Package ‘ForestDiffusion’

September 22, 2023

Title Generating and Imputing Tabular Data via Diffusion and Flow XGBoost Models

Version 1.0.0

Date 2023-09-12

Maintainer Alexia Jolicoeur-Martineau <alexia.jolicoeur-martineau@mail.mcgill.ca>

Description Tabular data is hard to acquire and is subject to missing values. This paper proposes a novel approach to generate and impute mixed-type (continuous and categorical) tabular data using score-based diffusion and conditional flow matching. Contrary to previous work that relies on neural networks, we instead utilize XGBoost, a popular Gradient-Boosted Tree method. In addition to being elegant, we empirically show on various datasets that our method i) generates highly realistic synthetic data when the training dataset is either clean or tainted by missing data and ii) generates diverse plausible data imputations. Our method often outperforms deep-learning generation methods and can be trained in parallel using 'CPUs' without the need for a 'GPU'. To make it easily accessible, we release our code through a Python library and an R package <arXiv:2309.09968>.

License MIT + file LICENSE

URL https://github.com/SamsungSAILMontreal/ForestDiffusion

Imports xgboost, foreach, parallelly, doParallel

Depends caret, stats, graphics

Encoding UTF-8

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, datasets, missForest, mice

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2022-01-11 22:52:53 UTC

Author Alexia Jolicoeur-Martineau [cre, aut]

R topics documented:

ForestDiffusion *Diffusion and Flow-based XGBoost Model for generating or imputing data*

Description

Train XGBoost regression models to estimate the score-function (for diffusion models) or the flow (flow-based models). These models can then be used to generate new fake samples or impute missing values.

Usage

```
ForestDiffusion(
  X,
  n_cores,
  label_y = NULL,
  name_y = "y",
  n_t = 50,
  flow = TRUE,
  max_depth = 7,
  n_estimators = 100,
  eta = 0.3,
  duplicate_K = 50,
  true_min_max_values = NULL,
  eps = 0.001,
  beta_min = 0.1,
  beta_max = 8,
  seed = NULL
)
```

Arguments

X	data.frame of the dataset to be used.
n_cores	number of cpu cores used (if NULL, it will use all cores, otherwise it will use min(n_cores, max_available_cores); using more cores makes training faster, but increases the memory cost (so reduce it if you have memory problems)
label_y	optional vector containing the outcome variable if it is categorical for improved performance by training separate models per class; cannot contain missing values
name_y	name of label_y
n_t	number of noise levels (and sampling steps); increase for higher performance, but slows down training and sampling
flow	If TRUE, uses flow (an ODE deterministic method); otherwise uses vp (a SDE stochastic method); 'vp' generally has slightly worse performance, but it is the only method that can be used for imputation
max_depth	max depth of the trees per XGBoost model
n_estimators	number of trees per XGBoost model
eta	learning rate per XGBoost model

duplicate_K	number of noise per sample (or equivalently the number of times the rows of the dataset are duplicated); should be as high as possible; higher values increase the memory demand
true_min_max_values	(optional) list of form [[min_x, min_y], [max_x, max_y]]; If provided, we use these values as the min/max for each variables when using clipping
eps	minimum noise level
beta_min	value of the beta_min in the vp process
beta_max	value of the beta_max in the vp process
seed	(optional) random seed used

Value

Returns an object of the class "ForestDiffusion" which is list containing the XGBoost model fits

References

Alexia Jolicoeur-Martineau, Kilian Fatras, Tal Kachman. Generating and Imputing Tabular Data via Diffusion and Flow-based Gradient-Boosted Trees. arXiv:2309.09968.

Examples

```
## Not run:
data(iris)
iris[,1:4] = missForest::prodNA(iris[,1:4], noNA = 0.2) # adding missing data
X = data.frame(iris[,1:4])
y = iris[,5]

## Generation

# Classification problem (outcome is categorical)
forest_model = ForestDiffusion(X=X, n_cores=1, label_y=y, n_t=50, duplicate_K=50, flow=TRUE)
# last variable will be the label_y
Xy_fake = ForestDiffusion.generate(forest_model, batch_size=NROW(iris))

# When you do not want to train a seperate model per model (or you have a regression problem)
Xy = X
Xy$y = y
forest_model = ForestDiffusion(X=Xy, n_cores=1, n_t=50, duplicate_K=50, flow=TRUE)
Xy_fake = ForestDiffusion.generate(forest_model, batch_size=NROW(iris))

## Imputation

# flow=TRUE generate better data but it cannot impute data
forest_model = ForestDiffusion(X=Xy, n_cores=1, n_t=50, duplicate_K=50, flow=FALSE)
nimp = 5 # number of imputations needed
# regular (fast)
Xy_fake = ForestDiffusion.impute(forest_model, k=nimp)
# REPAINT (slow, but better)
Xy_fake = ForestDiffusion.impute(forest_model, repaint=TRUE, r=10, j=5, k=nimp)

## End(Not run)
```

ForestDiffusion.generate
Generate new observations with a trained ForestDiffusion model

---

Description

Generate new observations by solving the reverse SDE (vp) / ODE (flow) starting from pure Gaussian noise.

Usage

```
ForestDiffusion.generate(object, batch_size = NULL, n_t = NULL, seed = NULL)
```

Arguments

object	a ForestDiffusion object
batch_size	(optional) number of observations generated; if not provided, will generate as many observations as the original dataset
n_t	(optional) number of noise levels (and sampling steps); increase for higher performance, but slows down training and sampling; if not provided, will use the same n_t as used in training.
seed	(optional) random seed used

Value

Returns a data.frame with the generated data

References

Alexia Jolicoeur-Martineau, Kilian Fatras, Tal Kachman. Generating and Imputing Tabular Data via Diffusion and Flow-based Gradient-Boosted Trees. arXiv:2309.09968.

Examples

```
 ## Not run:
 data(iris)
 X = data.frame(iris[,1:4])
 y = iris[,5]

 ## Generation

 Xy = X
 Xy$y = y
 forest_model = ForestDiffusion(X=Xy, n_cores=1, n_t=50, duplicate_K=50, flow=TRUE)
 Xy_fake = ForestDiffusion.generate(forest_model, batch_size=NROW(Xy))

## End(Not run)
```

`ForestDiffusion.impute` *Impute missing data with a trained ForestDiffusion model*

Description

Impute missing data by solving the reverse SDE while keeping the non-missing data intact.

Usage

```
ForestDiffusion.impute(
  object,
  k = 1,
  X = NULL,
  label_y = NULL,
  repaint = FALSE,
  r = 5,
  j = 0.1,
  n_t = NULL,
  seed = NULL
)
```

Arguments

`object`	a ForestDiffusion object
`k`	number of imputations
`X`	(optional) data.frame of the dataset to be imputed; If not provided, the training dataset will be imputed instead
`label_y`	(optional) vector containing the outcome variable if it is categorical for improved performance by training separate models per class; cannot contain missing values; if not provided, the training label_y will be used if it exists.
`repaint`	If TRUE, it will impute using the REPAINT technique for improved performance
`r`	number of repaints (default=10)
`j`	jump size in percentage (default: 10 percent of the samples), this is part of REPAINT
`n_t`	(optional) number of noise levels (and sampling steps); increase for higher performance, but slows down training and sampling; if not provided, will use the same n_t as used in training.
`seed`	(optional) random seed used

Value

Returns a data.frame with the generated data

References

Alexia Jolicoeur-Martineau, Kilian Fatras, Tal Kachman. Generating and Imputing Tabular Data via Diffusion and Flow-based Gradient-Boosted Trees. arXiv:2309.09968.

Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, Luc Van Gool. RePaint: Inpainting using Denoising Diffusion Probabilistic Models. arXiv:2201.09865.

Examples

```
 ## Not run:
 data(iris)
 X = data.frame(iris[,1:4])
 y = iris[,5]

 ## Imputation

 # add missing data
 Xy = missForest::prodNA(Xy, noNA = 0.2)

 nimp = 5 # number of imputations needed
 Xy = X
 Xy$y = y
 forest_model = ForestDiffusion(X=Xy, n_cores=1, n_t=50, duplicate_K=50, flow=FALSE)
 # regular (fast)
 Xy_fake = ForestDiffusion.impute(forest_model, k=nimp)
 # REPAINT (slow, but better)
 Xy_fake = ForestDiffusion.impute(forest_model, repaint=TRUE, r=10, j=5, k=nimp)

## End(Not run)
```

`with_datasets` *Evaluate an expression in multiple generated/imputed datasets*

Description

It performs a computation for each dataset (function modified from mice::with.mids). For example, you can use this function to train a different glm model per dataset and then pool the estimates (akin to with multiple imputations, but more general so that it can be applied to any dataset).

Usage

```
with_datasets(data, expr)
```

Arguments

`data`	a list of datasets
`expr`	An expression to evaluate for each imputed data set. Formula's containing a dot (notation for "all other variables") do not work.

Value

An object of S3 class mira

Examples

```
## Not run:
library(mice)

# Load iris
data(iris)
Xy = data.frame(iris[,1:4])
Xy$y = iris[,5]

# add missing data
Xy = missForest::prodNA(Xy, noNA = 0.2)

forest_model = ForestDiffusion(X=Xy, n_cores=1, n_t=50, duplicate_K=50, flow=FALSE)
nimp = 5 # number of imputations needed
# regular (fast)
Xy_imp = ForestDiffusion.impute(forest_model, k=nimp)
# REPAINT (slow, but better)
Xy_imp = ForestDiffusion.impute(forest_model, repaint=TRUE, r=10, j=5, k=nimp)

# Fit a model per imputed dataset
fits <- with_datasets(Xy_imp, glm(y ~ Sepal.Length, family = 'binomial'))

# Pool the results
mice::pool(fits)

## End(Not run)
```

Index