RandomForest Documentation

Release 0.0.10

Henrik Boström

Sep 08, 2017

Contents

Contents:

CHAPTER 1

RandomForest v. 0.0.10

A Julia package that implements random forests for classification, regression and survival analysis with conformal prediction. [NOTE: survival analysis under development]

There are two basic ways of working with the package:

- running an experiment with multiple datasets, possibly comparing multiple methods, i.e., random forests with different parameter settings, or
- working with a single dataset, to evaluate, generate, store, load or apply a random forest

All named arguments below are optional, while the others are mandatory.

The classification datasets included in uci.zip have been downloaded and adapted from:

Lichman, M. (2013). UCI Machine Learning Repository http://archive.ics.uci.edu/ml. Irvine, CA: University of California, School of Information and Computer Science.

The regression datasets included in regression.zip have been downloaded and adapted from the above source and from:

Rasmussen, C.E., Neal, R.M., Hinton, G., van Camp, D., Revow, M., Ghahramani, Z., Kustra, R., and Tibshirani, R. (1996) Delve data for evaluating learning in valid experiments http://www.cs.toronto.edu/~delve/data/datasets.html

The survival datasets included in survival.zip have been downloaded and adapted from:

Statistical Software Information, University of Massachusetts Amherst, Index of Survival Analysis Datasets, https://www.umass.edu/statdata/statdata/stat-survival.html

CHAPTER 2

To install the package

Start Julia (which can be downloaded from http://julialang.org/) at a command prompt:

```
julia
```

at the Julia REPL, give the following command to clone the package from GitHub:

```
julia> Pkg.clone("https://github.com/henrikbostrom/RandomForest.git")
```

Unless you do not already have it installed, install also the DataFrames package:

```
julia> Pkg.add("DataFrames")
```

Load the RandomForest package:

```
julia> using RandomForest
```

CHAPTER 3

Functions for running experiments

apply_model

To apply a model to loaded data:

```
julia> apply_model(<model>, confidence = <confidence>)
```

The argument should be on the following format:

```
model : a generated or loaded model (see generate_model and load_model)
confidence : a float between 0 and 1 or :std (default = :std)
             - probability of including the correct label in the prediction region
             - :std means employing the same confidence level as used during training
```

evaluate_method

To evaluate a method or several methods for generating a random forest:

```
julia> evaluate_method(method = forest(...), protocol = <protocol>)
julia> evaluate_methods(methods = [forest(...), ...], protocol = <protocol>)
```

The arguments should be on the following format:

```
method : a call to forest(...) as explained above (default = forest())
methods : a list of calls to forest(...) as explained above (default = [forest()])
protocol : integer, float, :cv or :test as explained above (default = 10)
```

experiment

An experiment is run by calling experiment(...) in the following way:

```
julia> experiment(files = <files>, separator = <separator>, protocol = <protocol>,
                  normalizetarget = <normalizetarget>, normalizeinput =
↪<normalizeinput>,
                  methods = [<method>, ...])
```

The arguments should be on the following format:

```
files : list of file names or path to directory (default = ".")
    - in a specified directory, files with extensions other than .txt and .csv are␣
↪ignored
    - each file should contain a dataset (see format requirements below)
    - example: files = ["uci/house-votes.txt", "uci/glass.txt"],
    - example: files = "uci"

separator : single character (default = ',')
    - the character to use as field separator
    - example: separator = '        ' (the tab character)

protocol : integer, float, :cv, :test (default = 10)
    - the experiment protocol:
        an integer means using cross-validation with this no. of folds.
        a float between 0 and 1 means using this fraction of the dataset for testing
        :cv means using cross-validation with folds specified by a column labeled FOLD
        :test means dividing the data into training and test according to boolean␣
↪values
         in a column labeled TEST (true means that the example is used for testing)
    - example: protocol = 0.25 (25% of the data is for testing)

normalizetarget : boolean (default = false)
    - true means that each regression value v will be replaced by
      (v-v_min)/(v_max-v_min), where v_min and V-max are the minimum and maximum␣
↪values
    - false means that the original regression values are kept

normalizeinput : boolean (default = false)
    - true means that each numeric input value v will be replaced by
      (v-v_min)/(v_max-v_min), where v_min and V-max are the minimum and maximum␣
↪values
    - false means that the original values are kept

method : a call on the form forest(...) (default = forest())

    - The call may have the following (optional) arguments:

        notrees : integer (default = 100)
            - no. of trees to generate in the forest

        minleaf : integer (default = 1)
            - minimum no. of required examples to form a leaf

        maxdepth : integer (default = 0)
            - maximum depth of generated trees (0 means that there is no depth limit)

        randsub : integer, float, :default, :all, :log2, or :sqrt (default = :default)
```

```
            - no. of randomly selected features to evaluate at each split:
               :default means :log2 for classification and 1/3 for regression
               :all means that all features are used (no feature subsampling takes␣
↪place)
               :log2 means that log2 of the no. of features are sampled
               :sqrt means that sqrt of the no. of features are sampled
               an integer (larger than 0) means that this number of features are␣
↪sampled
               a float (between 0.0 and 1.0) means that this fraction of features are␣
↪sampled

        randval : boolean (default = true)
            - true means that a single randomly selected value is used to form␣
↪conditions for each
              feature in each split
            - false mean that all values are used to form conditions when evaluating␣
↪features for
              each split

        splitsample : integer (default = 0)
            - no. of randomly selected examples to use for evaluating each split
            - 0 means that no subsampling of the examples will take place

        bagging : boolean (default = true)
            - true means that a bootstrap replicate of the training examples is used␣
↪for each tree
            - false means that the original training examples are used when building␣
↪each tree

        bagsize : float or integer (default = 1.0)
            - no. of randomly selected examples to include in the bootstrap replicate
            - an integer means that this number of examples are sampled with␣
↪replacement
            - a float means that the corresponding fraction of examples are sampled␣
↪with replacement

        modpred : boolean (default = false)
            - true means that for each test instance, the trees for which a randomly␣
↪selected training
              instance is out-of-bag is used for prediction and the training instance␣
↪is not used for
              calculating a calibration score
            - false means that all trees in the forest are used for prediction and␣
↪all out-of-bag scores
              are used for calibration

        laplace : boolean (default = false)
            - true means that class probabilities at each leaf node is Laplace␣
↪corrected
            - false means that class probabilities at each leaf node equal the␣
↪relative class
              frequencies

        confidence : a float between 0 and 1 (default = 0.95)
            - probability of including the correct label in the prediction region

        conformal : :default, :std, :normalized or :classcond (default = :default)
            - method used to calculate prediction regions
```

```
            - For classification, the following options are allowed:
               :default is the same as :std
               :std means that validity is guaranteed in general, but not for each␣
↪class
               :classcond means that validity is guaranteed for each class
            - For regression, the following options are allowed:
               :default is the same as :normalized
               :std results in the same region size for all predictions
               :normalized means that each region size is dependent on the spread
                of predictions among the individual trees


Examples:

The call experiment(files = "uci") is hence the same as

experiment(files = "uci", separator = ´,´, protocol = 10, methods = [forest()])

The following compares the default random forest to one with 1000 trees and a␣
↪maxdepth of 10:

julia> experiment(files = "uci", methods = [forest(), forest(notrees = 1000, maxdepth␣
↪= 10)])
```

A dataset should have the following format:

```
<names-row>
<data-row>
...
<data-row>
```

where

```
<names-row> = <name><separator><name><separator>...<name>
```

and

```
<data-row>  = <value><separator><value><separator>...<value>
```

<name> can be any of the following:

```
CLASS            - declares that the column contains class labels
REGRESSION       - declares that the column contains regression values
ID               - declares that the column contains identifier labels
IGNORE           - declares that the column should be ignored
FOLD             - declares that the column contains labels for cross-validation folds
WEIGHT           - declares that the column contains instance weights
any other value  - is used to create a variable name
```

<separator> is a single character (as specified above)

<value> can be any of the following:

```
integer          - is handled as a number if all values in the same column are of␣
↪type integer,
                   float or NA, and as a string otherwise
float            - is handled as a number if all values in the same column are of␣
↪type integer,
```

```
                    float or NA, and as a string otherwise
NA                - is handled as a missing value
any other value   - is handled as a string
```

Example:

```
ID,RI,Na,Mg,Al,Si,K,Ca,Ba,Fe,CLASS
1,1.52101,NA,4.49,1.10,71.78,0.06,8.75,0.00,0.00,1
2,1.51761,13.89,NA,1.36,72.73,0.48,7.83,0.00,0.00,1
3,1.51618,13.53,3.55,1.54,72.99,0.39,7.78,0.00,0.00,1
...
```

A sparse dataset should have the following format:

```
<data-row>
...
<data-row>
```

where

```
<data-row> = <column number>:<value><separator><column number>:<value><separator> ...
↪<column number>:<value><separator>
```

An example for a sparse dataset: https://archive.ics.uci.edu/ml/machine-learning-databases/dexter/DEXTER/dexter_test.data

<column number> an integer number representing column index

<value> can be integer, or float

<separator> is a single character (as specified above)

For classification tasks the following measures are reported:

```
Acc         - accuracy, i.e., fraction of examples correctly predicted
AUC         - area under ROC curve
Brier       - Brier score
AvAcc       - average accuracy for single trees in the forest
DEOAcc      - difference of the estimated and observed accuracy
AEEAcc      - absolute error of the estimated accuracy
AvBrier     - average Brier score for single trees in the forest
VBrier      - average squared deviation of single tree predictions from forest␣
↪predictions
Margin      - diff. between prob. for correct class and prob. for most prob. other␣
↪class
Prob        - probability for predicted class
Valid       - fraction of true labels included in prediction region
Region      - size, i.e., number of labels, in prediction region
OneC        - fraction of prediction regions containing exactly one true label
Size        - the number of nodes in the forest
Time        - the total time taken for both training and testing
```

For regression tasks the following measures are reported:

```
MSE         - mean squared error
Corr        - the Pearson correlation between predicted and actual values
AvMSE       - average mean squared error for single trees in the forest
VarMSE      - average squared deviation of single tree predictions from forest␣
↪predictions
```

```
DEOMSE      - difference of the estimated and observed MSE
AEEMSE      - absolute error of the estimated MSE
Valid       - fraction of true labels included in prediction region
Region      - average size of prediction region
Size        - the number of nodes in the forest
Time        - the total time taken for both training and testing
```

---

generate_model

To generate a model from the loaded dataset:

```
julia> m = generate_model(method = forest(...))
```

The argument should be on the following format:

```
method : a call to forest(...) as explained above (default = forest())
```

---

load_data

To load a dataset from a file or dataframe:

```
julia> load_data(<filename>, separator = <separator>)
julia> load_data(<dataframe>)
```

The arguments should be on the following format:

```
filename : name of a file containing a dataset (see format requirements above)
separator : single character (default = ',')
dataframe : a dataframe where the column labels should be according to the format␣
↪requirements above
```

---

load_model

To load a model from file:

```
julia> rf = load_model(<file>)
```

The argument should be on the following format:

```
file : name of file in which a model has been stored
```

---

load_sparse_data

To load a dataset from a file:

```
julia> load_sparse_data(<filename>, <labels_filename>, predictionType =
↪<predictionType>, separator = <separator>, n = <numberOfFeatures>)
```

The arguments should be on the following format:

```
filename : name of a file containing a sparse dataset (see format requirements above)
labels_filename :  name of a file containing a vector of labels
separator : single character (default = ' ')
predictionType : one of :CLASS, :REGRESSION, or :SURVIVAL
n : Number of features in the dataset (auto detected if not provided)
```

---

runexp

`runexp` is used to run experiments on a number of standard datasets

---

store_model

To store a model in a file:

```
julia> store_model(<model>, <file>)
```

The arguments should be on the following format:

```
model : a generated or loaded model (see generate_model and load_model)
file : name of file to store model in
```

CHAPTER 4

Functions for working with a single dataset

apply_model

To apply a model to loaded data:

```
julia> apply_model(<model>, confidence = <confidence>)
```

The argument should be on the following format:

```
model : a generated or loaded model (see generate_model and load_model)
confidence : a float between 0 and 1 or :std (default = :std)
             - probability of including the correct label in the prediction region
             - :std means employing the same confidence level as used during training
```

evaluate_method

To evaluate a method or several methods for generating a random forest:

```
julia> evaluate_method(method = forest(...), protocol = <protocol>)
julia> evaluate_methods(methods = [forest(...), ...], protocol = <protocol>)
```

The arguments should be on the following format:

```
method : a call to forest(...) as explained above (default = forest())
methods : a list of calls to forest(...) as explained above (default = [forest()])
protocol : integer, float, :cv or :test as explained above (default = 10)
```

generate_model

To generate a model from the loaded dataset:

```
julia> m = generate_model(method = forest(...))
```

The argument should be on the following format:

```
method : a call to forest(...) as explained above (default = forest())
```

---

load_data

To load a dataset from a file or dataframe:

```
julia> load_data(<filename>, separator = <separator>)
julia> load_data(<dataframe>)
```

The arguments should be on the following format:

```
filename : name of a file containing a dataset (see format requirements above)
separator : single character (default = ',')
dataframe : a dataframe where the column labels should be according to the format␣
↪requirements above
```

---

load_model

To load a model from file:

```
julia> rf = load_model(<file>)
```

The argument should be on the following format:

```
file : name of file in which a model has been stored
```

---

load_sparse_data

To load a dataset from a file:

```
julia> load_sparse_data(<filename>, <labels_filename>, predictionType =
↪<predictionType>, separator = <separator>, n = <numberOfFeatures>)
```

The arguments should be on the following format:

```
filename : name of a file containing a sparse dataset (see format requirements above)
labels_filename :  name of a file containing a vector of labels
separator : single character (default = ' ')
predictionType : one of :CLASS, :REGRESSION, or :SURVIVAL
n : Number of features in the dataset (auto detected if not provided)
```

---

store_model

To store a model in a file:

```
julia> store_model(<model>, <file>)
```

The arguments should be on the following format:

```
model : a generated or loaded model (see generate_model and load_model)
file : name of file to store model in
```

CHAPTER 5

Functions for printing

describe_data

To get a description of a loaded dataset: describe_data(<dataframe>)

describe_model

To get a description of a model:

```
julia> describe_model(<model>)
```

The argument should be on the following format:

```
model : a generated or loaded model (see generate_model and load_model)
```

CHAPTER 6

Summary of all functions

All named arguments are optional, while the others are mandatory.

To run an experiment:

experiment(files = <files>, separator = <separator>, protocol = <protocol>, methods = [<method>, ...])

To work with a single dataset:

load_data(<filename>, separator = <separator>)

load_data(<dataframe>)

describe_data(<dataframe>)

evaluate_method(method = forest(...), protocol = <protocol>)

evaluate_methods(methods = [forest(...), ...], protocol = <protocol>)

m = generate_model(method = forest(...))

describe_model(<model>)

store_model(<model>, <file>)

m = load_model(<file>)

apply_model(<model>, confidence = <confidence>)