Package ‘aurelius’

July 3, 2017

Type Package

Version 0.8.4

Date 2017-07-02

Title Generates PFA Documents from R Code and Optionally Runs Them

Description Provides tools for converting R objects and syntax into the Portable Format for Analytics (PFA). Allows for testing validity and runtime behavior of PFA documents through rPython and Titus, a more complete implementation of PFA for Python. The Portable Format for Analytics is a specification for event-based processors that perform predictive or analytic calculations and is aimed at helping smooth the transition from statistical model development to large-scale and/or online production. See <http://dmg.org/pfa> for more information.

URL `https://github.com/opendatagroup/hadrian/tree/master/aurelius`

BugReports `https://github.com/opendatagroup/hadrian/issues`

Depends R (>= 3.2.0)

Imports methods, stats, utils, jsonlite (>= 1.1), gbm (>= 2.1.1), glmnet (>= 2.0-5)

Enhances rPython

Suggests knitr, rmarkdown, testthat, MASS, forecast, caret, ipred, flexclust, e1071, randomForest, rpart

VignetteBuilder knitr

License Apache License 2.0 | file LICENSE

NeedsCompilation no

LazyData true

RoxygenNote 6.0.1

Collate 'arima.R' 'aurelius.R' 'avro.R' 'avro_from_df.R' 'avro_fullname.R' 'avro_type.R' 'avro_typemap.R' 'converters.R' 'ets.R' 'forecast.R' 'gbm.R' 'gen_unique_name.R' 'glm.R' 'glmnet.R' 'holtwinters.R' 'jsonobj.R' 'kcca.R' 'kmeans.R' 'knn.R' 'lda.R' 'lm.R' 'naiveBayes.R' 'pfa.R' 'pfa_cellpool.R' 'pfa_document.R' 'pfa_engine.R' 'pfa_expr.R' 'pfa_utils.R' 'randomForest.R' 'read_pfa.R' 'rpart.R' 'write_pfa.R' 'zzz.R'

Author Steven Mortimer [aut, cre],
Collin Bennett [aut],
Open Data Group [cph]

Maintainer Steven Mortimer <reportmort@gmail.com>

Repository CRAN

Date/Publication 2017-07-03 07:11:41 UTC

R topics documented:

aurelius *aurelius package*

Description

Generates PFA Documents from R Code and Optionally Runs Them

Details

Converts R syntax into PFA and provides tools for assembling a PFA document within R. Tests validity and runtime behavior of PFA by offloading PFA and data to Titus (through rPython). Facilitates conversion of common R model output to PFA using aurelius.* libraries. Aurelius is part of Hadrian and is on Github at https://github.com/opendatagroup/hadrian.

Examples

```
## Not run:
library("aurelius")

# build a model
lm_model <- lm(mpg ~ hp, data = mtcars)

# convert the lm object to a list of lists PFA representation
lm_model_as_pfa <- pfa(lm_model)

# save as plain-text JSON
write_pfa(lm_model_as_pfa, file = "my-model.pfa")

# read the model back in
read_pfa(file("my-model.pfa"))

## End(Not run)
```

avro_array *avro_array*

Description

Constructs a `list` of lists Avro schema for the array type.

Usage

```
avro_array(items)
```

Arguments

`items`	schema for the homogeneous array

Examples

```
avro_array(avro_int)
avro_array(avro_string)
```

avro_boolean *avro_boolean*

Description

Constructs a `list` of lists Avro schema for the boolean (logical) type.

Usage

```
avro_boolean
```

Format

An object of class `character` of length 1.

avro_bytes *avro_bytes*

Description

Constructs a `list` of lists Avro schema for the bytes (unstructured byte array) type.

Usage

```
avro_bytes
```

Format

An object of class `character` of length 1.

`avro_double` *avro_double*

Description

Constructs a `list` of lists Avro schema for the double (floating-point numeric with 64-bit precision) type.

Usage

```
avro_double
```

Format

An object of class `character` of length 1.

`avro_enum` *avro_enum*

Description

Constructs a `list` of lists Avro schema for the enum (set of symbols) type.

Usage

```
avro_enum(symbols, name = NULL, namespace = NULL)
```

Arguments

`symbols`	list of string-valued symbol names
`name`	required name (if missing, `gen_unique_enum_name` is invoked)
`namespace`	optional namespace

Source

gen_unique_name.R

Examples

```
avro_enum(list("one", "two", "three"))
```

`avro_fixed` *avro_fixed*

Description

Constructs a `list` of lists Avro schema for the fixed (byte array with fixed size) type.

Usage

```
avro_fixed(size, name = NULL, namespace = NULL)
```

Arguments

`size`	size of the byte array
`name`	required name (if missing, `gen_unique_fixed_name` is invoked)
`namespace`	optional namespace

Source

gen_unique_name.R

Examples

```
avro_fixed(6, "MACAddress")
```

`avro_float` *avro_float*

Description

Constructs a `list` of lists Avro schema for the float (floating-point numeric with 32-bit precision) type.

Usage

```
avro_float
```

Format

An object of class `character` of length 1.

`avro_from_df` *avro_from_df*

Description

Convenience function for creating an Avro input schema from a data frame

Usage

```
avro_from_df(df, exclude = list(), name = NULL, namespace = NULL)
```

Arguments

`df`	a `data.frame`
`exclude`	set of field names to exclude
`name`	required name of the record (if not specified, gen_unique_rec_name() will be invoked)
`namespace`	optional namespace of the record

Value

a `list` of lists representing an Avro record type

Examples

```
avro_from_df(data.frame(x = c(1, 3, 5)))
```

`avro_fullname` *avro_fullname*

Description

Yields the full type name (with namespace) of an Avro `list` of lists

Usage

```
avro_fullname(type)
```

Arguments

`type`	Avro `list` of lists

Value

string representing the full name

Examples

```
avro_fullname(avro_record(list(), "MyRecord"))                   # "MyRecord"
avro_fullname(avro_record(list(), "MyRecord", "com.wowzers"))   # "com.wowzers.MyRecord"
```

`avro_int` *avro_int*

Description

Constructs a `list` of lists Avro schema for the int (integer numeric with 32-bit precision) type.

Usage

```
avro_int
```

Format

An object of class `character` of length 1.

`avro_long` *avro_long*

Description

Constructs a `list` of lists Avro schema for the long (integer numeric with 64-bit precision) type.

Usage

```
avro_long
```

Format

An object of class `character` of length 1.

avro_map *avro_map*

Description

Constructs a `list` of lists Avro schema for the map type.

Usage

```
avro_map(values)
```

Arguments

`values`	schema for the homogeneous map

Examples

```
avro_map(avro_int)
avro_map(avro_string)
```

avro_null *avro_null*

Description

Constructs a `list` of lists Avro schema for the null type (type with only one value).

Usage

```
avro_null
```

Format

An object of class `character` of length 1.

`avro_record` *avro_record*

Description

Constructs a `list` of lists Avro schema for the record type.

Usage

```
avro_record(fields, name = NULL, namespace = NULL)
```

Arguments

`fields`	named list of field names and schemas
`name`	required name (if missing, gen_unique_rec_name is invoked)
`namespace`	optional namespace

Examples

```
avro_record(list(one = avro_int, two = avro_double, three = avro_string))
```

`avro_string` *avro_string*

Description

Constructs a `list` of lists Avro schema for the string (UTF-8) type.

Usage

```
avro_string
```

Format

An object of class `character` of length 1.

avro_type *avro_type*

Description

Inspects an R object and produces the corresponding Avro type name

Usage

```
avro_type(obj)
```

Arguments

obj	object to inspect

Value

a `list` of lists Avro schema

Examples

```
avro_type("hello")          # "string"
avro_type(factor("hello"))  # "string"
avro_type(3.14)             # "double"
avro_type(3)                # "int"
```

avro_typemap *avro_typemap*

Description

Convenience function for ensuring that Avro type schemas are declared exactly once. It returns a function that yields a full type declaration the first time it is invoked and just a name on subsequent times.

Usage

```
avro_typemap(...)
```

Arguments

...	key-value pairs of Avro type schemas

Value

a function that yields Avro type schemas or just their names

Examples

```
tm <- avro_typemap(
    MyType1 = avro_record(list(one = avro_int, two = avro_double, three = avro_string)),
    MyType2 = avro_array(avro_double)
)
tm("MyType1")           # produces the whole declaration
tm("MyType1")           # produces just "MyType1"
tm("MyType2")           # produces the whole declaration
tm("MyType2")          # produces the declaration again because this is not a named type
```

---

avro_union *avro_union*

Description

Constructs a `list` of lists Avro schema for the tagged union type.

Usage

```
avro_union(...)
```

Arguments

`...`	schemas for each of the possible sub-types

Examples

```
avro_union(avro_null, avro_int)         # a way to make a nullable int
avro_union(avro_double, avro_string)    # any set of types can be unioned
```

---

build_model *build_model*

Description

Builds an entire PFA list of lists based on a model object

Usage

```
build_model(object, ...)
```

Arguments

`object`	a model object
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the tree that can be inserted into a cell or pool

Examples

```
# all the "build_model" methods found
methods("build_model")
```

`build_model.gbm` *build_model.gbm*

Description

Builds an entire PFA list of lists based on a single gbm model tree

Usage

```
## S3 method for class 'gbm'
build_model(object, which_tree = 1, ...)
```

Arguments

`object`	a object of class gbm
`which_tree`	an integer indicating which single tree to build
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the tree that can be inserted into a cell or pool

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- ((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

bernoulli_model <- gbm::gbm(Y ~ X1 + X2,
                            data = dat,
                            distribution = 'bernoulli')
my_tree <- build_model(bernoulli_model, 1)
```

build_model.lda *build_model.lda*

Description

Builds an entire PFA list of lists based on a lda() fit

Usage

```
## S3 method for class 'lda'
build_model(object, ...)
```

Arguments

`object`	a object of class lda
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the linear discriminant model that can be inserted into a cell or pool

Examples

```
model <- MASS::lda(Species ~ ., data=iris)
model_built <- build_model(model)
```

build_model.naiveBayes *build_model.naiveBayes*

Description

Builds an entire PFA list of lists based on a naiveBayes

Usage

```
## S3 method for class 'naiveBayes'
build_model(object, threshold = 0.001, eps = 0, ...)
```

Arguments

`object`	a object of class naiveBayes
`threshold`	a value replacing cells with probabilities within eps range.
`eps`	a numeric for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by theshold.)
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the naiveBayes model that can be inserted into a cell or pool

Examples

```
model <- e1071::naiveBayes(Species ~ ., data=iris)
model_built <- build_model(model)
```

---

build_model.randomForest

build_model.randomForest

Description

Builds an entire PFA list of lists based on a single randomForest model tree

Usage

```
## S3 method for class 'randomForest'
build_model(object, which_tree = 1, ...)
```

Arguments

`object`	a object of class randomForest
`which_tree`	an integer indicating which single tree to build
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the tree that can be inserted into a cell or pool

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- factor((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

model <- randomForest::randomForest(Y ~ X1 + X2, data=dat, ntree=10)
my_tree <- build_model(model, 1)
```

`build_model.rpart` *build_model.rpart*

Description

Builds an entire PFA list of lists based on a single tree

Usage

```
## S3 method for class 'rpart'
build_model(object, ...)
```

Arguments

`object`	a object of class "rpart"
`...`	further arguments passed to or from other methods

Value

a `list` of lists representation of the tree that can be inserted into a cell or pool

Examples

```
model <- rpart::rpart(Kyphosis ~ Age + as.factor(Number), data = rpart::kyphosis)
my_tree <- build_model(model)
```

`extract_params` *extract_params*

Description

Extracts parameters of a model object

Usage

```
extract_params(object, ...)
```

Arguments

`object`	a model object
`...`	further arguments passed to or from other methods

Value

a `list` that is extracted from the tree model object

Examples

```
# all the "extract_params" methods found
methods("extract_params")
```

`extract_params.Arima` *extract_params.Arima*

Description

Extract model parameters from an ARIMA model created using the arima(), Arima(), or auto.arima() functions

Usage

```
## S3 method for class 'Arima'
extract_params(object, ...)
```

Arguments

`object`	an object of class "Arima"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- stats::arima(presidents, c(3, 0, 0))
extracted_model <- extract_params(model)

model <- forecast::Arima(USAccDeaths, order=c(2,2,2), seasonal=c(0,2,2))
extracted_model <- extract_params(model)

model <- forecast::auto.arima(WWWusage)
extracted_model <- extract_params(model)
```

`extract_params.cv.glmnet` *extract_params.cv.glmnet*

Description

Extract generalized linear model net parameters from a cv.glmnet object

Usage

```
## S3 method for class 'cv.glmnet'
extract_params(object, lambda = object[["lambda.1se"]],
  ...)
```

Arguments

`object`	an object of class "cv.glmnet"
`lambda`	a numeric value of the penalty parameter lambda at which coefficients are required
`...`	further arguments passed to or from other methods

Value

PFA as a `list` of lists that can be inserted into a cell or pool

Examples

```
X <- matrix(c(rnorm(100), runif(100)), nrow=100, ncol=2)
Y <- factor(3 - 5 * X[,1] + 3 * X[,2] + rnorm(100, 0, 3) > 0)

model <- glmnet::cv.glmnet(X, Y, family = 'binomial')
my_model_params <- extract_params(model)
```

`extract_params.ets` *extract_params.ets*

Description

Extract model parameters from an Exponential smoothing state space model created using the ets() function from the forecast package.

Usage

```
## S3 method for class 'ets'
extract_params(object, ...)
```

Arguments

`object`	an object of class "ets"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- forecast::ets(USAccDeaths, model="ZZZ")
extracted_model <- extract_params(model)
```

`extract_params.forecast` *extract_params.forecast*

Description

Extract model parameters from a model with class "forecast" created using the forecast package

Usage

```
## S3 method for class 'forecast'
extract_params(object, ...)
```

Arguments

`object`	an object of class "forecast"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- forecast::holt(airmiles)
extracted_model <- extract_params(model)
```

`extract_params.gbm` *extract_params.gbm*

Description

Extracts a parameters from an ensemble made by the gbm function

Usage

```
## S3 method for class 'gbm'
extract_params(object, which_tree = 1, ...)
```

Arguments

`object`	an object of class "gbm"
`which_tree`	the number of the tree to extract
`...`	further arguments passed to or from other methods

Value

a `list` that is extracted from the gbm object

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- ((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

bernoulli_model <- gbm::gbm(Y ~ X1 + X2,
                            data = dat,
                            distribution = 'bernoulli')
my_tree <- extract_params(bernoulli_model, 1)
```

`extract_params.glm` *extract_params.glm*

Description

Extract generalized linear model parameters from the glm library

Usage

```
## S3 method for class 'glm'
extract_params(object, ...)
```

Arguments

`object`	an object of class "glm"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
X1 <- rnorm(100)
X2 <- runif(100)
Y <- 3 - 5 * X1 + 3 * X2 + rnorm(100, 0, 3)
Y <- Y > 0

glm_model <- glm(Y ~ X1 + X2, family = binomial(logit))
model_params <- extract_params(glm_model)
```

`extract_params.glmnet` *extract_params.glmnet*

Description

Extract generalized linear model net parameters from the glmnet library

Usage

```
## S3 method for class 'glmnet'
extract_params(object, lambda = NULL, ...)
```

Arguments

`object`	an object of class "glmnet"
`lambda`	a numeric value of the penalty parameter lambda at which coefficients are required
`...`	further arguments passed to or from other methods

Value

a `list` of lists that can be modified to insert into a cell or pool

Examples

```
X <- matrix(c(rnorm(100), runif(100)), nrow=100, ncol=2)
Y <- factor(3 - 5 * X[,1] + 3 * X[,2] + rnorm(100, 0, 3) > 0)

model <- glmnet::glmnet(X, Y, family = 'binomial')
my_model_params <- extract_params(model)
```

`extract_params.HoltWinters` *extract_params.HoltWinters*

Description

Extract Holt Winters model parameters from the stats library

Usage

```
## S3 method for class 'HoltWinters'
extract_params(object, ...)
```

Arguments

`object`	an object of class "HoltWinters"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- HoltWinters(co2)
extracted_model <- extract_params(model)
```

`extract_params.ipredknn` *extract_params.ipredknn*

Description

Extract K-nearest neighbor model parameters from a ipredknn object created by the ipred library

Usage

```
## S3 method for class 'ipredknn'
extract_params(object, ...)
```

Arguments

`object`	an object of class "ipredknn"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
iris2 <- iris
colnames(iris2) <- gsub('\\.', '_', colnames(iris2))
model <- ipred::ipredknn(Species ~ ., iris2)
params <- extract_params(model)
```

extract_params.kcca *extract_params.kcca*

Description

Extract K-centroids model parameters from a kcca object created by the flexclust library

Usage

```
## S3 method for class 'kcca'
extract_params(object, ...)
```

Arguments

`object`	an object of class "kcca"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- flexclust::kcca(iris[,1:4], k = 3, family=flexclust::kccaFamily("kmeans"))
extracted_params <- extract_params(model)
```

`extract_params.kccasimple` *extract_params.kccasimple*

Description

Extract K-centroids model parameters from a kccasimple object created by the flexclust library

Usage

```
## S3 method for class 'kccasimple'
extract_params(object, ...)
```

Arguments

`object`	an object of class "kccasimple"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- flexclust::kcca(iris[,1:4], k = 3,
                         family=flexclust::kccaFamily("kmeans"), simple=TRUE)
extracted_params <- extract_params(model)
```

`extract_params.kmeans` *extract_params.kmeans*

Description

Extract K-means model parameters from a kmeans object

Usage

```
## S3 method for class 'kmeans'
extract_params(object, ...)
```

Arguments

`object`	an object of class "kmeans"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Source

kcca.R

Examples

```
model <- kmeans(x=iris[, 1:2], centers=3)
extracted_params <- extract_params(model)
```

---

extract_params.knn3 *extract_params.knn3*

Description

Extract K-nearest neighbor model parameters from a knn3 object created by the caret library

Usage

```
## S3 method for class 'knn3'
extract_params(object, ...)
```

Arguments

`object`	an object of class "knn3"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Source

pfa_utils.R

Examples

```
iris2 <- iris
colnames(iris2) <- gsub('\\.', '_', colnames(iris2))
model <- caret::knn3(Species ~ ., iris2)
extracted_params <- extract_params(model)
```

`extract_params.knnreg` *extract_params.knnreg*

Description

Extract K-nearest neighbor model parameters from a knnreg object created by the caret library

Usage

```
## S3 method for class 'knnreg'
extract_params(object, ...)
```

Arguments

`object`	an object of class "knnreg"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- caret::knnreg(mpg ~ cyl + hp + am + gear + carb, data = mtcars)
extracted_params <- extract_params(model)
```

`extract_params.lda` *extract_params.lda*

Description

Extract linear discriminant model parameters from a model created by the lda() function

Usage

```
## S3 method for class 'lda'
extract_params(object, ...)
```

Arguments

`object`	an object of class "lda"
`...`	further arguments passed to or from other methods

Value

PFA as a list-of-lists that can be inserted into a cell or pool

Examples

```
model <- MASS::lda(Species ~ ., data=iris)
model_params <- extract_params(model)
```

`extract_params.naiveBayes`
extract_params.naiveBayes

Description

Extracts a parameters from an ensemble made by the naiveBayes function

Usage

```
## S3 method for class 'naiveBayes'
extract_params(object, threshold = 0.001, eps = 0, ...)
```

Arguments

`object`	an object of class "naiveBayes"
`threshold`	a value replacing cells with probabilities within eps range.
`eps`	a numeric for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by theshold.)
`...`	further arguments passed to or from other methods

Value

a `list` that is extracted from the naiveBayes object

Examples

```
model <- e1071::naiveBayes(Species ~ ., data=iris)
model_params <- extract_params(model)
```

`extract_params.randomForest` *extract_params.randomForest*

Description

Extracts parameters from a forest made by the randomForest function

Usage

```
## S3 method for class 'randomForest'
extract_params(object, which_tree = 1, ...)
```

Arguments

`object`	an object of class "randomForest"
`which_tree`	the number of the tree to extract
`...`	further arguments passed to or from other methods

Value

a `list` that is extracted from the randomForest object

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- factor((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

model <- randomForest::randomForest(Y ~ X1 + X2, data=dat, ntree=10)
my_tree <- extract_params(model, 1)
```

`extract_params.rpart` *extract_params.rpart*

Description

Extracts parameters from a tree made by the rpart() function

Usage

```
## S3 method for class 'rpart'
extract_params(object, ...)
```

Arguments

`object`	an object of class "rpart"
`...`	further arguments passed to or from other methods

Value

a `list` that is extracted from the rpart object

Examples

```
model <- rpart::rpart(Kyphosis ~ Age + as.factor(Number), data = rpart::kyphosis)
my_tree <- extract_params(model)
```

`gen_unique_eng_name` *gen_unique_eng_name*

Description

Convenience or internal function for generating engine names; each call results in a new name.

Usage

```
gen_unique_eng_name()
```

Value

name in the form "Engine_###"

Examples

```
gen_unique_eng_name()
```

`gen_unique_enum_name` *gen_unique_enum_name*

Description

Convenience or internal function for generating enum names; each call results in a new name.

Usage

```
gen_unique_enum_name()
```

Value

name in the form "Enum_###"

Examples

```
gen_unique_enum_name()
```

`gen_unique_fixed_name` *gen_unique_fixed_name*

Description

Convenience or internal function for generating fixed names; each call results in a new name.

Usage

```
gen_unique_fixed_name()
```

Value

name in the form "Fixed_###"

Examples

```
gen_unique_fixed_name()
```

`gen_unique_rec_name` *gen_unique_rec_name*

Description

Convenience or internal function for generating record names; each call results in a new name.

Usage

```
gen_unique_rec_name()
```

Value

name in the form "Record_###"

Examples

```
gen_unique_rec_name()
```

`gen_unique_symb_name` *gen_unique_symb_name*

Description

Convenience or internal function for generating symbol names; each call results in a new name.

Usage

```
gen_unique_symb_name(symbols)
```

Arguments

`symbols`	Symbols

Value

name in the form "tmp_###"

Examples

```
gen_unique_symb_name()
```

`json_array` *json_array*

Description

Convenience function for making a (possibly empty) unnamed list, which converts to a JSON array.

Usage

```
json_array(...)
```

Arguments

`...`	optional contents of the unnamed list

Value

an unnamed list

Examples

```
json_array()
json_array(1, TRUE, "THREE")
```

`json_map` *json_map*

Description

Convenience function for making a (possibly empty) named list, which converts to a JSON object.

Usage

```
json_map(...)
```

Arguments

`...`	optional contents of the named list (as key-value pairs)

Value

a named list

Examples

```
json_map()
json_map(one = 1, two = TRUE, three = "THREE")
```

`pfa` *Generate PFA Document from Object*

Description

pfa is a generic function for generating valid PFA documents from the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

Usage

```
pfa(object, name = NULL, version = NULL, doc = NULL, metadata = NULL,
  randseed = NULL, options = NULL, ...)
```

Arguments

`object`	a model object for which a PFA document is desired
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes

`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose a valid PFA document

See Also

`pfa.lm` `pfa.glm`

Examples

```
# all the "pfa" methods found
methods("pfa")
```

`pfa.Arima` *PFA Formatting of ARIMA Models*

Description

This function takes an ARIMA model created using the arima(), Arima(), or auto.arima() functions and returns a list-of-lists representing in valid PFA document that could be used for scoring.

Usage

```
## S3 method for class 'Arima'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, cycle_reset = TRUE,
  ...)
```

Arguments

`object`	an object of class "Arima"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one

`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`cycle_reset`	a logical indicating whether to reset the state back to the last point of the trained model before forecasting or to continue cycling forward through trend and seasonality with every new call to the engine. The default is TRUE so that repeated calls yield the same forecast as repeated calls to `predict` or `forecast`.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`Arima` `auto.arima` `arima` `extract_params.Arima`

Examples

```
model <- forecast::Arima(USAccDeaths, order=c(2,2,2), seasonal=c(0,2,2))
model_as_pfa <- pfa(model)

# with regressors
n <- 100
ext_dat <- data.frame(x1=rnorm(n), x2=rnorm(n))
x <- stats::arima.sim(n=n, model=list(ar=0.4)) + 2 + 0.8*ext_dat[,1] + 1.5*ext_dat[,2]
model <- stats::arima(x, order=c(1,0,0), xreg = ext_dat)
model_as_pfa <- pfa(model)
```

`pfa.cv.glmnet` *PFA Formatting of Fitted glmnet objects*

Description

This function takes a generalized linear model fit using cv.glmnet and returns a list-of-lists representing a valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'cv.glmnet'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  lambda = object[["lambda.1se"]], pred_type = c("response", "prob"),
  cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "cv.glmnet"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`lambda`	a numeric value of the penalty parameter lambda at which coefficients are required
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R

See Also

`glmnet` `extract_params.glmnet`

Examples

```
X <- matrix(c(rnorm(100), runif(100)), nrow=100, ncol=2)
Y <- factor(3 - 5 * X[,1] + 3 * X[,2] + rnorm(100, 0, 3) > 0)

model <- glmnet::cv.glmnet(X, Y, family = 'binomial')
model_as_pfa <- pfa(model)
```

pfa.ets — *PFA Formatting of Fitted Exponential Smoothing State Space Models*

Description

This function takes an Exponential smoothing state space model created using the ets() function from the forecast package and returns a list-of-lists representing in valid PFA document that could be used for scoring.

Usage

```
## S3 method for class 'ets'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, cycle_reset = TRUE,
  ...)
```

Arguments

`object`	an object of class "ets"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`cycle_reset`	a logical indicating whether to reset the state back to the last point of the trained model before forecasting or to continue cycling forward through trend and seasonality with every new call to the engine. The default is TRUE so that repeated calls yield the same forecast as repeated calls to `forecast`.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`ets` `extract_params.ets`

Examples

```
model <- forecast::ets(USAccDeaths, model="ZZZ")
model_as_pfa <- pfa(model)
```

---

pfa.forecast *PFA Formatting of Time Series Models Fit using Forecast Package*

Description

This function takes model with class "forecast" created using the forecast package and returns a list-of-lists representing in valid PFA document that could be used for scoring.

Usage

```
## S3 method for class 'forecast'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, ...)
```

Arguments

`object`	an object of class "forecast"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`holt` `ses` `hw` `pfa.ets`

Examples

```
model1 <- forecast::holt(airmiles)
model1_as_pfa <- pfa(model1)

model2 <- forecast::hw(USAccDeaths,h=48)
model2_as_pfa <- pfa(model2)

model3 <- forecast::ses(LakeHuron)
model3_as_pfa <- pfa(model3)
```

`pfa.gbm` *PFA Formatting of Fitted GBMs*

Description

This function takes a gradient boosted machine (gbm) fit using gbm and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'gbm'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL, n.trees = NULL, ...)
```

Arguments

`object`	an object of class "gbm"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes

`n.trees`	an integer or vector of integers specifying the number of trees to use in building the model. If a vector is provided, then only the indices of thos trees will be used. If a single integer is provided then all trees up until and including that index will be used.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`gbm`

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- ((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

bernoulli_model <- gbm::gbm(Y ~ X1 + X2,
                            data = dat,
                            distribution = 'bernoulli')
model_as_pfa <- pfa(bernoulli_model)
```

`pfa.glm` *PFA Formatting of Fitted GLMs*

Description

This function takes a generalized linear model fit using glm and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'glm'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "glm"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`glm` `extract_params.glm`

Examples

```
X1 <- rnorm(100)
X2 <- runif(100)
Y <- 3 - 5 * X1 + 3 * X2 + rnorm(100, 0, 3)
Y <- Y > 0

glm_model <- glm(Y ~ X1 + X2, family = binomial(logit))
model_as_pfa <- pfa(glm_model)
```

pfa.glmnet *PFA Formatting of Fitted glmnet objects*

Description

This function takes a generalized linear model fit using glmnet and returns a list-of-lists representing a valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'glmnet'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, lambda = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "glmnet"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`lambda`	a numeric value of the penalty parameter lambda at which coefficients are required
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

glmnet extract_params.glmnet

Examples

```
X <- matrix(c(rnorm(100), runif(100)), nrow=100, ncol=2)
Y <- factor(3 - 5 * X[,1] + 3 * X[,2] + rnorm(100, 0, 3) > 0)

model <- glmnet::glmnet(X, Y, family = 'binomial')
model_as_pfa <- pfa(model)
```

---

pfa.HoltWinters *PFA Formatting of Fitted Holt Winters Models*

Description

This function takes a Holt Winters model fit using HoltWinters() and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'HoltWinters'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, ...)
```

Arguments

`object`	an object of class "HoltWinters"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`HoltWinters` `extract_params.HoltWinters`

Examples

```
model <- HoltWinters(co2)
model_as_pfa <- pfa(model)
```

`pfa.ipredknn` *PFA Formatting of Fitted knns*

Description

This function takes a k-nearest neighbor fit using ipredknn and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'ipredknn'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL,
  distance_measure = c("euclidean", "manhattan", "angle", "jaccard",
  "ejaccard"), ...)
```

Arguments

`object`	an object of class "ipredknn"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer

`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes.
`distance_measure`	a string representing the type of distance calculation in order to determine the nearest neighbours.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`ipredknn` `extract_params.knn3`

Examples

```
iris2 <- iris
colnames(iris2) <- gsub('\\.', '_', colnames(iris2))
model <- ipred::ipredknn(Species ~ ., iris2)
model_as_pfa <- pfa(model)
```

`pfa.kcca` *PFA Formatting of Fitted K-Centroid Models*

Description

This function takes a K-centroids model fit using kcca and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'kcca'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  cluster_names = NULL, ...)
```

Arguments

`object`	an object of class "kcca"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`cluster_names`	a character vector of length k to name the values relating to each cluster instead of just an integer. If not specified, then the predicted cluster will be the string representation of the cluster index.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`kcca` `extract_params.kcca`

Examples

```
model <- flexclust::kcca(iris[,1:4], k = 3, family=flexclust::kccaFamily("kmeans"))
model_as_pfa <- pfa(model)
```

`pfa.kccasimple` *PFA Formatting of Fitted K-Centroid Models*

Description

This function takes a K-centroids model fit using kccasimple and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'kccasimple'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  cluster_names = NULL, ...)
```

Arguments

`object`	an object of class "kccasimple"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`cluster_names`	a character vector of length k to name the values relating to each cluster instead of just an integer. If not specified, then the predicted cluster will be the string representation of the cluster index.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`kcca` `extract_params.kccasimple`

Examples

```
model <- flexclust::kcca(iris[,1:4], k = 3,
                         family=flexclust::kccaFamily("kmeans"), simple=TRUE)
model_as_pfa <- pfa(model)
```

pfa.kmeans *PFA Formatting of Fitted K-means Models*

Description

This function takes a K-means model fit using kmeans and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'kmeans'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  cluster_names = NULL, ...)
```

Arguments

`object`	an object of class "kmeans"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`cluster_names`	a character vector of length k to name the values relating to each cluster instead of just an integer. If not specified, then the predicted cluster will be the string representation of the cluster index.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R kcca.R

See Also

`kmeans` `pfa.kcca`

Examples

```
model <- kmeans(x=iris[, 1:2], centers=3)
model_as_pfa <- pfa(model)
```

---

pfa.knn3 *PFA Formatting of Fitted knns*

Description

This function takes a k-nearest neighbor fit using knn3 and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'knn3'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL,
  distance_measure = c("euclidean", "manhattan", "angle", "jaccard",
  "ejaccard"), ...)
```

Arguments

`object`	an object of class "knn3"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes.
`distance_measure`	a string representing the type of distance calculation in order to determine the nearest neighbours.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`knn3` `extract_params.knn3`

Examples

```
iris2 <- iris
colnames(iris2) <- gsub('\\.', '_', colnames(iris2))
model <- caret::knn3(Species ~ ., iris2)
model_as_pfa <- pfa(model)
```

`pfa.knnreg` *PFA Formatting of Fitted knns*

Description

This function takes a k-nearest neighbor fit using knnreg and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'knnreg'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL,
  distance_measure = c("euclidean", "manhattan", "angle", "jaccard",
  "ejaccard"), ...)
```

Arguments

`object`	an object of class "knnreg"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one

`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes.
`distance_measure`	a string representing the type of distance calculation in order to determine the nearest neighbours.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`knnreg` `extract_params.knn3`

Examples

```
model <- caret::knnreg(mpg ~ cyl + hp + am + gear + carb, data = mtcars)
model_as_pfa <- pfa(model)
```

`pfa.lda` *PFA Formatting of Fitted Linear Discriminant Models*

Description

This function takes a linear discriminant model fit using lda and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'lda'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  prior = object$prior, dimen = length(object$svd), method = c("plug-in"),
  pred_type = c("response", "prob"), cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "lda"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`prior`	a named vector specifying the prior probabilities of class membership. If unspecified, the class proportions for the training set are used.
`dimen`	an integer specifying the dimension of the space to be used. If this is less than min(p input variables, number of classes - 1) then the first N number of dimensions will be used in the calculation
`method`	a character string indicating the prediction method. Currently, only the plug-in method is supported.
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`lda` `extract_params.lda`

Examples

```
model <- MASS::lda(Species ~ ., data=iris)
model_as_pfa <- pfa(model)
```

`pfa.lm` *PFA Formatting of Fitted Linear models*

Description

This function takes a linear model fit using lm and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'lm'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, ...)
```

Arguments

`object`	an object of class "lm"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa.config.R avro.typemap.R avro.R

See Also

`lm` `pfa.glm`

Examples

```
X1 <- rnorm(100)
X2 <- runif(100)
Y <- 3 - 5 * X1 + 3 * X2 + rnorm(100, 0, 3)

model <- lm(Y ~ X1 + X2)
model_as_pfa <- pfa(model)
```

`pfa.naiveBayes` *PFA Formatting of Fitted naiveBayess*

Description

This function takes a Naive Bayes model fit using naiveBayes() and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'naiveBayes'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL, threshold = 0.001,
  eps = 0, pred_type = c("response", "prob"), cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "naiveBayes"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`threshold`	a value replacing cells with probabilities within eps range.
`eps`	a numeric for specifying an epsilon-range to apply laplace smoothing (to replace zero or close-zero probabilities by theshold.)
`pred_type`	a string with value "response" for returning the predicted class or the value "prob", which for returns the predicted probability of each class.
`cutoffs`	A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`naiveBayes`

Examples

```
model <- e1071::naiveBayes(Species ~ ., data=iris)
model_as_pfa <- pfa(model)
```

`pfa.randomForest` *PFA Formatting of Fitted Random Forest Models*

Description

This function takes a random forest model fit using randomForest() and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'randomForest'
pfa(object, name = NULL, version = NULL,
  doc = NULL, metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL, n.trees = NULL, ...)
```

Arguments

`object`	an object of class "randomForest"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer

`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`n.trees`	an integer or vector of integers specifying the number of trees to use in building the model. If a vector is provided, then only the indices of thos trees will be used. If a single integer is provided then all trees up until and including that index will be used.
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro_R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`randomForest`

Examples

```
dat <- data.frame(X1 = runif(100),
                  X2 = rnorm(100))
dat$Y <- factor((rexp(100,5) + 5 * dat$X1 - 4 * dat$X2) > 0)

model <- randomForest::randomForest(Y ~ X1 + X2, data=dat, ntree=10)
model_as_pfa <- pfa(model)
```

`pfa.rpart` *PFA Formatting of Fitted rpart Tree Models*

Description

This function takes an tree model fit using the rpart package and returns a list-of-lists representing in valid PFA document that could be used for scoring

Usage

```
## S3 method for class 'rpart'
pfa(object, name = NULL, version = NULL, doc = NULL,
  metadata = NULL, randseed = NULL, options = NULL,
  pred_type = c("response", "prob"), cutoffs = NULL, ...)
```

Arguments

`object`	an object of class "rpart"
`name`	a character which is an optional name for the scoring engine
`version`	an integer which is sequential version number for the model
`doc`	a character which is documentation string for archival purposes
`metadata`	a `list` of strings that is computer-readable documentation for archival purposes
`randseed`	a integer which is a global seed used to generate all random numbers. Multiple scoring engines derived from the same PFA file have different seeds generated from the global one
`options`	a `list` with value types depending on option name Initialization or runtime options to customize implementation (e.g. optimization switches). May be overridden or ignored by PFA consumer
`pred_type`	a string with value "response" for returning a prediction on the same scale as what was provided during modeling, or value "prob", which for classification problems returns the probability of each class.
`cutoffs`	(Classification only) A named numeric vector of length equal to number of classes. The "winning" class for an observation is the one with the maximum ratio of predicted probability to its cutoff. The default cutoffs assume the same cutoff for each class that is 1/k where k is the number of classes
`...`	additional arguments affecting the PFA produced

Value

a `list` of lists that compose valid PFA document

Source

pfa_config.R avro_typemap.R avro.R pfa_cellpool.R pfa_expr.R pfa_utils.R

See Also

`rpart`

Examples

```
model <- rpart::rpart(Species ~ ., data=iris)
model_as_pfa <- pfa(model)
```

pfa_cell *pfa_cell*

Description

Creates a list of lists representing a PFA cell.

Usage

```
pfa_cell(type, init, source = "embedded", shared = FALSE,
  rollback = FALSE)
```

Arguments

type	cell type, which is an Avro schema as list of lists (created by avro_* functions)
init	cell initial value, which is a list of lists, usually converted from a model
source	if "embedded", the init is the data structure, if "json", the init is a URL string pointing to an external JSON file
shared	if TRUE, the cell is shared across scoring engine instances
rollback	if TRUE, the cell's value would be rolled back if an uncaught exception is encountered

Value

a list of lists that can be inserted into pfa_config.

Examples

```
pfa_cell(avro_double, 12)
```

pfa_document *pfa_document*

Description

Create a complete PFA document as a list-of-lists. Composing with the JSON function creates a PFA file on disk.

Usage

```
pfa_document(input, output, action, cells = NULL, pools = NULL,
  fcns = NULL, validate = FALSE, name = NULL, method = NULL,
  begin = NULL, end = NULL, zero = NULL, merge = NULL,
  randseed = NULL, doc = NULL, version = NULL, metadata = NULL,
  options = NULL, env = parent.frame())
```

Arguments

`input`	input schema, which is an Avro schema as list-of-lists (created by avro_* functions)
`output`	output schema, which is an Avro schema as list-of-lists (created by avro_* functions)
`action`	R commands wrapped as an expression (see R's built-in expression function)
`cells`	named list of cell specifications (see the pfa_cell function)
`pools`	named list of pool specifications (see the pfa_cell function)
`fcns`	named list of R commands, wrapped as expressions
`validate`	a logical indicating whether the generated PFA document should be validated using Titus-in-Aurelius function `pfa_engine`
`name`	optional name for the scoring engine (string)
`method`	"map", "emit", "fold", or NULL (for "map")
`begin`	R commands wrapped as an expression
`end`	R commands wrapped as an expression
`zero`	list-of-lists representing the initial value for a "fold"'s tally
`merge`	R commands wrapped as an expression
`randseed`	optional random number seed (integer) for ensuring that the scoring engine is deterministic
`doc`	optional model documentation string
`version`	optional model version number (integer)
`metadata`	optional named list of strings to store model metadata
`options`	optional list-of-lists to specify PFA options
`env`	environment for resolving unrecognized symbols as substitutions

Value

a `list` of lists representing a complete PFA document

Source

pfa_engine.R

See Also

`pfa_engine`

Examples

```
pfa_document(avro_double, avro_double, expression(input + 10))
```

pfa_engine *pfa_engine*

Description

Create an executable PFA scoring engine in R by calling Titus through rPython. If this function is successful, then the PFA is valid (only way to check PFA validity in R).

Usage

```
pfa_engine(doc)
```

Arguments

doc	`list` of lists representing a complete PFA document

Source

json.R

Examples

```
## Not run:
my_pfa_doc <- pfa_document(avro_double, avro_double, expression(input + 10))
pfa_engine(my_pfa_doc)   # requres rPython and Titus to be installed

## End(Not run)
```

pfa_expr *pfa_expr*

Description

Convert a quoted R expression into a `list` of lists that can be inserted into PFA

Usage

```
pfa_expr(expr, symbols = list(), cells = list(), pools = list(),
  fcns = list(), env = parent.frame())
```

Arguments

expr	quoted R expression (e.g. quote(2 + 2))
symbols	list of symbol names that would be in scope when evaluating this expression
cells	list of cell names that would be in scope when evaluating this expression
pools	list of pool names that would be in scope when evaluating this expression
fcns	list of function names that would be in scope when evaluating this expression
env	environment for resolving unrecognized symbols as substitutions

Value

a `list` of lists representing a fragment of a PFA document

Examples

```
pfa_expr(quote(2 + 2))
```

`pfa_pool` *pfa_pool*

Description

Creates a `list` of lists representing a PFA pool.

Usage

```
pfa_pool(type, init, source = "embedded", shared = FALSE,
  rollback = FALSE)
```

Arguments

`type`	pool type, which is an Avro schema as `list` of lists (created by avro_* functions)
`init`	pool initial value, which is a `list` of lists, usually converted from a model
`source`	if "embedded", the init is the data structure, if "json", the init is a URL string pointing to an external JSON file
`shared`	if TRUE, the pool is shared across scoring engine instances
`rollback`	if TRUE, the pool's value would be rolled back if an uncaught exception is encountered

Value

a `list` of lists that can be inserted into pfa_config.

Examples

```
pfa_pool(avro_double, json_map(one = 1.1, two = 2.2, three = 3.3))
```

read_pfa *read_pfa*

Description

Convert a JSON string in memory or a JSON file on disk into a list-of-lists structure.

Usage

```
read_pfa(x)
```

Arguments

x	A string, file or url connection

Value

a `list` of lists structure in which null -> NULL, true -> TRUE, false -> FALSE, numbers -> numeric, strings -> character, array -> list, object -> named list

Examples

```
# literal JSON string  (useful for small examples)
toy_model <- read_pfa('{"input":"double","output":"double","action":[{"+":["input",10]}]}')

# from a local path, must be wrapped in "file" command to create a connection
local_model <- read_pfa(file(system.file("extdata", "my-model.pfa", package = "aurelius")))

# from a url (split on two lines so not to exceed 100 char wide during install)
url_model <- read_pfa(url(paste0('https://raw.githubusercontent.com/ReportMort/hadrian',
                '/feature/add-r-package-structure/aurelius/inst/extdata/my-model.pfa')))
```

write_pfa *write_pfa*

Description

Convert a PFA list of lists into a JSON string in memory or a JSON file on disk.

Usage

```
write_pfa(doc, file = "", force = TRUE, auto_unbox = TRUE,
  pretty = FALSE, digits = 8, ...)
```

Arguments

`doc`	The document to convert.
`file`	a string representing file path to write to. If ” then the string of JSON is returned
`force`	a logical indicating to unclass/skip objects of classes with no defined JSON mapping
`auto_unbox`	a logical indicating to automatically unbox all atomic vectors of length 1
`pretty`	a logical indicating to add indentation whitespace to JSON output.
`digits`	max number of decimal digits to print for numeric values. Use I() to specify significant digits. Use NA for max precision.
`...`	additional arguments passed to toJSON

Examples

```
## Not run:
my_pfa_doc <- pfa_document(avro_double, avro_double, expression(input + 10))
write_pfa(my_pfa_doc)
write_pfa(my_pfa_doc, file = "my-model.pfa")

## End(Not run)
```

Index