vivid: An R package for Variable Importance and Variable Interactions Displays for Machine Learning Models

by Alan Inglis, Andrew Parnell, and Catherine Hurley

Abstract We present **vivid**, an R package for visualizing variable importance and variable interactions in machine learning models. The package provides heatmap and graph-based displays for viewing variable importance and interaction jointly and partial dependence plots in both a matrix layout and an alternative layout emphasizing important variable subsets. With the intention of increasing a machine learning models' interpretability and making the work applicable to a wider readership, we discuss the design choices behind our implementation by focusing on the package structure and providing an in-depth look at the package functions and key features. We also provide a practical illustration of the software in use on a data set.

Introduction

Our motivation behind the creation of the **vivid** package is to investigate machine learning models in a way that is simple to understand while also offering helpful insights into how variables affect the fit. We do this through the use of heatmaps, network graphs, and both a generalized pairs plot style partial dependence plot (PDP) (Friedman 2000) and a space saving PDP based on key variable subsets. While the techniques and fundamental goals of these visualizations have been discussed in Inglis, Parnell, and Hurley (2022), we focus here on the implementation details of the package by providing a complete listing of the functions and arguments included in the **vivid** package with further examples indicating advanced usage beyond that previously shown. In this work we examine the decisions made when designing the package and provide an in-depth look at the package functions and features with the intention of making the work applicable to a larger readership. This article outlines the general architectural principles implemented in **vivid**, such as the data structures we use and data formatting, function design, filtering techniques, and more. We illustrate each function by way of a practical example. Our package is available on the Comprehensive R Archive Network at `https://cran.r-project.org/web/packages/vivid` or on GitHub at `https://github.com/AlanInglis/vivid`.

In recent years machine learning (ML) algorithms have emerged as a valuable tool for both industry and science. However, due to the black-box nature of many of these algorithms it can be challenging to communicate the reasoning behind the algorithm's decision-making processes. With the need for transparency in ML growing it is important to gain understanding and clarity about how these algorithms are making predictions (Antunes et al. 2018; Felzmann et al. 2019). Many R packages are now available that aid in creating interpretable machine learning (IML) models such as **iml** (Molnar, Bischl, and Casalicchio 2018), **DALEX** (Biecek 2018), and **lime** (Hvitfeldt, Pedersen, and Benesty 2022). For a comprehensive review of IML see Molnar (2022), and Biecek and Burzykowski (2021).

How we choose to visualize aspects of the model output is of vital importance in how a researcher can interpret and communicate their findings. Consequently, model summaries such as variable importance and variable interactions (VImp and VInt; together we term these VIVI) are frequently used in various fields to comprehend and explain the hidden structure in an ML fit. In ecology they are employed to determine the causes of ecological phenomena (e.g. Murray and Conner 2009); in meteorology VImp measures and partial dependence plots are used to examine air quality (e.g. Grange et al. 2018); in bioinformatics, understanding gene-environment interactions have made these measures an important tool for genomic analysis (e.g. Chen and Ishwaran 2012).

In Table 1 we summarize VIVI measures and visualizations provided by a selection of R packages. VIVI measures models fall into two categories; model specific (embedded) methods or model agnostic methods. In embedded methods the variable importance is incorporated into the ML algorithm. For example, random forests (RF) (Breiman 2001) and gradient boosting machines (GBM) (Friedman 2000) use the tree structure to evaluate the performance of the model. Bayesian additive regression tree models (BART) (Chipman, George, and McCulloch 2010) also use an embedded method to obtain VIVI measures by looking at the proportion of splitting rules used in the trees. The package **randomForestExplainer** (Paluszynska, Biecek, and Jiang 2020) provides a set of tools to understand what is happening inside a random forest and uses the concept of minimal depth (Ishwaran et al. 2010) to assess both importance and interaction strength by examining the position of a variable within the trees. For gradient boosted machines, the **EIX** (Maksymiuk, Karbowiak, and Biecek 2021) package can be used to measure and identify VIVI and visualize the results.

Package	Description	Visualisation
vip	A general framework for constructing VImp plots from various types of ML models in R. Both model-agnostic and model-specific methods are catered for.	Has built-in ggplot2 functionality to display VIVI measures. Also provides univariate PDPs and ICE curves and ability to plot Shapley values.
iml	A general framework for analysing the behavior of ML models. Includes model-agnostic VIVI measures.	Ability to plot VIVI measures using lollipop, dot, and barplots. Also includes univariate and bivariate PDPs, ICE curves, LIME, and Shapley visualizations. Built with ggplot2.
flashlight	A general framework for analyzing the behavior of ML models. Includes model-agnostic VIVI measures.	Ability to plot VIVI measures using barplots. Includes univariate and bivariate PDPs, ICE curves, Global surrogate, and SHAP visualizations. Built with ggplot2 .
DALEX	A general framework for analyzing the behavior of ML models. Includes model-agnostic VImp measures.	Contains a suite of visualizations including Ceteris Paribus, Shapley, PDPs, model performance, and diagnostic plots. Built with ggplot2.
lime	A general framework for fitting a local interpretable model. Includes model-agnostic VImp measures.	Ability to create VImp and model visualizations using barplots and heatmaps. Can also create interactive plots. Built with ggplot2.
randomForestExplainer	Contains a set of model-specific tools to determine which random forests variables are most important. Can assess VIVI.	Ability to create VIVI plots displaying the mean minimal depth distribution and conditional minimal depth. Can also display multi-way importance, pairs plots containing different metrics, and Bivariate PDP. Built with ggplot2.
EIX	Contains a set of model-specific tools to determine which GBM variables are most important. Can assess VIVI.	Ability to create VIVI plots using lollipops, barplots, and heatmaps. Can also display dot and radar plots. Built with ggplot2.
varImp	Computes model-specific random forest VImps for the conditional inference random forest (cforest) of the party package.	None available.
bartMachine	Used to build Bayesian additive regression tree models. Can assess VIVI.	Ability to plot VIVI measures with uncertainty included using barplots. Also includes a suite of model diagnostic plots and univariate PDP. Built using base R.
pdp	A general framework for constructing PDPs from various types machine learning models, bivariate, and trivariate PDPs and ICE curves. Built with ggplot2.	

Table 1: Summary of a selection of R packages that can be used to assess the variable importance, variable interactions, or partial dependence and if these metrics are model-specific or model-agnostic. A brief description of available visualizations for evaluating model behavior is also provided. For more on the lime and varImp packages see Hvitfeldt, Pedersen, and Benesty (2022) and Probst (2020), respectively.

Model-agnostic methods are techniques that can, in principle, be applied to any ML algorithm. Agnostic methods not only provide flexibility in relation to model selection but are also useful for comparing different fitted ML models. An example of a model agnostic approach for evaluating VImps is permutation importance (Breiman 2001). This method calculates the difference in a model's predictive performance following a variable's permutation; implementations are available in the packages **iml**, **flashlight**, **DALEX** and **vip** (Greenwell and Boehmke 2020). For VInts, Friedman's H-statistic (Friedman and Popescu 2008) is an agnostic interaction measure derived from the partial dependence by comparing a pair of variables' partial dependency with their marginal effects. Packages **iml** and **flashlight** provide implementations.

Partial dependence plots (PDPs) were first introduced by Friedman (2000) as a model agnostic way to visualize the relationship between a specified predictor variable and the fit, averaging over other predictors' effects. Similar to PDPs, individual conditional expectation curves (ICE) (Goldstein et al. 2015) show the relationship between a specified predictor and the fit, fixing the levels of other predictors at those of a particular observation. PDP curves are then the average of the ICE curves over all observations in the dataset. R packages offering PDPs include **iml**, **DALEX**, **pdp** (Greenwell 2017); the package **ICEbox** (Goldstein et al. 2015) provides ICE curves and variations.

In **vivid** we provide a suite of functions (see Table 2) for calculating and visualizing variable importance, interactions and the partial dependence. Our displays conveniently show (both model specific and agnostic) VImp and VInt jointly using heatmaps and network graphs, thus providing a more informative picture identifying relevant features. Our generalized PDP (GPDP) displays partial dependence plots in a matrix layout combining univariate and bivariate partial dependence plots with variable scatterplots. We furthermore provide a more compact version of the GDPD, the so-called zen-partial dependence plot (ZPDP) consisting only of those bivariate partial dependence plots with high VInt. All of our displays are designed to quickly identify how variables, both singly and jointly, affect the fitted response and can be used for regression or classification fits. As the output of our displays are **ggplot2** objects (Wickham 2016), they are easily customizable and provide the flexibility to create custom VIVI visualizations.

Function	Description	Type
`vivi`	Create a VIVI matrix of class `vivid`.	VIVI construction
`viviReorder`	Reorders a square matrix so high VIVI values are pushed to the top left of the matrix.	VIVI construction
`CVpredictfun`	Heatmap plot of VIVI values.	Visualization
`viviHeatmap`	Network plot of VIVI values.	Visualization
`viviNetwork`	Univariate partial dependence plot with ICE curves displayed as a grid.	Visualization
`pdpVars`	Univariate partial dependence plot with ICE curves displayed as a grid.	Visualization
`pdpPairs`	Pairs plot showing bivariate PDP, ice/univariate PDP, and data.	Visualization
`pdpZen`	A zigzag expanded navigation plot (zenplot) displaying partial dependence values.	Visualization
`zPath`	Constructs a zenpath for connecting and displaying pairs to be used with `pdpZen`.	Utility
`as.data.frame.vivid`	Takes a matrix of class `vivid` and turns it into a data frame.	Utility
`vip2vivid`	Takes measured importance and interactions from the `vip` package and turns them into `vivid` matrix which can be used for plotting	Utility

Table 2: Summary of functions available in the **vivid** package. The main construction function is `vivi` which is used to calculate the VIVI values for subsequent use in the visualizations.

This paper is structured as follows. First we introduce a dataset and fit models that will be used as examples throughout this paper. Following this, we describe **vivid** functionality for calculating VIVI. We then move on to visualizations and focus on the functionality provided by the two functions `viviHeatmap` and `viviNetwork` for displaying VIVI, and two functions for displaying PDPs namely, `pdpPairs` and `pdpZen`. Finally we provide some concluding discussion.

Example: Data and Models

The well-known Boston housing data (Harrison Jr and Rubinfeld 1978) from the R package **MASS** (Venables and Ripley 2002) concerns prices of 506 houses and 14 predictor variables including property attributes such as number of rooms and social attributes including crime rate and pollution levels. The response is the median value of owner-occupied homes in $1000s (medv).

We first fit a random forest (using the **randomForest** package). In order to avail of embedded variable importance scores for the random forest, the `importance` argument must be `TRUE`.

```
library("randomForest")
library("MASS")
set.seed(1701)
data("Boston")

rf <- randomForest(medv ~.,
                   data = Boston,
                   importance = TRUE)
```

Next we fit a gradient boosted machine (using the **xgboost** package). For the GBM we set the maximum number of boosting iterations, `nrounds`, to 100 as no default is provided in **xgboost**.

```
library("xgboost")
gbst <- xgboost(data = as.matrix(Boston[,1:13]),
                label =  as.matrix(Boston[,14]),
                nrounds = 100,
                verbose = 0)
```

In the following sections we will explain how aspects of the two fits can be compared with **vivid** software. We will also explain aspects of our software design with reference to these fits.

Calculating VIVI

The first step in using **vivid** is to calculate variable importance and interactions for a model fit. The `vivi` function calculates both of these, creating a square, symmetric matrix containing variable importance on the diagonal and variable interactions on the off-diagonal. Required inputs are a fitted ML model, a data frame on which the model was trained, and the name of the response variable for the fit. The returned matrix has importance and interaction values for all variables in the supplied data frame, excluding the response. Variables that are not used by the supplied ML fit will have their importance and interaction values set to zero. Our visualizations functions `viviHeatmap` and `viviNetwork` are designed to show the results of a `vivi` calculation, but will work equally well for any square matrix with identical row and column names. (Note, the symmetry assumption is not required for `viviHeatmap` and `viviNetwork` uses interaction values from the lower-triangular part of the matrix only.)

The code snippet below shows the creation of a `vivid` matrix for the random forest fit. For clarity, we include all of the `vivi` function arguments for the random forest fit, though only the first three are required. Other inputs will be described in the section `vivi` function additional arguments.

```
library("vivid")

set.seed(1701)
viviRf <- vivi(fit = rf,
               data = Boston,
               response = "medv",
               reorder = FALSE,
               normalized = FALSE,
               importanceType = 'agnostic',
               gridSize = 50,
               nmax = 500,
               class = 1,
               predictFun = NULL)
```

In the absence of any model-specific importance measure we use an agnostic permutation method described by Fisher, Rudin, and Dominici (2019) to obtain the variable importance scores. In this method a model error score (root mean square error) is calculated, then each feature is randomly permuted and the model error is re-calculated. The difference in performance is considered to be the variable importance score for that feature.

The `vivi` function calculates importance using an S3 method called `vividImportance`. We provide methods for **randomForest**, **ranger** (Wright and Ziegler 2017), **mlr** (Bischl et al. 2016), **mlr3** (Lang et al. 2019), and **tidymodels** (Kuhn and Wickham 2020) to access embedded model-specific measures. When `vivi` is provided with a model fitted using one of these packages, importance defaults to the embedded method, as set when the model was fit. By specifying `importanceType = "agnostic"` in the call to `vivi` as in the example above, agnostic importance is calculated instead. If the model fit offers more than one embedded importance measure, these may be selected by specifying suitable values to `importanceType`. `vivid` relies on the package **flashlight** package to calculate agnostic importance via `flashlight::light_importance` which currently works for numeric and numeric binary responses only.

For variable interactions, we use the model-agnostic Friedman's H-statistic to identify any pairwise interactions. As discussed in Inglis, Parnell, and Hurley (2022), we recommend the unnormalized version of the H-statistic which prevents detection of spurious interactions which can occur when the bivariate partial dependence function (used in the construction of the H-statistic) is flat. In the case of a binary response classification model, we follow Hastie, Tibshirani, and Friedman (2009) and compute the H-statistic and partial dependence on the logit scale.

The `vivi` function calculates interactions using an S3 method called `vividInteraction`, which again relies on the **flashlight** package to calculate Friedman's H-statistic via `flashlight::light_interaction`. Friedman's H-statistic is the only interaction measure currently available in **vivid**, though the method of Greenwell, Boehmke, and McCarthy (2018) could also be used for this purpose. Embedded interaction measures could easily be incorporated via S3 methods in future.

flashlight simplifies the calculation of VIVI values as it allows a custom predict function to be supplied for the calculation of agnostic importance and the H-statistic; this flexibility means importance and the H-statistic can be calculated for any ML model. We supply an internal custom predict function called `CVpredictfun` to both `flashlight::light_importance` and `flashlight::light_interaction`. `CVpredictfun` is a wrapper around `CVpredict` from the **condvis2** package (C. Hurley, OConnell, and Domijan 2022), which adds an option for the classification to select (via the `class` argument to `vivi`) the class to be used for prediction and calculates predictions on the logit scale by default. `CVpredict` accepts a broad range of fit classes thus streamlining the process of calculating VIVI.

In situations where the fit class is not handled by `CVpredict` (as is the case for the GBM model created from **xgboost**), supplying a custom predict function to the `vivi` function by way of the `predictFun` argument allows the agnostic VIVI values to be calculated. In the code snippet below, we build the `vivid` matrix for the GBM fit using a custom predict function, which must be of the form given in the code snippet. For brevity we omit some of the optional `vivi` function arguments.

```
# predict function for GBM
pFun <- function(fit, data, ...) predict(fit, as.matrix(data[,1:13]))

set.seed(1701)
viviGBst <- vivi(fit = gbst,
                 data = Boston,
                 response = "medv",
                 reorder = FALSE,
                 normalized = FALSE,
                 predictFun = pFun)
```

`vivi` function additional arguments

The `vivi` function has 10 arguments. Some of these have been discussed above, including `fit`, `data`, `response`, `importanceType`, and `predictFun`. Here we provide a summary of the remaining arguments. First, the `normalized` argument determines if Friedman's H-statistic should be normalized or not (see Inglis, Parnell, and Hurley (2022), for the pros and cons of each version). The arguments `gridSize` and `nmax` are used to set the size of the grid for evaluating the predictions and maximum number of data rows to consider, respectively, in the calculation of the H-statistic. Lowering the grid size can provide a significant speed boost, though at the expense of predictive accuracy. Additionally, sampling the data via `nmax` offers a speed boost. The default values for `gridSize` and `nmax` are 50 and 500, respectively.

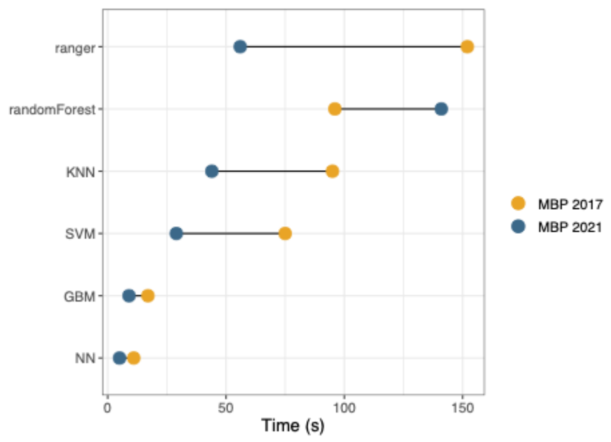

Figure 1: Mean time over five runs, on two MacBooks, for the creation of a vivid matrix for different models. Times are highly dependent on the model fit, with NN the fastest and random forests the slowest.

Speed tests

A drawback of using Friedman's H-statistic as a measure of interaction is that it is a computationally expensive calculation, and may be especially time-consuming for models where prediction is slow. Figure 1 shows the build time (rounded to the nearest second) averaged over five runs for the creation of a `vivid` matrix with default parameters for different ML algorithms using the Boston Housing data. As the Boston housing data has 13 predictor variables, Friedman's H-statistic is computed for 91 predictor pairs. The ML algorithms are: GBM, random forest, support vector machine (SVM), neural network (NN), and k-nearest neighbors (KNN). The SVM, NN, and KNN were built using the **e1071** (Meyer et al. 2021), **nnet** (Venables and Ripley 2002), and **kknn** (Schliep and Hechenbichler 2016) packages, with the KNN being built through the **mlr3** (Lang et al. 2019) framework. Each of the models were built using their default settings and, for each model fit, the agnostic VImp was measured. The speed tests were performed on both a 2017 MacBook Pro 2.3 GHz Dual-Core Intel Core i5 with 8GB of RAM and a 2021 32GB MacBook M1 Pro. Here we are essentially comparing predict times for the various fits. The NN fit created using the **nnet** package was the fastest, followed by GBM. Both random forests are the slowest, and surprisingly, the older Mac beats its higher spec cousin for the **randomForest** fit.

Alternative construction of a `vivid` matrix

A `vivid` matrix may also be obtained from variable importance and interaction values calculated elsewhere. The package **vip** offers these, and evaluates interactions using a method called the *feature importance ranking measure* (FIRM, see Greenwell, Boehmke, and McCarthy (2018), for more details). The `vip2vivid` function we provide in **vivid** takes VIVI values created in **vip** and turns them into a `vivid` matrix, that can be subsequently used with our plotting tools. For example, in the code below, model-specific VImp and FIRM VInt scores are calculated for the random forest fit, and subsequently arranged into a `vivid` matrix with the VImps on the diagonal and VInts on the off-diagonal.

```
library("vip")
# get model specific VImps using vip package
vipVImp <- vi(rf, method = 'model')
# get VInts using vip package
vipVInt <- vint(rf, feature_names = names(Boston[-14]))

# turn into vivi-matrix
vipViviMat <- vip2vivid(importance = vipVImp, interaction = vipVInt)
```

Heatmap of Variable Importance and Variable Interactions

The `viviHeatmap` function constructs a heatmap displaying both importance and interactions, with importance on the diagonal and interactions on the off-diagonals. A `vivid` matrix is the only required input (not necessarily symmetric). Color palettes for the importance and interactions are optionally provided via `impPal` and `intPal` arguments. For the default color palette we choose single-hue, color-blind friendly sequential color palettes from Zeileis et al. (2020), where low and high VIVI values are represented by low and high luminance color values respectively, aiding in highlighting values of interest.

The ordering of the heatmap is taken from the ordering of the input matrix. As `reorder` was set to `FALSE` when building both the random forest and GBM fit `vivid` matrix, the ordering of the heatmaps matches the variable order in the dataset. This is useful for directly comparing multiple heatmaps, however it does not necessarily lend itself for easy identification of the largest VIVI values. If we were to seriate both `vivi-matrices` separately, we would end up with different optimal orderings for each matrix. An alternative is to create a common ordering by averaging over the two `vivid` matrix objects and applying the `vividReorder` function to the result. (This function uses a seriation algorithm based on the techniques of Earle and Hurley (2015) designed to place high interaction variables adjacently and to pull high VIVI variables towards the top-left; see Inglis, Parnell, and Hurley (2022) for details.) Both VIVI matrices are then re-ordered using the newly obtained variable order.

```
# average over matrices and seriate to get common ordering
viviAvg <- (viviRf + viviGBst) / 2
viviAvgReorder <- vividReorder(viviAvg)

# reorder vivi-matrices
ord <- colnames(viviAvgReorder)
viviRf <- viviRf[ord,ord]
viviGBst <- viviGBst[ord,ord]
```

Arguments `impLims` and `intLims` specify the range of importance and interaction values to be mapped to colors. Default values are calculated from the maximum and minimum VIVI values in the `vivid` matrix. Importance and interaction values falling outside the supplied limits are squished to the closest limit. It can be useful to specify these limits in the situation where there is an extremely large VIVI value that dominates the display, or where we wish two or more plots to have the same limits for comparison purposes, as in the example below.

Figure 2 shows our improved ordering so that variables with high VIVI values are pushed to the top left of the plots. Filtering can also be applied to the input matrix to display a subset of variables. When compared to the GBM fit in (b), the random forest fit in (a) has weaker interactions and lower importance scores. Both plots identify *lstat* as being the most important. Both fits also show that *lstat* interacts with several other variables, though the interactions are much stronger for the GBM. Notably, the strongest interaction in both fits are different. These are $lstat : crim$ (where *crim* is the per capita crime rate by town) for the random forest fit and $lstat : nox$ (where *nox* is parts per 10 million nitrogen oxides concentration) for the GBM fit.

```
viviHeatmap(viviRf, angle = 45, intLims = c(0,1), impLims = c(0,8))
viviHeatmap(viviGBst, angle = 45, intLims = c(0,1), impLims = c(0,8))
```

Network of Variable Importance and Variable Interactions

The `viviNetwork` function constructs a network graph displaying both importance and interactions. Similarly to `viviHeatmap` , this function takes a `vivid` matrix as the only required input and provides a visual representation of the magnitude of the importance and interaction values through the size of the nodes and edges in the graph, in addition to color. In the plot each variable is represented as a node, with its importance being represented through size and color such that larger, darker nodes indicate a higher importance. Each pairwise interaction is represented by a connecting edge, where larger interaction values get thicker, darker edges; Figure 3 provides an example. This type of plot benefits from being able to quickly identify the magnitude of the importance and interactions of the variables that have the most impact on the response. The `viviNetwork` function optional arguments follows the same conventions as `viviHeatmap`: custom color palettes for importance and interactions are provided via the `impPal` and `intPal`, and the range of VIVI values to be mapped to the colors are specified via the `impLims` and `intLims`.

By default, we choose a circular layout to display the graphs, as when coupled with the seriated `vivid` matrix, variables with high VIVI are grouped in a clock-wise arrangement starting at the top. This arrangement allows for easy identification of variables with high VIVI. Custom layouts are possible by providing a numeric matrix with two columns and one row per node to the `layout` argument. Additionally, any of the layouts available in the **igraph** package (Csardi and Nepusz 2006) can be specified.

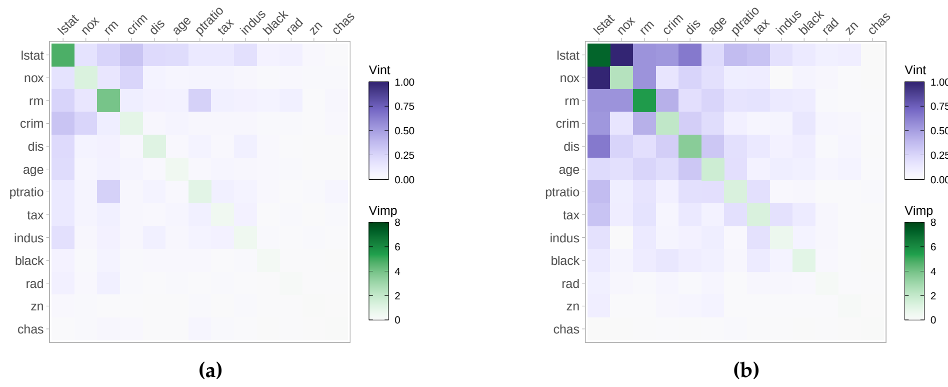

Figure 2: Agnostic variable importance and variable interaction scores for a random forest fit in (a) and GBM fit in (b) on the Boston housing data displayed as a heatmap. The random forest fit has weaker interactions and lower importance scores than the GBM fit. Both fits identify *lstat* as the most important followed by *rm*. In both fits, *lstat* has numerous interactions with other variables, notably *crim* in the random forest fit and *nox* in the GBM fit.

We provide options to filter the graph via the `intThreshold` and `removeNode` arguments. This helps to highlight variables with high VIVI scores, which is useful in settings with many predictors. The `intThreshold` argument filters edges with weight (i.e., VInt value) below a specified value and `removeNode` removes nodes with no connecting edges after thresholding interaction values. We can optionally cluster similar variables together with respect to their VIVI scores via the `cluster` argument, thereby aiding in the process of highlighting variables of interest. The `cluster` argument can take either a vector of cluster memberships for nodes or an appropriate **igraph** clustering function.

We demonstrate network plots displaying VIVI values for the GBM fit. In Figure 3, we show both a default network plot including all variables in (a) and a filtered and clustered network plot in (b). For the filtered plot we select VIVI values in the top decile. This selection allows us to focus only on the variables with the most impact on the response. The variables that remain are *lstat*, *nox*, *rm*, *crim*, *dis* (weighted mean of distances to five Boston employment centers), *tax* (full-value property-tax rate per $10,000), and *ptratio* (pupil-teacher ratio by town). We then perform a hierarchical clustering treating variable interactions as similarities, with the goal of grouping together high-interaction variables. Finally we rearrange the layout using **igraph**. Here, `igraph::layout_as_star` places the first variable (deemed most relevant using the VIVI seriation process above) at the center, which in Figure 3 (b) emphasizes its key role as the most important predictor which also has the strongest interactions.

```
# default network plot for GBM
viviNetwork(viviGBst)

# clustered and filtered network for GBM
intVals <- viviGBst
diag(intVals) <- NA

# select VIVI values in top 10%
impTresh <- quantile(diag(viviGBst),.9)
intThresh <- quantile(intVals,.9,na.rm=TRUE)
sv <- which(diag(viviGBst) > impTresh |
              apply(intVals, 1, max, na.rm=TRUE) > intThresh)

h <- hclust(-as.dist(viviGBst[sv,sv]), method="single")

viviNetwork(viviGBst[sv,sv],
            intLims = c(0,1),
            impLims = c(0,8),
            cluster = cutree(h, k = 3), # specify number of groups
            layout = igraph::layout_as_star)
```

In Figure 3(a), when displaying all the variables, we can clearly identify which variables have the highest VIVI values. The large darker nodes of *lstat* and *rm* indicate their importance and the dark, thick connecting edge between *lstat* and *nox* tell us that these two variables strongly interact. In (b),

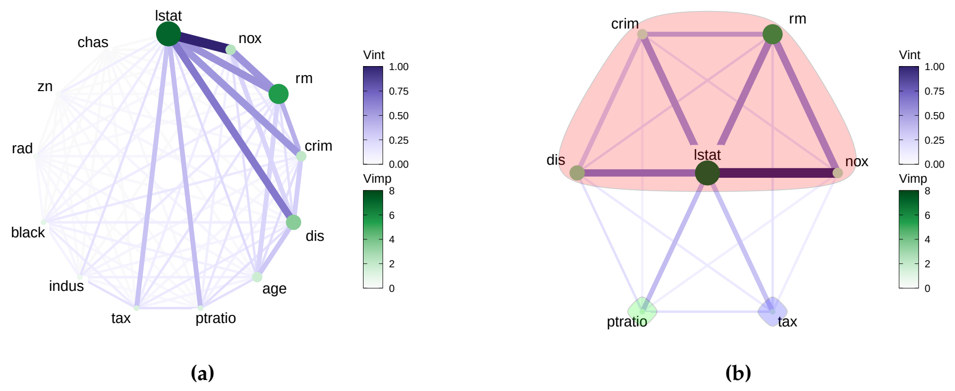

Figure 3: Network plots showing VIVI scores obtained from a GBM fit on the Boston housing data. In (a) we display the all values in a circle. In (b) we use a hierarchical clustering to group variable with high VIVI together and rearrange the layout using an igraph function.

after applying a hierarchical clustering on the variables with a VIVI value above the median, we can see the strongest mutual interactions have been grouped together for the GBM fit. Namely, *lstat*, *nox*, *crim*, *rm*, and *dis* are all grouped together. The remaining variables are individually clustered.

We provide a conversion of `vivid` matrix objects to a data frame via an `as.data.frame` method, as demonstrated below. This facilitates plotting with base R and **ggplot2**, for example a barplot of either VImp or VInt values. Note that while `vivi` returns a matrix of class `vivid`, the class attribute was dropped when the matrix was re-ordered.

```
class(viviRf)<- c("vivid", class(viviRf))
head(as.data.frame(viviRf), 4)

#>   Variable_1 Variable_2     Value Measure Row Col
#> 1      lstat      lstat 4.7720394    Vimp   1   1
#> 2        nox      lstat 0.1789794    Vint   2   1
#> 3         rm      lstat 0.2562817    Vint   3   1
#> 4       crim      lstat 0.3371364    Vint   4   1
```

Partial Dependence and Individual Conditional Expectation Curves

Univariate Partial Dependence Plot

The `pdpVars` function constructs a grid of univariate PDPs with ICE curves for selected variables. We use ICE curves to assist in the identification of linear or non-linear effects. The fit, data frame used to train the model, and the name of the response variable are required inputs. In the code below, we show an example of the partial dependence and ICE curves for the first five features from the GBM `vivid` matrix, with output shown in Figure 4. We use the custom GBM predict function given previously.

```
top5 <- colnames(viviGBst)[1:5]
pdpVars(data = Boston,
        fit = gbst,
        response = 'medv',
        vars = top5,
        predictFun = pFun)
```

All of our PDP variants handle categorical responses and predictors. The color palette is customized via the `pal` argument. In all of our PDPs, this defaults to a diverging palette which accentuates fitted values that differ from the average. Dark red and dark blue are used to indicate high and low values of $\hat{y}$ respectively. The middle values are displayed in yellow. The `nIce` argument specifies the number of ICE curves to be drawn. This is either a single number specifying the number of observations to be sampled for the ICE curves, or a vector of row indices, an option that is useful for example to display ICE curves from particular classes. The default value for `nIce` is 30, which allows individual curves to be seen.

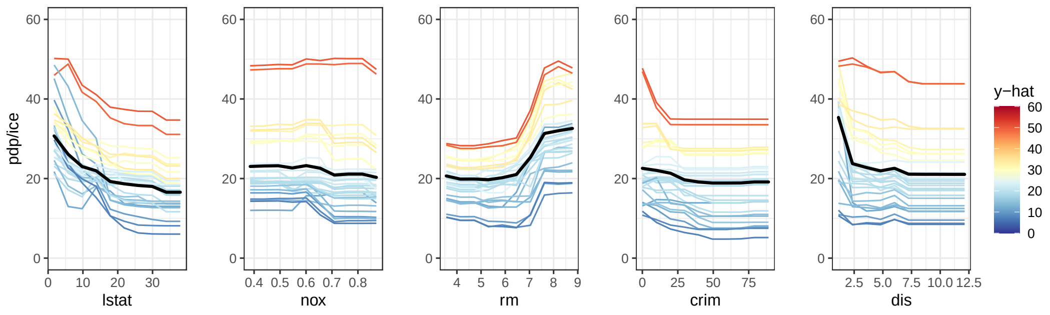

Figure 4: Partial dependence plots (black line) with individual conditional expectation curves (colored lines) of a GBM fit on the Boston housing data. The changing partial dependence and ICE curves of *lstat* and *rm* indicate that these variables have some impact on the response.

The ordering of the PDPs is taken from the ordering of variables in the data set, or may be specified via the `vars` argument. As with the construction of the `vivid` matrix, the `gridSize` and `nmax` arguments determine the number of predictions.

In Figure 4 we can see from the changing PDP and ICE curves that *lstat* and *rm* have the clearest impact on the response, with the predicted median house price being higher for low values of *lstat* and high values of *rm*. Additionally, the predicted median house price appears to be higher for low values of *dis* before leveling off at around 2.5. The remaining variables have generally flat partial dependence and ICE curves.

Generalized Pairs Partial Dependence Plot

The `pdpPairs` function creates a generalized pairs partial dependence plot (GPDP). In our GPDP, we use a matrix layout and plot the univariate partial dependence (with ICE curves) on the diagonal, bivariate partial dependence on the upper diagonal and a scatterplot of variables on the lower diagonal, where all colours are assigned to points and ICE curves by the predicted $\hat{y}$ value. As with the univariate PDP, the fit, data frame used to train the model, and the name of the response variable are required inputs.

```
set.seed(1701)
pdpPairs(data = Boston,
         fit = gbst,
         response = "medv",
         gridSize = 20,
         nIce = 50,
         vars = top5,
         convexHull = TRUE,
         fitlims = "pdp",
         predictFun = pFun)
```

In the above code, we display only the interesting variables seen in previous plots via the `var` argument. We also chose to display 50 ICE curves. As with `pdpVars`, additional arguments specify the color palette and number of ice curves, while `gridSize` and `nmax` determine the number of predictions.

For our GPDP, we follow the general design choices in **vivid** and specify the range of predicted values to be mapped to the colors via the `fitlims` argument. We set the default fit range for the color map for the GPDP to the range of the collection of PDP surfaces with `fitlims = "pdp"`. The setting of this argument at its default value allows for maximum resolution of the bivariate PDPs. Since predictions for specific observations and ICE curves would likely exceed these bounds, the closest value within the color map's bounds is used to allocate colors. Alternatively `fitlims = 'all"` specifies that limits are calculated as the full range of predictions shown.

In the upper diagonals we exclude extrapolated areas from the bivariate PDPs to prevent interpretation of the PDPs in areas where there is no data. The removal of extrapolated areas can be prevented by specifying `convexHull = FALSE`.

In Figure 5, in addition to the univariate PDPs, we capture the effects of the variables on the response via the bivariate PDP on the upper-diagonal and the distribution of the data in the lower-diagonal. The scatterplots are useful for determining if variables are highly correlated, as highly correlated variables may spuriously affect the partial dependence and give erroneous results (Apley

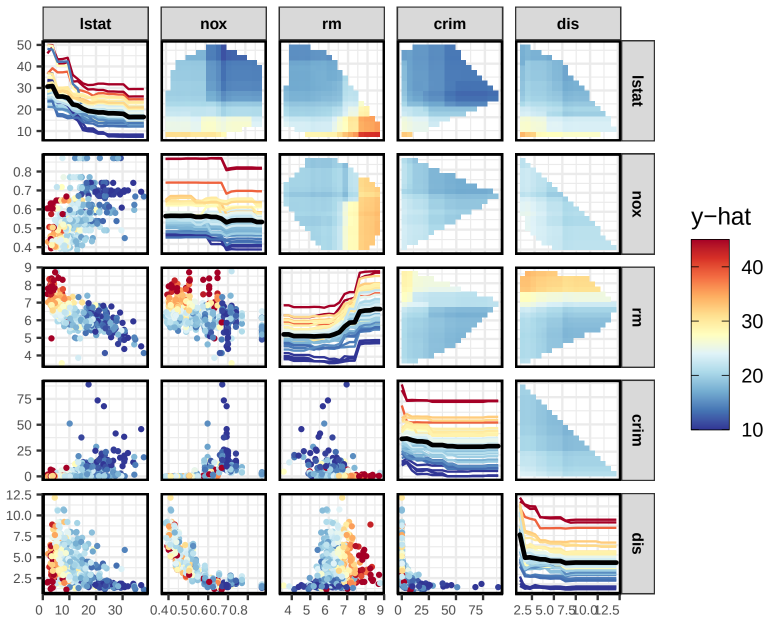

Figure 5: Filtered generalized pairs partial dependence plot for a GBM fit on the Boston housing data. From both the univariate and bivariate PDPs, we can see that *lstat* and *rm* have an impact on the response. As *lstat* decreases and *rm* increases, predicted median house price value goes up. The bivariate PDP of *lstat* : *nox* shows that as *nox* increases, the predicted value decreases.

and Zhu 2020). Of note are the variables *lstat* and *rm*. We can clearly see that when the number of rooms (*rm*) is high and the percentage of lower status of the population (*lstat*) is low, the predicted $\hat{y}$ median house price value is high. This is exemplified in the changing bivariate PDP.

In the case of categorical predictors, the partial dependence for each factor level is shown in the upper-diagonal (for an example of this, see Inglis, Parnell, and Hurley (2022)).

Zen Partial Dependence Plots

The pdpZen function creates partial dependence plots utilizing a space-saving method based on graph Eulerian. These we call zen-partial dependence plots (ZPDP). The display is based on the zigzag expanded navigation plots, known as zenplots , available in the **zenplots** package (Hofert and Oldford 2020). Zenplots were created to display paired graphs of high-dimensional data focusing on the most important 2D displays. In our adaptation we show bivariate PDPs that focus on the most important interacting variables in a compact zigzag layout, helpful when predictor space is high-dimensional.

The code below illustrates pdpZen, here displaying the first five variables from GBM's vivid matrix. Later we show an example focusing on high-interacting pairs of variables. We use the same convention as our previous PDPs with regard to color palette and limits, grid size, and the number of rows considered for evaluation. The ZPDP also has a variable rug plot on each axis to avoid interpretation problems that may occur in the presence of skewness.

```
pdpZen(data = Boston,
       fit = gbst,
       response = "medv",
       convexHull = TRUE,
       zpath = top5,
       predictFun = pFun)
```

The argument zpath specifies the variables to be plotted, defaulting to all dataset variables aside from the response. In the code above, zpath is the vector *lstat*, *nox*, *rm*, *crim* and *dis*. The resulting plot shown in Figure 6 presents the bivariate PDP for every consecutive pair of variables in a zigzag layout.

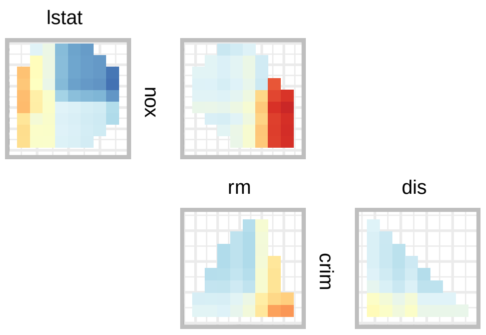

Figure 6: Zen partial dependence plot for the GBM fit on the Boston data. Here we display first five variables from the GBM's 'vivid' matrix. Only plots for consecutive variables are shown.

Zen-paths

ZPDP are most useful when the bivariate PDPs plotted are selected to be an interesting subset of all pairwise plots. To obtain this subset, we consider a network graph displaying VIVI values, such as that in Figure 3 (a). We then filter the edges below a selected interaction value, leaving only highly interacting variable pairs, as in Figure 3(b). Our goal is to then build a ZPDP consisting of the bivariate plots represented by each edge of the thresholded graph. The `zPath` function creates a sequence or sequences of variable paths for use in `pdpZen`.

The `zPath` function takes four arguments. These are: `viv` - a matrix of interaction values, `cutoff` - exclude interaction values below this threshold, `method` - a string indicating which method to use to create the path, and `connect` - a logical value indicating if separate Eulerians should be connected

Two methods are provided, either `"greedy.weighted"` or `"strictly.weighted"`. The first option uses the greedy Eulerian path algorithm (C. B. Hurley and Oldford 2011, 2022) for connected graphs. This visits each edge at least once, beginning at the edge with the highest weight and traversing through the remaining edges, giving priority to the highest-weighted edge. Some edges may be visited more than once or additional edges may be visited if the number of nodes in the graph is not even. The second method `"strictly.weighted"` (provided by **zenplot**) visits edges strictly in decreasing order by weight (here the interaction values). If `connect` is `TRUE` the sequences obtained by the strictly weighted method are concatenated to form a single path.

In the code below, we provide two examples of creating zen-paths, from the top 10% of interaction scores in `viviGBst`.

```
intThresh <- quantile(intVals,.9,na.rm=TRUE)
# set zpaths with different parameters
zpGw  <- zPath(viv = viviGBst, cutoff = intThresh, method = 'greedy.weighted')
zpGw

#>  [1] "nox"     "lstat"   "dis"     "ptratio" "lstat"   "rm"      "crim"
#>  [8] "lstat"   "tax"     "rm"      "nox"

zpSw  <- zPath(viv = viviGBst, cutoff = intThresh, connect = FALSE, method = 'strictly.weighted')
zpSw

#> [[1]]
#> [1] "nox"   "lstat" "dis"
#>
#> [[2]]
#> [1] "lstat" "rm"    "nox"
#>
#> [[3]]
#> [1] "lstat" "crim"  "rm"
#>
#> [[4]]
#> [1] "ptratio" "lstat"   "tax"
```

Our first created zen-path object, `zpGw`, uses the `greedy.weighted` method and visits each edge at exactly once. The second zen-path uses the `strictly.weighted` method with `connect = FALSE`. `zpSw` consists of four unconnected paths. The zenplots for two of these paths are constructed below.

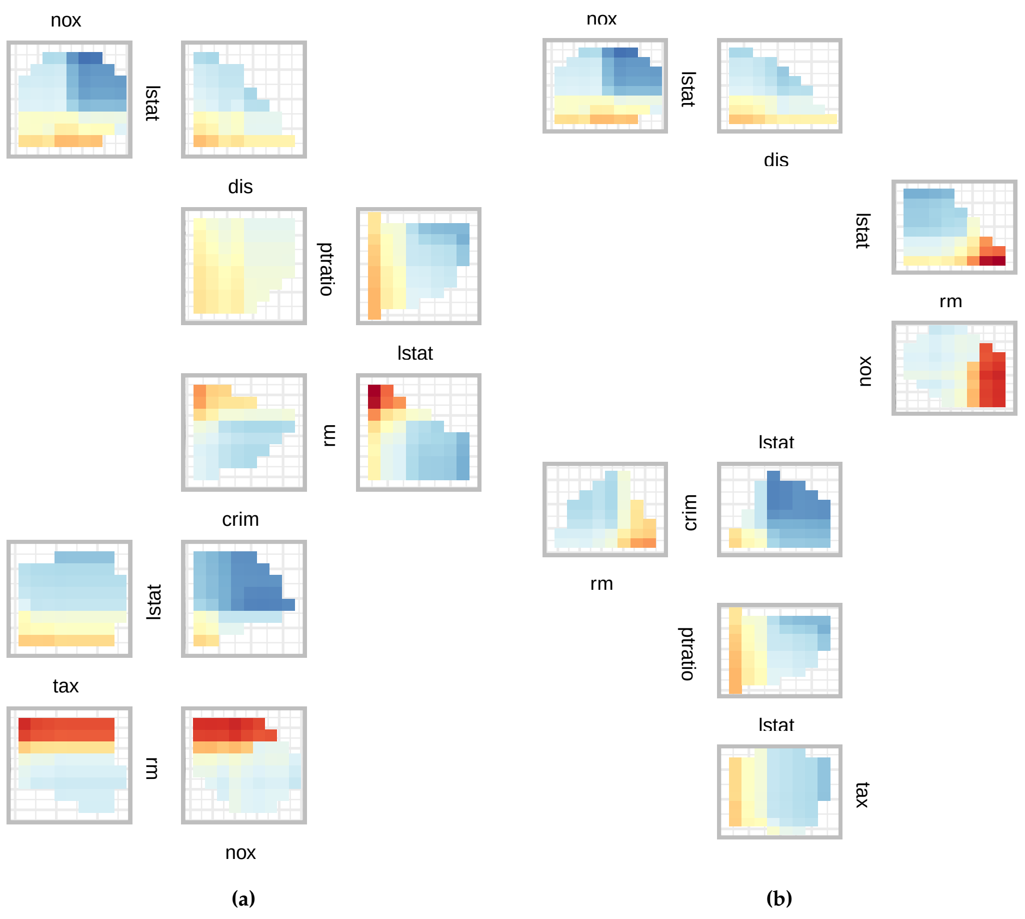

Figure 7: ZPDP for a GBM fit on the Boston data. In (a) the zpath is defined by the 'greedy.weighted' sorting method. In (b), the sorting method is defined by the 'strictly.weighted' method and is unconnected. For low values of *lstat* and and high values of *rm*, predicted median house price value increases.

```
pdpZen(data = Boston,
       fit = gbst,
       response = "medv",
       zpath = zpGw,
       convexHull = TRUE,
       predictFun = pFun)

pdpZen(data = Boston,
       fit = gbst,
       response = "medv",
       zpath = zpSw,
       convexHull = TRUE,
       predictFun = pFun)
```

Note that there are 7 different variables involved in high interactions, which could be displayed in a 7×7 GPDP, showing a total of 21 bivariate PDPs. But only 8 of these have VInt values above the 90% quantile, and Figure 7(b) using the `strictly.weighted` path shows just these bivariate PDPs compact layout. Using the `greedy.weighted` sorting method in (a) produces a smaller, neater plot but at the expense of including some plots that are not particularly interesting (for example the pair *dis* : *ptratio*).

Summary

We have presented a detailed exposition of our R package **vivid** which contains a suite of integrated functions implementing algorithms and novel visualizations for exploring variable importance and variable interactions in machine learning models. Our techniques are intuitive, adaptable, easy to customize and facilitate model comparison. When building the `vivid` matrix to use in our heatmap and network visualizations, VIVI metrics that are model specific or model-agnostic may be employed. For measuring interactions we currently only provide the option to use the agnostic Friedman's H-statistic. However, as outlined in the Calculating VIVI section, the inclusion of different VIVI measures is easily possible.

Our **vivid** package is a useful addition to the other packages in the area of model visualization, such as those discussed in the Introduction section. Our heatmap and network plots efficiently determine which variables have the greatest impact on the response. When coupled with the seriation, filtering, and clustering techniques, these visualizations enhance the interpretation of ML predictions. Our GPDP and ZPDP can be used to provide a thorough examination of the behavior of a fitted ML model by examining the individual variable effects and their pairwise interactions. These plots combine the bivariate PDP, ICE curves, and scatterplots of the raw variable values. They further allow focusing on subsets of variables with high VInt, and so allow us to efficiently explore a fitted ML model by focusing attention to only the most important aspects.

For future work, the inclusion of other model summaries could be incorporated into **vivid**, such as the interaction statistics described in Greenwell, Boehmke, and McCarthy (2018) or the use of accumulated local effects (ALE) of Apley and Zhu (2020). This latter method was created to address bias problems with partial dependency functions and could be used in place of the bivariate PDPs seen in both the GPDP and ZPDP. However the calculation of an agnostic, easily interpretable variable interaction measure that accounts for correlated variables remains an ongoing research goal.

References

Antunes, Nuno, Leandro Balby, Flavio Figueiredo, Nuno Lourenco, Wagner Meira, and Walter Santos. 2018. "Fairness and Transparency of Machine Learning for Trustworthy Cloud Services." In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-w)*, 188–93. IEEE.

Apley, Daniel W, and Jingyu Zhu. 2020. "Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 82 (4): 1059–86.

Biecek, Przemyslaw. 2018. "DALEX: Explainers for Complex Predictive Models in R." *Journal of Machine Learning Research* 19 (84): 1–5. https://jmlr.org/papers/v19/18-416.html.

Biecek, Przemyslaw, and Tomasz Burzykowski. 2021. *Explanatory Model Analysis: Explore, Explain and Examine Predictive Models*. Chapman; Hall/CRC.

Bischl, Bernd, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. 2016. "mlr: Machine Learning in R." *Journal of Machine Learning Research* 17 (170): 1–5. https://jmlr.org/papers/v17/15-066.html.

Breiman, Leo. 2001. "Random Forests." *Machine Learning* 45 (1): 5–32. https://doi.org/10.1023/A:1010933404324.

Chen, Xi, and Hemant Ishwaran. 2012. "Random Forests for Genomic Data Analysis." *Genomics* 99 (6): 323–29.

Chipman, Hugh A, Edward I George, and Robert E McCulloch. 2010. "BART: Bayesian Additive Regression Trees." *The Annals of Applied Statistics* 4 (1): 266–98.

Csardi, Gabor, and Tamas Nepusz. 2006. "The Igraph Software Package for Complex Network Research." *InterJournal* Complex Systems: 1695. https://igraph.org.

Earle, Denise, and Catherine Hurley. 2015. "Advances in Dendrogram Seriation for Application to Visualization." *Journal of Computational and Graphical Statistics* 24 (March). https://doi.org/10.1080/10618600.2013.874295.

Felzmann, Heike, Eduard Fosch Villaronga, Christoph Lutz, and Aurelia Tamò-Larrieux. 2019. "Transparency You Can Trust: Transparency Requirements for Artificial Intelligence Between Legal Norms and Contextual Concerns." *Big Data & Society* 6 (1): 2053951719860542.

Fisher, Aaron, Cynthia Rudin, and Francesca Dominici. 2019. "All Models Are Wrong, but Many Are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously." *J. Mach. Learn. Res.* 20 (177): 1–81.

Friedman, J. H. 2000. "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics* 29 (November). https://doi.org/10.1214/aos/1013203451.

Friedman, J. H., and B. E. Popescu. 2008. "Predictive Learning via Rule Ensembles." *The Annals of Applied Statistics.*, 916–54.

Goldstein, Alex, Adam Kapelner, Justin Bleich, and Emil Pitkin. 2015. "Peeking Inside the Black Box: Visualizing Statistical Learning with Plots of Individual Conditional Expectation." *Journal of Computational and Graphical Statistics* 24 (1): 44–65. https://doi.org/10.1080/10618600.2014.907095.

Grange, Stuart K, David C Carslaw, Alastair C Lewis, Eirini Boleti, and Christoph Hueglin. 2018. "Random Forest Meteorological Normalisation Models for Swiss PM 10 Trend Analysis." *Atmospheric Chemistry and Physics* 18 (9): 6223–39.

Greenwell, Brandon M. 2017. "pdp: An r Package for Constructing Partial Dependence Plots." *The R Journal* 9 (1): 421–36. https://journal.r-project.org/archive/2017/RJ-2017-016/index.html.

Greenwell, Brandon M., and Bradley C. Boehmke. 2020. "Variable Importance Plots—an Introduction to the vip Package." *The R Journal* 12 (1): 343–66. https://doi.org/10.32614/RJ-2020-013.

Greenwell, Brandon M., Bradley C. Boehmke, and Andrew J. McCarthy. 2018. "A Simple and Effective Model-Based Variable Importance Measure." *arXiv Preprint arXiv:1805.04755*.

Harrison Jr, David, and Daniel L Rubinfeld. 1978. "Hedonic Housing Prices and the Demand for Clean Air." *Journal of Environmental Economics and Management* 5 (1): 81–102.

Hastie, T., R. Tibshirani, and J. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* 2nd ed. Springer Series in Statistics. Springer-Verlag.

Hofert, Marius, and Wayne Oldford. 2020. "Zigzag Expanded Navigation Plots in R: The R Package zenplots." *Journal of Statistical Software* 95 (4): 1–44. https://doi.org/10.18637/jss.v095.i04.

Hurley, Catherine B., and R. W. Oldford. 2011. "Eulerian Tour Algorithms for Data Visualization and the PairViz Package." *Computational Statistics* 26 (December): 613–33. https://doi.org/10.1007/s00180-011-0229-5.

———. 2022. *PairViz: Visualization Using Graph Traversal*. https://CRAN.R-project.org/package=PairViz.

Hurley, Catherine, Mark OConnell, and Katarina Domijan. 2022. *Condvis2: Interactive Conditional Visualization for Supervised and Unsupervised Models in Shiny*.

Hvitfeldt, Emil, Thomas Lin Pedersen, and Michaël Benesty. 2022. *Lime: Local Interpretable Model-Agnostic Explanations*. https://CRAN.R-project.org/package=lime.

Inglis, Alan, Andrew Parnell, and Catherine B Hurley. 2022. "Visualizing Variable Importance and Variable Interaction Effects in Machine Learning Models." *Journal of Computational and Graphical Statistics* 31 (3): 766–78. https://doi.org/10.1080/10618600.2021.2007935.

Ishwaran, Hemant, Udaya B Kogalur, Eiran Z Gorodeski, Andy J Minn, and Michael S Lauer. 2010. "High-Dimensional Variable Selection for Survival Data." *Journal of the American Statistical Association* 105 (489): 205–17.

Kuhn, Max, and Hadley Wickham. 2020. *Tidymodels: A Collection of Packages for Modeling and Machine Learning Using Tidyverse Principles.* https://www.tidymodels.org.

Lang, Michel, Martin Binder, Jakob Richter, Patrick Schratz, Florian Pfisterer, Stefan Coors, Quay Au, Giuseppe Casalicchio, Lars Kotthoff, and Bernd Bischl. 2019. "mlr3: A Modern Object-Oriented Machine Learning Framework in R." *Journal of Open Source Software*, December. https://doi.org/10.21105/joss.01903.

Maksymiuk, Szymon, Ewelina Karbowiak, and Przemyslaw Biecek. 2021. *EIX: Explain Interactions in 'XGBoost'*. https://CRAN.R-project.org/package=EIX.

Meyer, David, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. 2021. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. https://CRAN.R-project.org/package=e1071.

Molnar, Christoph. 2022. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. 2nd ed. https://christophm.github.io/interpretable-ml-book.

Molnar, Christoph, Bernd Bischl, and Giuseppe Casalicchio. 2018. "Iml: An r Package for Interpretable Machine Learning." *JOSS* 3 (26): 786. https://doi.org/10.21105/joss.00786.

Murray, Kim, and Mary M Conner. 2009. "Methods to Quantify Variable Importance: Implications for the Analysis of Noisy Ecological Data." *Ecology* 90 (2): 348–55.

Paluszynska, Aleksandra, Przemyslaw Biecek, and Yue Jiang. 2020. *randomForestExplainer: Explaining and Visualizing Random Forests in Terms of Variable Importance*. https://CRAN.R-project.org/package=randomForestExplainer.

Probst, Philipp. 2020. *varImp: RF Variable Importance for Arbitrary Measures*. https://CRAN.R-project.org/package=varImp.

Schliep, Klaus, and Klaus Hechenbichler. 2016. *Kknn: Weighted k-Nearest Neighbors*. https://CRAN.R-project.org/package=kknn.

Venables, W. N., and B. D. Ripley. 2002. *Modern Applied Statistics with s*. Fourth. New York: Springer. https://www.stats.ox.ac.uk/pub/MASS4/.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. https://ggplot2.tidyverse.org.

Wright, Marvin N., and Andreas Ziegler. 2017. "ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R." *Journal of Statistical Software* 77 (1): 1–17. https://doi.org/10.18637/jss.v077.i01.

Zeileis, Achim, Jason C. Fisher, Kurt Hornik, Ross Ihaka, Claire D. McWhite, Paul Murrell, Reto Stauffer, and Claus O. Wilke. 2020. "Colorspace: A Toolbox for Manipulating and Assessing Colors and Palettes." *Journal of Statistical Software, Articles* 96 (1): 1–49. https://doi.org/10.18637/jss.

v096.i01.

Alan Inglis
Maynooth University
Department of Mathematics and Statistics
Maynooth, Ireland
ORCiD: 0000-0002-1151-6657
alan.inglis@mu.ie

Andrew Parnell
Maynooth University
Hamilton Institute
Maynooth, Ireland
ORCiD: 0000-0001-7956-7939
andrew.parnell@mu.ie

Catherine Hurley
Maynooth University
Department of Mathematics and Statistics
Maynooth, Ireland
ORCiD: 0000-0003-2758-5531
catherine.hurley@mu.ie