence Expert*, 6(4):46–51, 1991. [p8]

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`. [p8]

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. [p8]

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`. [p10]

B. Greenwell and B. Boehmke. *vip: Variable Importance Plots*, 2018. URL `https://CRAN.R-project.org/package=vip`. R package version 0.1.0. [p2]

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`. [p9]

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. [p2, 8]

A. Karatzoglou, A. Smola, K. Hornik, and A. Zeileis. Kernlab – an S4 package for kernel methods in R. *Journal of Statistical Software*, 11(9):1–20, 2004. URL `https://doi.org/10.18637/jss.v011.i09`. [p14]

M. Kuhn. *AmesHousing: The Ames Iowa Housing Data*, 2017a. URL `https://CRAN.R-project.org/package=AmesHousing`. R package version 0.0.3. [p14]

M. Kuhn. *caret: Classification and Regression Training*, 2017b. URL https://CRAN.R-project.org/package=caret. R package version 6.0-76. [p2]

M. Kuhn and K. Johnson. *Applied Predictive Modeling*. SpringerLink : Bücher. Springer New York, 2013. ISBN 9781461468493. [p2]

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. [p2]

S. Milborrow. *earth: Multivariate Adaptive Regression Splines*, 2017. URL https://CRAN.R-project.org/package=earth. R package version 4.5.0. [p6, 14]

C. Molnar. *Interpretable Machine Learning*. 2019. https://christophm.github.io/interpretable-ml-book/. [p1, 13]

C. Molnar, B. Bischl, and G. Casalicchio. iml: An r package for interpretable machine learning. *JOSS*, 3(26):786, 2018. doi: 10.21105/joss.00786. URL http://joss.theoj.org/papers/10.21105/joss.00786. [p1]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2019. URL https://CRAN.R-project.org/package=tibble. R package version 2.1.1. [p2]

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. [p8]

E. Polley, E. LeDell, C. Kennedy, and M. van der Laan. *SuperLearner: Super Learner Prediction*, 2018. URL https://CRAN.R-project.org/package=SuperLearner. R package version 2.0-24. [p14]

Revolution Analytics and S. Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2015a. URL https://CRAN.R-project.org/package=doParallel. R package version 1.0.10. [p16]

Revolution Analytics and S. Weston. *foreach: Provides Foreach Looping Construct for R*, 2015b. URL https://CRAN.R-project.org/package=foreach. R package version 1.4.3. [p15]

G. Ridgeway. *gbm: Generalized Boosted Regression Models*, 2017. URL https://CRAN.R-project.org/package=gbm. R package version 2.1.3. [p2]

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. [p8]

H. Wickham. The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, 40(1): 1–29, 2011. URL https://doi.org/10.18637/jss.v040.i01. [p15]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org. [p2]

M. N. Wright. *ranger: A Fast Implementation of Random Forests*, 2017. URL https://CRAN.R-project.org/package=ranger. R package version 0.7.0. [p14]

Y. Xie, J. Cheng, and X. Tan. *DT: A Wrapper of the JavaScript Library 'DataTables'*, 2018. URL https://CRAN.R-project.org/package=DT. R package version 0.5. [p13]

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