

# Package ‘ikde’

December 18, 2018

**Title** Iterative Kernel Density Estimation

**Version** 0.0.1

**Description** Estimation of model marginal likelihoods for Bayesian model selection using iterative kernel density estimation. A multitude of methods exist for performing model selection in general and estimating marginal likelihood in specific, but none are particularly well-suited to large models (such as Gaussian processes) applied to relatively limited datasets. Methods are provided to specify and construct Stan models, estimate those models, and estimate the marginal likelihood of those models.

**Maintainer** Taylor McKenzie <tkmckenzie@gmail.com>

**Depends** R (>= 3.5.0),  
rstan

**Imports** invgamma,  
methods,  
mvtnorm,  
quantreg,  
stats

**Suggests** ggplot2 (>= 3.1.0),  
knitr (>= 1.20),  
Rdpack (>= 0.10.1),  
rmarkdown (>= 1.10)

**VignetteBuilder** knitr

**License** GPL-3

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**RdMacros** Rdpack

## R topics documented:

build.model . . . . .	2
create.restricted.models . . . . .	3
define.model . . . . .	4
evaluate.expression . . . . .	6
evaluate.likelihood . . . . .	7
evaluate.marginal.likelihood . . . . .	8
evaluate.posterior . . . . .	9

evaluate.priors . . . . .	10
evaluate.statement . . . . .	11
fit.model . . . . .	12
gibbs.lm . . . . .	13
lm.generated . . . . .	15
prostatic.nodes . . . . .	15
stan.dist.to.r.dist . . . . .	16
stan.operator.to.r.operator . . . . .	17
%stan*% . . . . .	17

<b>Index</b>	<b>19</b>
--------------	-----------

---

build.model	<i>Build Stan model</i>
-------------	-------------------------

---

## Description

Builds and compiles a defined Stan model

## Usage

```
build.model(ikde.model)
```

## Arguments

ikde.model      An object of class ikde.model, e.g., from define.model

## Details

Builds Stan model using defined ikde.model, then compiles the model and stores DSO for fast running.

## Value

Returns an ikde.model object with the following elements

data	A list of data passed to the Stan program
transformed.data	A list describing data transformations for the Stan program to perform
parameters	A list of parameters used in the Stan program
transformed.parameters	A list describing parameter transformations for the Stan program to perform
model	A list describing the Stan model
stan.code	Stan code for the model
stan.data	Data passed to Stan for estimation
stan.dso	DSO for Stan model, allows Stan to run model without recompilation
built	Boolean indicating whether the model has been built
density.variable	List containing two elements: "name" of the variable on which density estimation should be performed on, and "value" indicating the point at which density should be estimated

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
             k = list("int<lower=1>", ncol(X)),
             X = list("matrix[N, k]", X),
             y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                   sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                       "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)
ikde.model <- build.model(ikde.model)

cat(ikde.model$stan.code)
```

---

```
create.restricted.models
```

*Creates restricted models for IKDE*

---

## Description

Creates set of restricted models to be used for posterior density estimation

## Usage

```
create.restricted.models(ikde.model, eval.point)
```

## Arguments

<code>ikde.model</code>	An object of class <code>ikde.model</code> , does not necessarily have to be built
<code>eval.point</code>	A list of parameter names and the point to evaluate densities

## Details

Posterior density can be estimated by breaking the multi-dimensional density into one-dimensional components. This method creates restricted models from which conditional densities can be estimated. Each real parameter and each entry of vector parameters are restricted one at a time, with values restricted at the specified point.

## Value

Returns a list of built `ikde.models` for each restricted model

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
             k = list("int<lower=1>", ncol(X)),
             X = list("matrix[N, k]", X),
             y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                   sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                       "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)
eval.point <- list(beta = c(1, 2, 3, 4),
                  sigma = 5)

ikde.model.list <- create.restricted.models(ikde.model, eval.point)
for (restricted.ikde.model in ikde.model.list){
  cat(restricted.ikde.model$stan.code)
  cat("-----\n")
}
```

---

define.model

*Define Stan model*


---

## Description

Defines Stan model and stores input data

## Usage

```
define.model(data, parameters, model, transformed.data = list(),
             transformed.parameters = list())
```

## Arguments

data	A list of data passed to the Stan program. Should be of the form <code>list(data.name = list(data.type, data.object))</code> .
parameters	A list of parameters used in the Stan program. Should be of the form <code>list(parameter.name = parameter.type)</code> .
model	A list describing the Stan model. Should be a list with components "priors" and "likelihood".
transformed.data	A list describing data transformations for the Stan program to perform. Should be of the form <code>list(variable.name = list(variable.type, variable.expression))</code> .

transformed.parameters

A list describing parameter transformations for the Stan program to perform.  
Should be of the form `list(variable.name = list(variable.type, variable.expression))`.

## Details

Defines inputs to be used for building and eventually fitting Stan model.

## Value

Returns an `ikde.model` object with the following elements

<code>data</code>	A list of data passed to the Stan program
<code>transformed.data</code>	A list describing data transformations for the Stan program to perform
<code>parameters</code>	A list of parameters used in the Stan program
<code>transformed.parameters</code>	A list describing parameter transformations for the Stan program to perform
<code>model</code>	A list describing the Stan model
<code>stan.code</code>	Stan code for the model
<code>stan.data</code>	Data passed to Stan for estimation
<code>stan.dso</code>	DSO for Stan model, allows Stan to run model without recompilation
<code>built</code>	Boolean indicating whether the model has been built
<code>density.variable</code>	List containing two elements: "name" of the variable on which density estimation should be performed on, and "value" indicating the point at which density should be estimated

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
            k = list("int<lower=1>", ncol(X)),
            X = list("matrix[N, k]", X),
            y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                  sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                      "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)
```

---

evaluate.expression	<i>Evaluate expression from Stan program</i>
---------------------	--

---

## Description

Evaluate expression from Stan program

## Usage

```
evaluate.expression(stan.expression, ...)
```

## Arguments

stan.expression	String representing Stan expression. All variables must be passed in ... .
...	Any variables present in the parent environment that are needed to evaluate stan.expression

## Details

First, all variables specified in ... are loaded into the function environment. Then, all multiplication is replaced by

## Value

The result of the Stan expression

## Examples

```
X <- matrix(1:9, nrow = 3)
b <- c(4, 5, 6)

stan.expression <- "(3 + 2) * X * (5 * b)"

# These results match:
evaluate.expression(stan.expression)
print((3 + 2) * X %*% (5 * b))
#      [,1]
# [1,] 1650
# [2,] 2025
# [3,] 2400
```

---

evaluate.likelihood	<i>Stan model likelihood evaluation</i>
---------------------	---

---

## Description

Evaluates likelihood of Stan model at specified evaluation point

## Usage

```
evaluate.likelihood(ikde.model, eval.point)
```

## Arguments

ikde.model	An object of class ikde.model which has been built
eval.point	A list of parameter names and the point to evaluate the likelihood

## Details

Parses sampling statements in ikde.model\$model\$likelihood and evaluates them at the specified evaluation point.

## Value

A real number indicating value of the log-likelihood at the specified evaluation point

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
            k = list("int<lower=1>", ncol(X)),
            X = list("matrix[N, k]", X),
            y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                  sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                      "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)

eval.point <- list(beta = c(1, 2, 3, 4), sigma_sq = 5)

# These results match:
evaluate.likelihood(ikde.model, eval.point)
sum(dnorm(y, X %*% eval.point$beta, eval.point$sigma_sq, log = TRUE))
# [1] -1054.093
```

---

```
evaluate.marginal.likelihood
```

*Stan model marginal likelihood evaluation*

---

## Description

Evaluates marginal likelihood of Stan model at the posterior mean

## Usage

```
evaluate.marginal.likelihood(ikde.model, burn.iter = 1000,
  sample.iter = 1000, control = NULL, refresh = NULL,
  display.output = FALSE, show.trace = FALSE)
```

## Arguments

<code>ikde.model</code>	An object of class <code>ikde.model</code> , does not necessarily have to be built
<code>burn.iter</code>	Number of warmup iterations
<code>sample.iter</code>	Number of sampling iterations
<code>control</code>	Control parameters used in the Markov chain. See <code>?rstan::stan</code> for details.
<code>refresh</code>	How frequently should progress be reported, in numbers of iterations
<code>display.output</code>	Boolean indicating whether output from <code>rstan::stan</code> should be printed
<code>show.trace</code>	Boolean indicating whether to show trace plots

## Details

Uses `evaluate.likelihood`, `evaluate.priors`, and `evaluate.posterior` to form an estimate of marginal likelihood at the posterior mean.

## Value

A real number indicating value of the log-marginal-likelihood at the posterior mean

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
  k = list("int<lower=1>", ncol(X)),
  X = list("matrix[N, k]", X),
  y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
  sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
  "sigma_sq ~ inv_gamma(1, 1)"),
  likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))
```



```

ikde.model <- define.model(data, parameters, model)

# Only an estimation, may not exactly match presented result
evaluate.marginal.likelihood(ikde.model)
# [1] -388.9264

```

---

evaluate.posterior	<i>Stan model posterior evaluation</i>
--------------------	--

---

## Description

Evaluates posterior of Stan model at specified evaluation point

## Usage

```

evaluate.posterior(ikde.model, eval.point, burn.iter = 1000,
  sample.iter = 1000, control = NULL, refresh = NULL,
  display.output = FALSE, show.trace = FALSE)

```

## Arguments

<code>ikde.model</code>	An object of class <code>ikde.model</code> , does not necessarily have to be built
<code>eval.point</code>	A list of parameter names and the point to evaluate the posterior
<code>burn.iter</code>	Number of warmup iterations
<code>sample.iter</code>	Number of sampling iterations
<code>control</code>	Control parameters used in the Markov chain. See <code>?rstan::stan</code> for details.
<code>refresh</code>	How frequently should progress be reported, in numbers of iterations
<code>display.output</code>	Boolean indicating whether output from <code>rstan::stan</code> should be printed
<code>show.trace</code>	Boolean indicating whether to show trace plots

## Details

Uses list of `ikde.model` objects created by `create.restricted.models` to estimate posterior density. Each `ikde.model` is fit, then conditional posterior density is estimated at the specified point.

## Value

A real number indicating value of the log-posterior at the specified evaluation point

## Examples

```

data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
  k = list("int<lower=1>", ncol(X)),

```

```

      X = list("matrix[N, k]", X),
      y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                  sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                      "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)
ikde.model <- build.model(ikde.model)
stan.fit <- fit.model(ikde.model)
stan.extract <- rstan::extract(stan.fit)

eval.point <- list(beta = apply(stan.extract$beta, 2, mean),
                  sigma_sq = mean(stan.extract$sigma_sq))

# Only an estimation, may not exactly match presented result
evaluate.posterior(ikde.model, eval.point)
# [1] -1.889711

```

---

evaluate.priors

*Stan model prior evaluation*


---

## Description

Evaluates prior of Stan model at specified evaluation point

## Usage

```
evaluate.priors(ikde.model, eval.point)
```

## Arguments

<code>ikde.model</code>	An object of class <code>ikde.model</code> which has been built
<code>eval.point</code>	A list of parameter names and the point to evaluate priors

## Details

Parses sampling statements in `ikde.model$model$priors` and evaluates them at the specified evaluation point.

## Value

A real number indicating value of the log-prior at the evaluation point

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
             k = list("int<lower=1>", ncol(X)),
             X = list("matrix[N, k]", X),
             y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                   sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                       "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)

eval.point <- list(beta = c(1, 2, 3, 4), sigma_sq = 5)

# These results match:
evaluate.priors(ikde.model, eval.point)
sum(dnorm(eval.point$beta, 0, 10, log = TRUE),
    invgamma::dinvgamma(eval.point$sigma_sq, 1, 1, log = TRUE))
# [1] -16.45497
```

---

evaluate.statement	<i>Evaluate sampling statement from Stan program</i>
--------------------	--

---

## Description

Evaluate sampling statement from Stan program

## Usage

```
evaluate.statement(statement, ikde.model, eval.point)
```

## Arguments

statement	A string containing a sampling statement
ikde.model	An object of class ikde.model, which has been built
eval.point	A list of parameter names and the point to evaluate the statement

## Details

Parses the given sampling statement and evaluates it at the specified evaluation point. The ikde.model object and eval.point object are needed to resolve variable values in the statement.

## Value

A real number indicating value of the log-density of the statement at the evaluation point

## Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
             k = list("int<lower=1>", ncol(X)),
             X = list("matrix[N, k]", X),
             y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                   sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                       "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)

statement <- ikde.model$model$likelihood[1]
eval.point <- list(beta = c(1, 2, 3, 4), sigma_sq = 5)

# These results match:
evaluate.statement(statement, ikde.model, eval.point)
sum(dnorm(y, mean = X %*% eval.point$beta, sd = sqrt(eval.point$sigma_sq), log = TRUE))
# [1] -4178.641
```

---

fit.model

*Fits Stan model*


---

## Description

Uses a built ikde.model to draw samples from posterior distribution using Stan

## Usage

```
fit.model(ikde.model, burn.iter = 1000, sample.iter = 1000,
          chains = 1, control = NULL, refresh = NULL,
          display.output = FALSE)
```

## Arguments

ikde.model	An object of class ikde.model which has been built
burn.iter	Number of warmup iterations
sample.iter	Number of sampling iterations
chains	Number of independent chains to use
control	Control parameters used in the Markov chain. See ?rstan::stan for details.
refresh	How frequently should progress be reported, in numbers of iterations
display.output	Boolean indicating whether output from rstan::stan should be printed

**Details**

Takes a built ikde.model object, which contains model DSO, and fits the model using rstan::stan.

**Value**

An object of S4 class stanfit. See rstan::stan for more details.

**Examples**

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

data <- list(N = list("int<lower=1>", nrow(X)),
             k = list("int<lower=1>", ncol(X)),
             X = list("matrix[N, k]", X),
             y = list("vector[N]", y))
parameters <- list(beta = "vector[k]",
                   sigma_sq = "real<lower=0>")
model <- list(priors = c("beta ~ normal(0, 10)",
                       "sigma_sq ~ inv_gamma(1, 1)"),
             likelihood = c("y ~ normal(X * beta, sqrt(sigma_sq))"))

ikde.model <- define.model(data, parameters, model)
ikde.model <- build.model(ikde.model)
stan.fit <- fit.model(ikde.model)
stan.extract <- extract(stan.fit)

# Only an estimation, may not exactly match presented result
print(apply(stan.extract$beta, 2, mean))
# [1] 3.199021 1.620546 4.489716 1.226508
```

---

gibbs.lm

---

*Linear model Gibbs sampling*


---

**Description**

Fits a linear model using Gibbs sampling and estimates marginal likelihood as in Chib (1995)

**Usage**

```
gibbs.lm(X, y, priors = list(), burn.iter = 1000, sample.iter = 1000)
```

**Arguments**

X	Matrix of input variables
y	Vector of output variables
priors	A named list of parameter priors; should include beta.prior.mean (vector), beta.prior.var (matrix), sigma_sq.prior.shape (scalar), and sigma_sq.prior.rate (scalar)

<code>burn.iter</code>	Number of warmup iterations
<code>sample.iter</code>	Number of sampling iterations

### Details

Uses a standard formulation of a linear model from which a Gibbs sampler can be derived. Specifically, for a model of the form

$$\begin{aligned}\beta &\sim N(\mu_\beta, \Sigma_\beta) \\ \sigma^2 &\sim \Gamma(s_\sigma, r_\sigma) \\ y &= X\beta + \varepsilon \\ \varepsilon &\sim N\left(0, \frac{1}{\sqrt{\sigma^2}}I\right),\end{aligned}$$

Gibbs sampling can be performed using the conditional distributions

$$\begin{aligned}\beta|\sigma^2, X, y &\sim N(\tilde{\mu}_\beta, \tilde{\Sigma}_\beta) \\ \sigma^2|\beta, X, y &\sim \Gamma^{-1}\left(\frac{N}{2} + s_\sigma, \frac{e'e}{2} + r_\sigma\right),\end{aligned}$$

where  $N$  is the number of observations and

$$\begin{aligned}\tilde{\Sigma}_\beta &= \frac{X'X}{\sigma^2} + \Sigma_\beta^{-1} \\ \tilde{\mu}_\beta &= \tilde{\Sigma}_\beta \left( \frac{X'y}{\sigma^2} + \Sigma_\beta^{-1}\mu_\beta \right) \\ e &= y - X\beta.\end{aligned}$$

### Value

Returns an list with the following elements

<code>samples</code>	Named list of samples from the posterior, with elements "beta" and "sigma.sq"
<code>log.marginal</code>	Estimate of the model's log-marginal-likelihood
<code>priors</code>	List of priors used for the model

### Examples

```
data(lm.generated)

X <- lm.generated$X
y <- lm.generated$y

gibbs.fit <- gibbs.lm(X, y,
  priors = list(beta.prior.mean = rep(0, 4),
    beta.prior.var = 100 * diag(4),
    sigma.sq.prior.shape = 1,
    sigma.sq.prior.rate = 1))

print(apply(gibbs.fit$samples$beta, 2, mean)) # [1] 3.181184 1.643960 4.480879 1.213804
print(mean(gibbs.fit$samples$sigma.sq)) # [1] 97.52314
print(gibbs.fit$log.marginal) # [1] -389.001
```

lm.generated

*Randomly generated multivariate linear model data***Description**

A dataset for estimation of linear models

**Usage**

```
lm.generated
```

**Format**

A list with two components:

**X** Matrix of independent variables

**y** Vector of dependent variable observations

**Details**

Generated with the following code:

```
set.seed(100)

N <- 100
k <- 4
sd <- 10

X <- cbind(1, matrix(runif(N * (k - 1), -10, 10), ncol = k - 1))
beta <- runif(k, -5, 5)
y <- X
y <- c(y)
```

prostatic.nodes

*Prostatic nodal development data***Description**

A dataset replicated from Chib (1995) indicating presence of prostatic nodal development among patients prostate cancer

**Usage**

```
prostatic.nodes
```

**Format**

A data.frame with 53 observations of 7 variables:

**Case** Patient identifier

**y** Binary outcome indicating nodal development

**X.1** Explanatory variable

**X.2** Explanatory variable

**X.3** Binary explanatory variable

**X.4** Binary explanatory variable

**X.5** Binary explanatory variable

**Details**

These data were replicated from Chib (1995)

**References**

Chib S (1995). “Marginal likelihood from the Gibbs output.” *Journal of the American Statistical Association*, **90**(432), 1313–1321.

---

stan.dist.to.r.dist     *Mapping between Stan and R distribution functions*

---

**Description**

Mapping between Stan and R distribution functions

**Usage**

```
stan.dist.to.r.dist
```

**Format**

An object of class list of length 12.

**Details**

A list of Stan distributions, associated R distribution functions, and arguments to those functions.



---

stan.operator.to.r.operator

*Mapping between Stan and R operators*


---

## Description

Mapping between Stan and R operators

## Usage

```
stan.operator.to.r.operator
```

## Format

An object of class `list` of length 3.

## Details

A list of Stan operators (regex) and associated R operators.

---

%stan\*%

*Function to replicate multiplication in Stan*


---

## Description

Function to replicate multiplication in Stan

## Usage

```
x %stan*% y
```

## Arguments

<code>x</code>	First term in product
<code>y</code>	Second term in product

## Details

Accepts arguments `x` and `y`. If either is a singleton, returns the value of `x*y` (in R notation). If both arguments are matrices or vectors, returns `x`

## Value

Returns an object of the same type as the base

**Examples**

```
X <- matrix(1:9, nrow = 3)
b <- c(4, 5, 6)

(3 + 2) * X %stan*% (5 * b)
#      [,1]
# [1,] 1650
# [2,] 2025
# [3,] 2400
```

# Index

## \*Topic **datasets**

lm.generated, [15](#)

prostatic.nodes, [15](#)

stan.dist.to.r.dist, [16](#)

stan.operator.to.r.operator, [17](#)

%stan\*%, [17](#)

build.model, [2](#)

create.restricted.models, [3](#)

define.model, [4](#)

evaluate.expression, [6](#)

evaluate.likelihood, [7](#)

evaluate.marginal.likelihood, [8](#)

evaluate.posterior, [9](#)

evaluate.priors, [10](#)

evaluate.statement, [11](#)

fit.model, [12](#)

gibbs.lm, [13](#)

lm.generated, [15](#)

prostatic.nodes, [15](#)

stan.dist.to.r.dist, [16](#)

stan.operator.to.r.operator, [17](#)