Exploring Learner Predictions

Table of Contents

Source: `vignettes/tutorial/partial_dependence.Rmd` (https://github.com/mlr-org/mlr/blob/master/vignettes/tutorial/partial_dependence.Rmd)

Learners use features to learn a prediction function and make predictions, but the effect of those features is often not apparent. `mlr` can estimate the partial dependence of a learned function on a subset of the feature space using `generatePartialDependenceData()` (`../../reference/generatePartialDependenceData.html`).

Partial dependence plots reduce the potentially high dimensional function estimated by the learner, and display a marginalized version of this function in a lower dimensional space. For example suppose $Y = f(X) + \epsilon$, where $\mathbb{E}[\epsilon|X] = 0$. With (X, Y) pairs drawn independently from this statistical model, a learner may estimate $\hat{f}$, which, if X is high dimensional, can be uninterpretable. Suppose we want to approximate the relationship between some subset of X. We partition X into two sets, X_s and X_c such that $X = X_s \cup X_c$, where X_s is a subset of X of interest.

The partial dependence of f on X_s is

$$f_{X_s} = \mathbb{E}_{X_c} f(X_s, X_c).$$

X_c is integrated out. We use the following estimator:

$$\hat{f}_{X_s} = \frac{1}{N} \sum_{i=1}^{N} \hat{f}(X_s, x_{ic}).$$

The individual conditional expectation of an observation can also be estimated using the above algorithm absent the averaging, giving $\hat{f}^{(i)}_{X_s}$. This allows the discovery of features of $\hat{f}$ that may be obscured by an aggregated summary of $\hat{f}$.

The partial derivative of the partial dependence function, $\frac{\partial \hat{f}_{X_s}}{\partial X_s}$, and the individual conditional expectation function, $\frac{\partial \hat{f}^{(i)}_{X_s}}{\partial X_s}$, can also be computed. For regression and survival tasks the partial derivative of a single feature X_s is the gradient of the partial dependence function, and for classification tasks where the learner can output class probabilities the Jacobian. Note that if the learner produces discontinuous partial dependence (e.g., piecewise constant functions such as decision trees, ensembles of decision trees, etc.) the derivative will be 0 (where the function is not changing) or trending towards positive or negative infinity (at the discontinuities where the derivative is undefined). Plotting the partial dependence function of such learners may give the impression that the function is not discontinuous because the prediction grid is not composed of all discontinuous points in the predictor space. This results in a line interpolating that makes the function appear to be piecewise linear (where the derivative would be defined except at the boundaries of each piece).

The partial derivative can be informative regarding the additivity of the learned function in certain features. If $\hat{f}^{(i)}_{X_s}$ is an additive function in a feature X_s, then its partial derivative will not depend on any other features (X_c) that may have been used by the learner. Variation in the estimated partial derivative indicates that there is a region of interaction between X_s and X_c in $\hat{f}$. Similarly, instead of using the mean to estimate the expected value of the function at different values of X_s, instead computing the variance can highlight regions of interaction between X_s and X_c.

See Goldstein, Kapelner, Bleich, and Pitkin (2014) (http://arxiv.org/abs/1309.6392) for more details and their package `ICEbox` for the original implementation. The algorithm works for any supervised learner with classification, regression, and survival tasks.

Generating partial dependences

Our implementation, following `mlr`'s visualization (visualization.html) pattern, consists of the above mentioned function `generatePartialDependenceData()` (`../../reference/generatePartialDependenceData.html`), as well as two visualization functions, `plotPartialDependence()` (`../../reference/plotPartialDependence.html`) and `plotPartialDependenceGGVIS()`. The former generates input (objects of class `PartialDependenceData()` (`../../reference/generatePartialDependenceData.html`)) for the latter.

The first step executed by `generatePartialDependenceData()` (`../../reference/generatePartialDependenceData.html`) is to generate a feature grid for every element of the character vector `features` passed. The data are given by the `input` argument, which can be a `Task()` (`../../reference/Task.html`) or a `data.frame`. The feature grid can be generated in several ways. A uniformly spaced grid of length `gridsize` (default 10) from the empirical minimum to the empirical maximum is created by default, but arguments `fmin` and `fmax` may be used to override the empirical default (the lengths of `fmin` and `fmax` must match the length of `features`). Alternatively the feature data can be resampled, either by using a bootstrap or by subsampling.

```
lrn.classif = makeLearner (../../reference/makeLearner.html)("classif.ksvm", pred
fit.classif = train (../../reference/train.html)(lrn.classif, iris.task)
pd = generatePartialDependenceData (../../reference/generatePartialDependenceData
## Loading required package: mmpf
pd
## PartialDependenceData
## Task: iris-example
## Features: Petal.Width
## Target: Petal.Width
## Derivative: FALSE
## Interaction: FALSE
## Individual: FALSE
##     Class Probability Petal.Width
## 1: setosa   0.4912530   0.1000000
## 2: setosa   0.4408135   0.3666667
## 3: setosa   0.3786951   0.6333333
## 4: setosa   0.3199833   0.9000000
## 5: setosa   0.2494105   1.1666667
## 6: setosa   0.1770095   1.4333333
## ... (#rows: 30, #cols: 3)
```

As noted above, X_s does not have to be unidimensional. If it is not, the `interaction` flag must be set to `TRUE`. Then the individual feature grids are combined using the Cartesian product, and the estimator above is applied, producing the partial dependence for every combination of unique feature values. If the `interaction` flag is `FALSE` (the default) then by default X_s is assumed unidimensional, and partial dependencies are generated for each feature separately. The resulting output when `interaction = FALSE` has a column for each feature, and `NA` where the feature was not used.

```
pd.lst = generatePartialDependenceData (../../reference/generatePartialDependence
head(pd.lst$data)
##     Class Probability Petal.Width Petal.Length
## 1: setosa   0.4912530   0.1000000           NA
## 2: setosa   0.4408135   0.3666667           NA
## 3: setosa   0.3786951   0.6333333           NA
## 4: setosa   0.3199833   0.9000000           NA
## 5: setosa   0.2494105   1.1666667           NA
## 6: setosa   0.1770095   1.4333333           NA

tail(pd.lst$data)
##        Class Probability Petal.Width Petal.Length
## 1: virginica   0.1983692          NA     3.622222
## 2: virginica   0.3046362          NA     4.277778
## 3: virginica   0.4301455          NA     4.933333
## 4: virginica   0.5796330          NA     5.588889
## 5: virginica   0.6853425          NA     6.244444
## 6: virginica   0.7009223          NA     6.900000
```

```
pd.int = generatePartialDependenceData (../../reference/generatePartialDependence
pd.int
## PartialDependenceData
## Task: iris-example
## Features: Petal.Width, Petal.Length
## Target: Petal.Width, Petal.Length
## Derivative: FALSE
## Interaction: TRUE
## Individual: FALSE
##     Class Probability Petal.Width Petal.Length
## 1: setosa   0.6691406         0.1     1.000000
## 2: setosa   0.6640290         0.1     1.655556
## 3: setosa   0.6218785         0.1     2.311111
## 4: setosa   0.5372541         0.1     2.966667
## 5: setosa   0.4218425         0.1     3.622222
## 6: setosa   0.3224443         0.1     4.277778
## ... (#rows: 300, #cols: 4)
```

At each step in the estimation of $\hat{f}_{X_s}$ a set of predictions of length N is generated. By default the mean prediction is used. For classification where `predict.type = "prob"` this entails the mean class probabilities. However, other summaries of the predictions may be used. For regression and survival tasks the function used here must either return one number or three, and, if the latter, the numbers must be sorted lowest to highest. For classification tasks the function must return a number for each level of the target feature.

As noted, the `fun` argument can be a function which returns three numbers (sorted low to high) for a regression task. This allows further exploration of relative feature importance. If a feature is relatively important, the bounds are necessarily tighter because the feature accounts for more of the variance of the predictions, i.e., it is “used” more by the learner. More directly setting `fun = var` identifies regions of interaction between X_s and X_c.

```
lrn.regr = makeLearner (../../reference/makeLearner.html)("regr.ksvm")
fit.regr = train (../../reference/train.html)(lrn.regr, bh.task)
pd.regr = generatePartialDependenceData (../../reference/generatePartialDependen
pd.regr
## PartialDependenceData
## Task: BostonHousing-example
## Features: lstat
## Target: lstat
## Derivative: FALSE
## Interaction: FALSE
## Individual: FALSE
##        medv     lstat
## 1: 24.98185  1.730000
## 2: 23.75768  5.756667
## 3: 22.37005  9.783333
## 4: 20.71224 13.810000
## 5: 19.55165 17.836667
## 6: 18.97297 21.863333
##  ... (#rows: 10, #cols: 2)
```

```
pd.ci = generatePartialDependenceData (../../reference/generatePartialDependence
  fun = function(x) quantile(x, c(.25, .5, .75)))
pd.ci
## PartialDependenceData
## Task: BostonHousing-example
## Features: lstat
## Target: lstat
## Derivative: FALSE
## Interaction: FALSE
## Individual: FALSE
##        medv Function     lstat
## 1: 21.37156 medv.25%  1.730000
## 2: 20.76052 medv.25%  5.756667
## 3: 19.88281 medv.25%  9.783333
## 4: 18.68150 medv.25% 13.810000
## 5: 16.51643 medv.25% 17.836667
## 6: 14.76512 medv.25% 21.863333
##  ... (#rows: 30, #cols: 3)
```

```
pd.classif = generatePartialDependenceData (../../reference/generatePartialDepen
pd.classif
## PartialDependenceData
## Task: iris-example
## Features: Petal.Length
## Target: Petal.Length
## Derivative: FALSE
## Interaction: FALSE
## Individual: FALSE
##     Class Probability Petal.Length
## 1: setosa  0.29470748     1.000000
## 2: setosa  0.23653080     1.655556
## 3: setosa  0.16849333     2.311111
## 4: setosa  0.09503590     2.966667
## 5: setosa  0.04712780     3.622222
## 6: setosa  0.02567515     4.277778
##  ... (#rows: 30, #cols: 3)
```

In addition to bounds based on a summary of the distribution of the conditional expectation of each observation, learners which can estimate the variance of their predictions can also be used. The argument `bounds` is a numeric vector of length two which is added (so the first number should be negative) to the point prediction to produce a confidence interval for the partial dependence. The default is the .025 and .975 quantiles of the Gaussian distribution.

```
fit.se = train (../../reference/train.html)(makeLearner (../../reference/makeLea
pd.se = generatePartialDependenceData (../../reference/generatePartialDependence
head(pd.se$data)
##       lower     medv    upper     lstat crim
## 1: 12.62367 31.34372 50.06376  1.730000   NA
## 2: 14.26151 26.21193 38.16234  5.756667   NA
## 3: 13.48184 23.52293 33.56403  9.783333   NA
## 4: 14.34106 22.12538 29.90971 13.810000   NA
## 5: 12.89603 20.42116 27.94629 17.836667   NA
## 6: 11.81290 19.75491 27.69692 21.863333   NA

tail(pd.se$data)
##       lower     medv    upper lstat     crim
## 1: 10.81796 22.04066 33.26335    NA 39.54849
## 2: 10.73350 22.01862 33.30373    NA 49.43403
## 3: 10.69249 22.00300 33.31351    NA 59.31957
## 4: 10.69576 22.00448 33.31321    NA 69.20512
## 5: 10.71337 22.00945 33.30553    NA 79.09066
## 6: 10.71372 22.00971 33.30569    NA 88.97620
```

As previously mentioned if the aggregation function is not used, i.e., it is the identity, then the conditional expectation of $\hat{f}^{(i)}_{X_s}$ is estimated. If `individual = TRUE` then `generatePartialDependenceData()`

(../../reference/generatePartialDependenceData.html) returns n partial dependence estimates made at each point in the prediction grid constructed from the features.

```
pd.ind.regr = generatePartialDependenceData (../../reference/generatePartialDeper
pd.ind.regr
## PartialDependenceData
## Task: BostonHousing-example
## Features: lstat
## Target: lstat
## Derivative: FALSE
## Interaction: FALSE
## Individual: TRUE
##        medv n     lstat
## 1: 20.42031 1  1.730000
## 2: 20.62861 1  5.756667
## 3: 21.08401 1  9.783333
## 4: 21.67691 1 13.810000
## 5: 22.25122 1 17.836667
## 6: 22.66261 1 21.863333
##  ... (#rows: 5060, #cols: 3)
```

The resulting output, particularly the element `data` in the returned object, has an additional column `idx` which gives the index of the observation to which the row pertains.

For classification tasks this index references both the class and the observation index.

```
pd.ind.classif = generatePartialDependenceData (../../reference/generatePartialDe
pd.ind.classif
## PartialDependenceData
## Task: iris-example
## Features: Petal.Length
## Target: Petal.Length
## Derivative: FALSE
## Interaction: FALSE
## Individual: TRUE
##     Class Probability n Petal.Length
## 1: setosa   0.9562404 1     1.000000
## 2: setosa   0.9630777 1     1.655556
## 3: setosa   0.9423127 1     2.311111
## 4: setosa   0.8676090 1     2.966667
## 5: setosa   0.7040386 1     3.622222
## 6: setosa   0.5097881 1     4.277778
##  ... (#rows: 4500, #cols: 4)
```

Partial derivatives can also be computed for individual partial dependence estimates and aggregate partial dependence. This is restricted to a single feature at a time. The derivatives of individual partial dependence estimates can be useful in finding regions of interaction between the feature for which the derivative is estimated and the features excluded.

```
pd.regr.der = generatePartialDependenceData (../../reference/generatePartialDepen
head(pd.regr.der$data)
##           medv     lstat
## 1: -0.2554174  1.730000
## 2: -0.3555116  5.756667
## 3: -0.4123333  9.783333
## 4: -0.4185369 13.810000
## 5: -0.3762343 17.836667
## 6: -0.2948220 21.863333
```

```
pd.regr.der.ind = generatePartialDependenceData (../../reference/generatePartialD
  individual = TRUE)
head(pd.regr.der.ind$data)
##           medv   n     lstat
## 1: -0.5831248 379  1.730000
## 2: -0.7896842 379  5.756667
## 3: -0.8955349 379  9.783333
## 4: -0.8738373 379 13.810000
## 5: -0.7285446 379 17.836667
## 6: -0.4916859 379 21.863333
```

```
pd.classif.der = generatePartialDependenceData (../../reference/generatePartialDe
head(pd.classif.der$data)
##     Class Probability Petal.Width
## 1: setosa  -0.1333332   0.1000000
## 2: setosa  -0.2304429   0.3666667
## 3: setosa  -0.2227234   0.6333333
## 4: setosa  -0.2331371   0.9000000
## 5: setosa  -0.2888623   1.1666667
## 6: setosa  -0.2325219   1.4333333
```

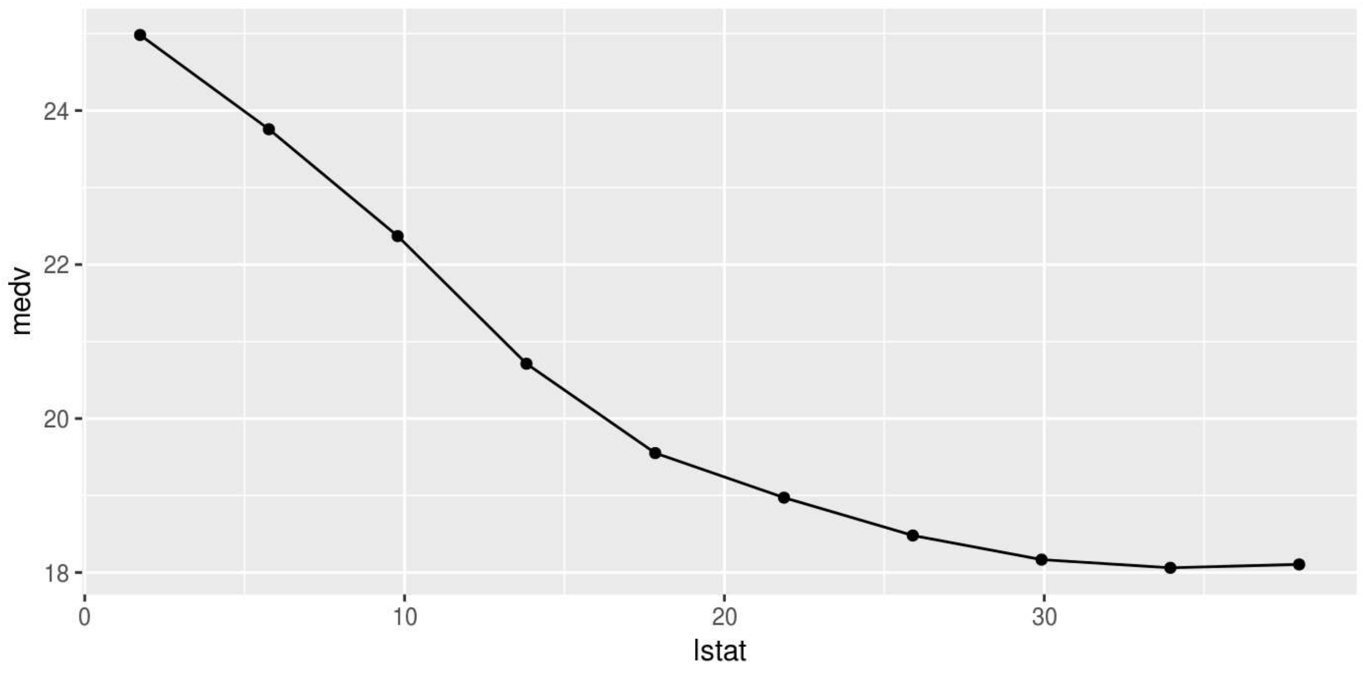
```
pd.classif.der.ind = generatePartialDependenceData (../../reference/generatePart
  individual = TRUE)
head(pd.classif.der.ind$data)
##     Class Probability  n Petal.Width
## 1: setosa  0.04558373 32   0.1000000
## 2: setosa -0.03449532 32   0.3666667
## 3: setosa -0.16621555 32   0.6333333
## 4: setosa -0.46694229 32   0.9000000
## 5: setosa -0.78820973 32   1.1666667
## 6: setosa -0.67724741 32   1.4333333
```

Plotting partial dependences

Results from `generatePartialDependenceData()` `(../../reference/generatePartialDependenceData.html)` and `generateFunctionalANOVAData()` can be visualized with `plotPartialDependence()` `(../../reference/plotPartialDependence.html)` and `plotPartialDependenceGGVIS()`.

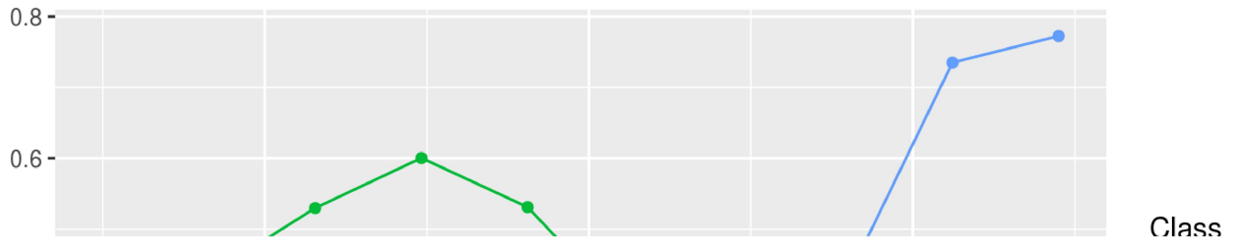
With one feature and a regression task the output is a line plot, with a point for each point in the corresponding feature's grid.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.regr)
```



With a classification task, a line is drawn for each class, which gives the estimated partial probability of that class for a particular point in the feature grid.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.classif)
```



For regression tasks, when the `fun` argument of `generatePartialDependenceData()` `(../../reference/generatePartialDependenceData.html)` is used, the bounds will automatically be displayed using a gray ribbon.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.ci)
```

The same goes for plots of partial dependences where the learner has `predict.type = "se"`.

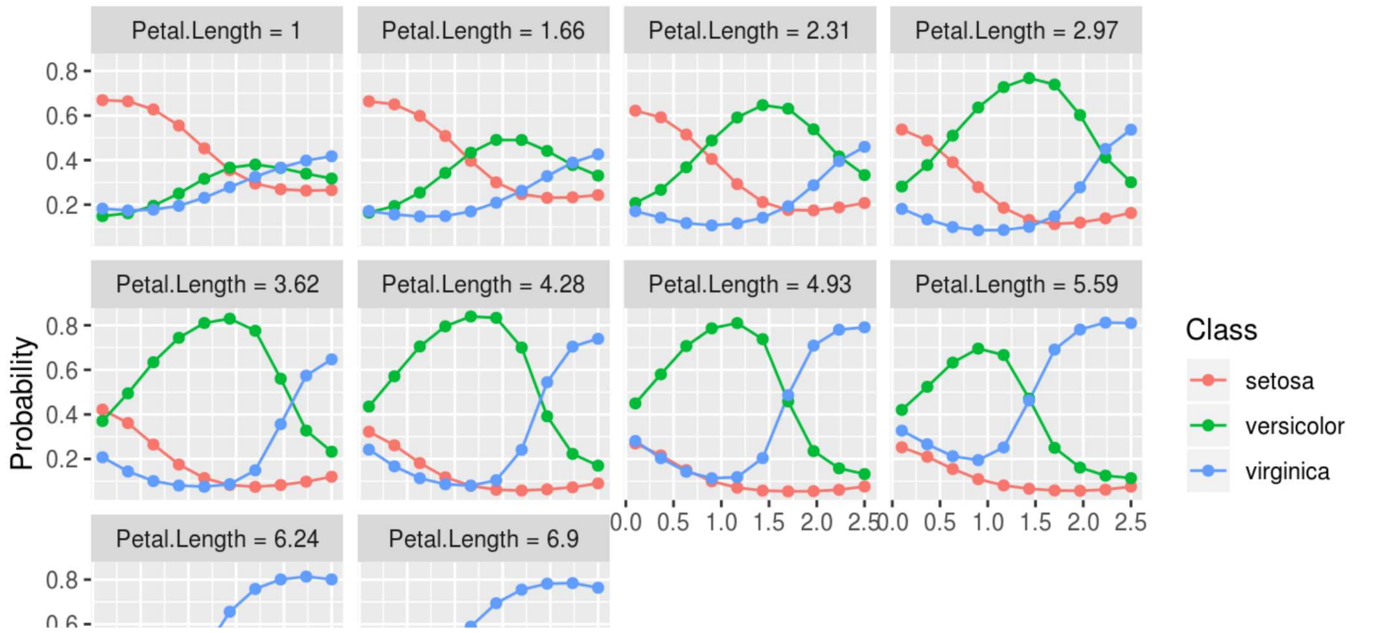
```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.se)
```

When multiple features are passed to `generatePartialDependenceData()` `(../../reference/generatePartialDependenceData.html)` but `interaction = FALSE`, facetting is used to display each estimated bivariate relationship.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.lst)
```

When `interaction = TRUE` in the call to `generatePartialDependenceData()` `(../../reference/generatePartialDependenceData.html)`, one variable must be chosen to be used for facetting, and a subplot for each value in the chosen feature's grid is created, wherein the other feature's partial dependences within the facetting feature's value are shown. Note that this type of plot is limited to two features.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.int, facet
```



`plotPartialDependenceGGVIS()` can be used similarly, however, since `ggvis` currently lacks subplotting/facetting capabilities, the argument `interact` maps one feature to an interactive sidebar where the user can select a value of one feature.

```
plotPartialDependenceGGVIS(pd.int, interact = "Petal.Length")
```

When `individual = TRUE` each individual conditional expectation curve is plotted.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.ind.regr)
```

Plotting partial derivative functions works the same as partial dependence. Below are estimates of the derivative of the mean aggregated partial dependence function, and the individual partial dependence functions for a regression and a classification task respectively.

```
plotPartialDependence (../../reference/plotPartialDependence.html)(pd.regr.der)
```