Package ‘BayesTree’

July 6, 2016

Title Bayesian Additive Regression Trees
Version 0.3-1.4
Date 2016-2-21
Author Hugh Chipman <hugh.chipman@gmail.com>, Robert McCulloch
       <robert.e.mcculloch@gmail.com>
Description This is an implementation of BART:Bayesian Additive Regression Trees,
       by Chipman, George, McCulloch (2010).
Maintainer Robert McCulloch <robert.e.mcculloch@gmail.com>
Imports nnet
License GPL (>= 2)
Repository CRAN
Date/Publication 2016-07-06 10:26:13
NeedsCompilation yes

R topics documented:

    bart . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
    makeind . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
    pdbart . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

Index

bart

Bayesian Additive Regression Trees
Description

BART is a Bayesian “sum-of-trees” model. For numeric response \( y \), we have \( y = f(x) + \epsilon \), where \( \epsilon \sim N(0, \sigma^2) \). For a binary response \( y \), \( P(Y = 1|x) = F(f(x)) \), where \( F \) denotes the standard normal cdf (probit link).

In both cases, \( f \) is the sum of many tree models. The goal is to have very flexible inference for the unknown function \( f \).

In the spirit of “ensemble models”, each tree is constrained by a prior to be a weak learner so that it contributes a small amount to the overall fit.

Usage

```r
bart(
    x.train, y.train, x.test=matrix(0,0,0),
    sigest=NA, sigdf=3, sigquant=.90,
    k=2.0,
    power=2.0, base=.95,
    binaryOffset=0,
    ntree=200,
    ndpost=1000, nskip=100,
    printevery=100, keepevery=1, keeptrainfits=TRUE,
    usequants=FALSE, numcut=100, printcutoffs=0,
    verbose=TRUE)
```

# S3 method for class ‘bart’
plot(
    x,
    plquants=c(.05,.95), cols =c('blue','black'),
    ...)  
```

Arguments

- **x.train** Explanatory variables for training (in sample) data.
  May be a matrix or a data frame, with (as usual) rows corresponding to observations and columns to variables.
  If a variable is a factor in a data frame, it is replaced with dummies. Note that q dummies are created if q>2 and one dummy is created if q=2, where q is the number of levels of the factor. `makeind` is used to generate the dummies. `bart` will generate draws of \( f(x) \) for each \( x \) which is a row of `x.train`.

- **y.train** Dependent variable for training (in sample) data.
  If \( y \) is numeric a continous response model is fit (normal errors).
  If \( y \) is a factor (or just has values 0 and 1) then a binary response model with a probit link is fit.

- **x.test** Explanatory variables for test (out of sample) data.
  Should have same structure as `x.train`.
  `bart` will generate draws of \( f(x) \) for each \( x \) which is a row of `x.test`.

- **sigest** The prior for the error variance (\( \sigma^2 \)) is inverted chi-squared (the standard conditionally conjugate prior). The prior is specified by choosing the degrees of
freedom, a rough estimate of the corresponding standard deviation and a quantile to put this rough estimate at. If sigest=NA then the rough estimate will be the usual least squares estimator. Otherwise the supplied value will be used. Not used if y is binary.

**sigdf**
 Degrees of freedom for error variance prior. Not used if y is binary.

**sigquant**
The quantile of the prior that the rough estimate (see sigest) is placed at. The closer the quantile is to 1, the more aggressive the fit will be as you are putting more prior weight on error standard deviations ($\sigma$) less than the rough estimate. Not used if y is binary.

**k**
For numeric y, k is the number of prior standard deviations $E(Y|x) = f(x)$ is away from +/-5. The response (y.train) is internally scaled to range from -.5 to .5. For binary y, k is the number of prior standard deviations $f(x)$ is away from +/-3. In both cases, the bigger k is, the more conservative the fitting will be.

**power**
Power parameter for tree prior.

**base**
Base parameter for tree prior.

**binaryoffset**
Used for binary y. The model is $P(Y = 1|x) = F(f(x) + binaryOffset)$. The idea is that f is shrunk towards 0, so the offset allows you to shrink towards a probability other than .5.

**ntree**
The number of trees in the sum.

**ndpost**
The number of posterior draws after burn in, ndpost/keepevery will actually be returned.

**nskip**
Number of MCMC iterations to be treated as burn in.

**printevery**
As the MCMC runs, a message is printed every printevery draws.

**keepevery**
Every keepevery draw is kept to be returned to the user. A “draw” will consist of values of the error standard deviation ($\sigma$) and $f^\ast(x)$ at $x = \text{rows from the train(optionally) and test data}$, where $f^\ast$ denotes the current draw of $f$.

**keeptrainfits**
If true the draws of $f(x)$ for $x = \text{rows of x.train}$ are returned.

**usequants**
Decision rules in the tree are of the form $x \leq c$ vs. $x > c$ for each variable corresponding to a column of x.train. usequants determines how the set of possible $c$ is determined. If usequants is true, then the $c$ are a subset of the values $(xs[i]+xs[i+1])/2$ where xs is unique sorted values obtained from the corresponding column of x.train. If usequants is false, the cutoffs are equally spaced across the range of values taken on by the corresponding column of x.train.

**numcut**
The number of possible values of $c$ (see usequants). If a single number if given, this is used for all variables. Otherwise a vector with length equal to ncol(x.train) is required, where the $i^{th}$ element gives the number of $c$ used for the $i^{th}$ variable in x.train. If usequants is false, numcut equally spaced cutoffs are used covering the range of values in the corresponding column of x.train. If usequants is true, then min(numcut, the number of unique values in the corresponding columns of x.train - 1) $c$ values are used.

**printcutoffs**
The number of cutoff rules $c$ to printed to screen before the MCMC is run. Give a single integer, the same value will be used for all variables. If 0, nothing is printed.
verbose Logical, if FALSE supress printing.

x Value returned by bart which contains the information to be plotted.

plquants In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median and a lower and upper quantile. plquants is a double vector of length two giving the lower and upper quantiles.

cols Vector of two colors. First color is used to plot the median of $f(x)$ and the second color is used to plot the lower and upper quantiles.

... Additional arguments passed on to plot.

Details

BART is a Bayesian MCMC method. At each MCMC iteration, we produce a draw from the joint posterior $(f, \sigma | (x, y))$ in the numeric $y$ case and just $f$ in the binary $y$ case.

Thus, unlike a lot of other modelling methods in R, we do not produce a single model object from which fits and summaries may be extracted. The output consists of values $f^*(x)$ (and $\sigma^*$ in the numeric case) where * denotes a particular draw. The $x$ is either a row from the training data (x.train) or the test data (x.test).

Value

The plot method sets mfrow to c(1,2) and makes two plots.
The first plot is the sequence of kept draws of $\sigma$ including the burn-in draws. Initially these draws will decline as BART finds fit and then level off when the MCMC has burnt in.
The second plot has $y$ on the horizontal axis and posterior intervals for the corresponding $f(x)$ on the vertical axis.

bart returns a list assigned class ‘bart’. In the numeric $y$ case, the list has components:

yhat.train A matrix with (ndpost/keepevery) rows and nrow(x.train) columns. Each row corresponds to a draw $f^*$ from the posterior of $f$ and each column corresponds to a row of x.train. The $(i, j)$ value is $f^*(x)$ for the $i^{th}$ kept draw of $f$ and the $j^{th}$ row of x.train. Burn-in is dropped.

yhat.test Same as yhat.train but now the x’s are the rows of the test data.

yhat.train.mean train data fits = mean of yhat.train columns.

yhat.test.mean test data fits = mean of yhat.test columns.

sigma post burn in draws of sigma, length = ndpost/keepevery.

first.sigma burn-in draws of sigma.

varcount a matrix with (ndpost/keepevery) rows and nrow(x.train) columns. Each row is for a draw. For each variable (corresponding to the columns), the total count of the number of times that variable is used in a tree decision rule (over all trees) is given.

sigest The rough error standard deviation ($\sigma$) used in the prior.

y The input dependent vector of values for the dependent variable. This is used in plot.bart.
In the binary y case, the returned list has the components yhat.train, yhat.test, and varcount as above. In addition the list has a binaryOffset component giving the value used. Note that in the binary y, case yhat.train and yhat.test are \( f(x) + \text{binaryOffset} \). If you want draws of the probability \( P(Y = 1|x) \) you need to apply the normal cdf (\text{pnorm}) to these values.

**Note**

There was a bug in BayesTree\_0.1-0 (now fixed of course). If the number of test observations was less than the number of trees (200 is the default), the yhat.test and yhat.test.mean components were suspect.

**Author(s)**

Hugh Chipman: <hugh.chipman@gmail.com>
Robert McCulloch: <robert.e.mcculloch@gmail.com>.

**References**


**See Also**

\text{pdbart}

**Examples**

```r
##simulate data (example from Friedman MARS paper)
f = function(x){
10*sin(pi*x[,1]*x[,2]) + 20*(x[,3]<-.5)^2+10*x[,4]*x[,5]
}  
sigma = 1.0  # y = f(x) + sigma*z, z~N(0,1)
n = 100  #number of observations
set.seed(99)
x=matrix(runif(n=10),n,10) #10 variables, only first 5 matter
Ey = f(x)
y=Ey+sigma*rnorm(n)
lmFit = lm(y~,data.frame(x,y))  #compare lm fit to BART later
#run BART
set.seed(99)
bartFit = bart(x,y,ndpost=200)  #default is ndpost=1000, this is to run example fast.
plot(bartFit)  # plot bart fit
#compare BART fit to linear matter and truth = Ey
fitmat = cbind(y,Ey,lmFit$fitted,bartFit$yhat.train.mean)
colnames(fitmat) = c('y','Ey','lm','bart')
print(cor(fitmat))
```
**makeind**

*Build x matrix from x data frame (convert factors to dummies)*

**Description**

Converts factors to dummies.

Note that with more than one level, BART needs a dummy for each level of a factor (unlike in linear regression where one of the dummies is dropped).

**Usage**

```r
makeind(
  x,
  all=TRUE)
```

**Arguments**

- `x` Data frame of explanatory variables.
- `all` If all=TRUE, a factor with p levels will be replaced by all p dummies. If all=FALSE, the pth dummy is dropped.

**Details**

Uses function class.ind from the nnet library. Note that if you have train and test data frames, it may be best to rbind the two together, apply makeind to the result, and then pull them back apart.

**Value**

A matrix.

Numerical variables come first, and then the appended dummies.

**Author(s)**

Hugh Chipman: <hugh.chipman@acadiau.ca>
Robert McCulloch: <robert.mcculloch@chicagogs.edu>.

**See Also**

`bart`

**Examples**

```r
x1 = 1:10
x2 = as.factor(c(rep(1,5),rep(2,5)))
x3 = as.factor(c(1,1,1,2,2,4,4,4,4))
levels(x3) = c('rob','hugh','ed')
x = data.frame(x1,x2,x3)

junk = makeind(x)
```
Description

Run \texttt{bart} at test observations constructed so that a plot can be created displaying the effect of a single variable (\texttt{pdbart}) or pair of variables (\texttt{pdRbart}). Note the \(y\) is a binary with \(P(Y = 1|x) = F(f(x))\) with \(F\) the standard normal cdf, then the plots are all on the \(f\) scale.

Usage

\begin{verbatim}
pdbart(xNtrainL yNtrainL xindL levsL levquantsL plL plquantsL N)
## S3 method for class 'pdbart'
plot(xL xindL plquantsL colsL N)
pdRbart(xNtrainL yNtrainL xindL levsL levquantsL plL plquantsL N)
## S3 method for class 'pdRbart'
plot(xL plquantsL contour.colorL justmedianL N)
\end{verbatim}

Arguments

- \texttt{x.train} \\
  Explanatory variables for training (in sample) data.
  Must be a matrix (typeof double) with (as usual) rows corresponding to observations and columns to variables.
  Note that for a categorical variable you need to use dummies and if there are more than two categories, you need to put all the dummies in (unlike linear regression).

- \texttt{y.train} \\
  Dependent variable for training (in sample) data.
  Must be a vector (typeof double) with length equal to the number of observations (equal to the number of rows of \texttt{x.train}).

- \texttt{xind} \\
  Integer vector indicating which variables are to be plotted.
  In \texttt{pdbart}, variables (columns of \texttt{x.train}) for which plot is to be constructed.
  In \texttt{plot.pdbart}, indices in list returned by \texttt{pdbart} for which plot is to be constructed.
  In \texttt{pdRbart}, integer vector of length 2, indicating the pair of variables (columns of \texttt{x.train}) to plot.
levs  
Gives the values of a variable at which the plot is to be constructed.  
List, where $i^{th}$ component gives the values for $i^{th}$ variable.  
In pdbart, should have same length as xind.  
In pd2bart, should have length 2.  
See also argument levquants.

levquants  
If levs in NULL, the values of each variable used in the plot is set to the quantiles  
(in x.train) indicated by levquants.  
Double vector.

pl  
For pdbart and pd2bart, if true, plot is made (by calling plot.*).

plquants  
In the plots, beliefs about $f(x)$ are indicated by plotting the posterior median  
and a lower and upper quantile. plquants is a double vector of length two giving  
the lower and upper quantiles.

...  
Additional arguments.  
In pdbart,pd2bart, passed on to bart.  
In plot.pdbart, passed on to plot.  
In plot.pd2bart, passed on to image.

x  
For plot.*, object returned from pdbart or pd2bart.

cols  
Vector of two colors.  
First color is for median of $f$, second color is for the upper and lower quantiles.

contour.color  
Color for contours plotted on top of the image.

justmedian  
Boolean, if true just one plot is created for the median of $f(x)$ draws. If false,  
three plots are created one for the median and two additional ones for the lower  
and upper quantiles. In this case, mfrow is set to c(1,3).

Details

We divide the predictor vector $x$ into a subgroup of interest, $x_s$ and the complement $x_c = x \setminus x_s$.  
A prediction $f(x)$ can then be written as $f(x_s, x_c)$. To estimate the effect of $x_s$ on the prediction,  
Friedman suggests the partial dependence function  
$$f_s(x_s) = \frac{1}{n} \sum_{i=1}^{n} f(x_s, x_{ic})$$  
where $x_{ic}$ is the $i^{th}$ observation of $x_c$ in the data. Note that $(x_s, x_{ic})$ will generally not be one of the  
observed data points. Using BART it is straightforward to then estimate and even obtain uncertainty  
bounds for $f_s(x_s)$. A draw of $f^*_s(x_s)$ from the induced BART posterior on $f_s(x_s)$ is obtained by  
simply computing $f^*_s(x_s)$ as a byproduct of each MCMC draw $f^*$. The median (or average) of  
these MCMC draws $f^*_s(x_s)$ then yields an estimate of $f_s(x_s)$, and lower and upper quantiles can  
be used to obtain intervals for $f_s(x_s)$.

In pdbart $x_s$ consists of a single variable in $x$ and in pd2bart it is a pair of variables.  
This is a computationally intensive procedure. For example, in pdbart, to compute the partial  
dependence plot for 5 $x_s$ values, we need to compute $f(x_s, x_c)$ for all possible $(x_s, x_{ic})$ and there  
would be $5n$ of these where $n$ is the sample size. All of that computation would be done for each  
kept BART draw. For this reason running BART with keepevery larger than 1 (eg. 10) makes the  
procedure much faster.
Value

The plot methods produce the plots and don’t return anything.
pdbart and pd2bart return lists with components given below. The list returned by pdbart is assigned class ‘pdbart’ and the list returned by pd2bart is assigned class ‘pd2bart’.

fd
A matrix whose \((i, j)\) value is the \(i^{th}\) draw of \(f_s(x_s)\) for the \(j^{th}\) value of \(x_s\). “fd” is for “function draws”.

For pdbart fd is actually a list whose \(k^{th}\) component is the matrix described above corresponding to the \(k^{th}\) variable chosen by argument xind.

The number of columns in each matrix will equal the number of values given in the corresponding component of argument levs (or number of values in levquants).

For pd2bart, fd is a single matrix. The columns correspond to all possible pairs of values for the pair of variables indicated by xind. That is, all possible \((x_i, x_j)\) where \(x_i\) is a value in the levs component corresponding to the first \(x\) and \(x_j\) is a value in the levs components corresponding to the second one. The first \(x\) changes first.

levs
The list of levels used, each component corresponding to a variable.
If argument levs was supplied it is unchanged.
Otherwise, the levels in levs are as constructed using argument levquants.

xlbs
vector of character strings which are the plotting labels used for the variables.

The remaining components returned in the list are the same as in the value of bart. They are simply passed on from the BART run used to create the partial dependence plot. The function plot.bart can be applied to the object returned by pdbart or pd2bart to examine the BART run.

Author(s)
Hugh Chipman: <hugh.chipman@gmail.com>.
Robert McCulloch: <robert.e.mcculloch@gmail.com>.

References

Examples
```r
##simulate data
f = function(x) { return(.5*x[,1] + 2*x[,2]*x[,3]) }
sigma=.2 # y = f(x) + sigma*z
n=100 #number of observations
set.seed(27)
x = matrix(2*runif(n*3)-1,ncol=3) ; colnames(x) = c('rob','hugh','ed')
Ey = f(x)
y = Ey + sigma*rnorm(n)
lmFit = lm(y~.,data.frame(x,y)) #compare lm fit to BART later
par(mfrow=c(1,3)) #first two for pdbart, third for pd2bart
#pdbart: one dimensional partial dependence plot
```
```r
set.seed(99)
pdb1 = pdbbart(x,y,xind=c(1,2),
   levs=list(seq(-1,1,2),seq(-1,1,2)),pl=FALSE,
   keepevery=10,ntree=100,nskip=100,ndpost=200) #should run longer!
plot(pdb1,ylim=c(-.6,.6))
##pbart: two dimensional partial dependence plot
set.seed(99)
pdb2 = pdbbart(x,y,xind=c(2,3),
   levquants=c(.05,.1,.25,.5,.75,.9,.95),pl=FALSE,
   ntree=100,keepevery=10,verbose=FALSE,nskip=100,ndpost=200) #should run longer!
plot(pdb2)
##compare bart fit to linear model and truth = Ey
fitmat = cbind(y,Ey,lmfit$fitted,pdb1$yhat,pred.train.mean)
colnames(fitmat) = c('y','Ey','lm','bart')
print(cor(fitmat))
## plot.bart(pdb1) displays the BART run used to get the plot.
```
Index

*Topic **dplot**
  pdbart, 7

*Topic **nonlinear**
  bart, 1
  makeind, 6
  pdbart, 7

*Topic **nonparametric**
  bart, 1
  makeind, 6
  pdbart, 7

*Topic **regression**
  bart, 1
  makeind, 6
  pdbart, 7

*Topic **tree**
  bart, 1
  makeind, 6
  pdbart, 7

bart, 1, 6, 8, 9

image, 8

makeind, 6

pd2bart (pdbart), 7
pdbart, 5, 7
plot, 8
plot.bart, 9
plot.bart (bart), 1
plot.pd2bart (pdbart), 7
plot.pdbart (pdbart), 7