# Package ‘WpProj’

February 5, 2025

**Type** Package

**Title** Linear p-Wasserstein Projections

**Version** 0.2.3

**Date** 2025-02-03

**Description** Performs Wasserstein projections from the predictive distributions of any model into the space of predictive distributions of linear models. We utilize L1 penalties to also reduce the complexity of the model space. This package employs the methods as described in Dunipace, Eric and Lorenzo Trippa (2020) <doi:10.48550/arXiv.2012.09999>.

**License** GPL (== 3.0)

**Depends** R (>= 4.0)

**Imports** approxOT (>= 1.2), glmnet, oem, Rcpp, rlang, ROI, ROI.plugin.ecos, ROI.plugin.lpsolve, Matrix, rqPen, quantreg, doParallel, foreach, doRNG, dplyr, stats, magrittr, methods, slam, lifecycle

**LinkingTo** approxOT (>= 1.2), BH, Rcpp (>= 1.0.0), RcppCGAL, RcppEigen, RcppProgress, RSpectra

**Suggests** ggplot2, ggsci, ggridges, testthat (>= 2.1.0), transport, Rmosek, spelling, ECOSolveR

**RoxygenNote** 7.3.2

**URL** https://github.com/ericdunipace/WpProj

**BugReports** https://github.com/ericdunipace/WpProj/issues

**SystemRequirements** C++17

**Encoding** UTF-8

**Language** en-US

**NeedsCompilation** yes

**Author** Eric Dunipace [aut, cre] (<https://orcid.org/0000-0001-8909-213X>),
Clemens Schmid [ctb] (<https://orcid.org/0000-0003-3448-5715>, ETA progres bar is adapted from their code),
Espen Bernton [ctb] ('Hilbert Sort' adapted from their code),
Mathieu Gerber [ctb] ('Hilbert Sort' adapted from their code),

Pierre Jacob [ctb] ('Hilbert Sort' adapted from their code),
Bin Dai [ctb] (W2 projections adapted from their 'OEM' code),
Jared Huling [ctb] (<https://orcid.org/0000-0003-0670-4845>, W2 projections adapted from their 'OEM' code),
Yixuan Qiu [ctb] (W2 projections adapted from their 'OEM' code),
Dominic Schuhmacher [ctb] ('Shortsimplex 'optimal transport method adapted from their code),
Nicolas Bonneel [ctb] ('Network Simplex' algorithm adapted from their code)

**Maintainer** Eric Dunipace <edunipace@mail.harvard.edu>

**Repository** CRAN

**Date/Publication** 2025-02-05 18:20:02 UTC

# Contents



---

`binary_program_method_options`

*Options For Use With the Binary Program Method*

---

## Description

Options For Use With the Binary Program Method

## Usage

```
binary_program_method_options(
  maxit = 500L,
  infimum.maxit = 100L,
  transport.method = transport_options(),
```

```
  epsilon = 0.05,
  OTmaxit = 100L,
  model.size = NULL,
  nvars = NULL,
  tol = 1e-07,
  display.progress = FALSE,
  parallel = NULL,
  solver.options = NULL
)
```

## Arguments

| | |
|---|---|
| `maxit` | The maximum iterations for the optimizer. Default is 500. |
| `infimum.maxit` | Maximum iterations to alternate binary program and Wasserstein distance calculations |
| `transport.method` | Method for Wasserstein distance calculation. Should be one the outputs of `transport_options()` |
| `epsilon` | A value > 0 for the penalty parameter of if using the Sinkhorn method |
| `OTmaxit` | The number of iterations to run the Wasserstein distance solvers. |
| `model.size` | What is the maximum number of coefficients to have in the final model. Default is NULL. If NULL, will find models from the minimum size, 0, to the number of columns in `X`. |
| `nvars` | The number of variables to explore. Should be an integer vector of model sizes. Default is NULL which will explore all models from 1 to `model.size`. |
| `tol` | The tolerance for convergence |
| `display.progress` | Logical. Should intermediate progress be displayed? TRUE or FALSE. Default is FALSE. |
| `parallel` | A cluster backend to be used by `foreach::foreach()`. See `foreach::foreach()` for details about how to set them up. The `WpProj` functions will register the cluster with the `doParallel::registerDoParallel()` function internally. |
| `solver.options` | Options to be passed on to the solver. See details |

## Details

This function will setup the default arguments used by the binary program method. Of note, for the argument `solver.options`, If using the "lasso" solver, you should provide arguments such as "penalty", "nlambda", "lambda.min.ratio", "gamma", and "lambda" in a list. A simple way to do this is to feed the output of the `L1_method_options()` function to the argument `solver.options`. This will tell the approximate solver, which uses a lasso method that then will project the parameters back to the $\{0, 1\}$ space. For the other solvers, you can see the options in the ECOS solver package, `ECOSolveR::ecos.control()`, and the options for the mosek solver, `Rmosek::mosek()`.

## Value

A list with names corresponding to each argument above.

### See Also

WpProj()

### Examples

```
binary_program_method_options()
# is using the lasso solver for the binary program method to give an approximate solution
binary_program_method_options(solver.options = L1_method_options(nlambda = 50L))
```

---

## combine.WPR2 *A Function to Combine $W\_pR\hat{}2$ Objects*

---

### Description

**[Experimental]** Will combine $W_pR^2$ objects into a single object.

### Usage

```
combine.WPR2(...)
```

### Arguments

| | |
|---|---|
| ... | List of $W_pR^2$ objects |

### Value

A vector of $W_pR^2$ objects

### See Also

WPR2()

### Examples

```
if (rlang::is_installed("stats")) {
n <- 128
p <- 10
s <- 99
x <- matrix( stats::rnorm( p * n ), nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta + stats::rnorm(n)
post_beta <- matrix(beta, nrow=p, ncol=s) + stats::rnorm(p*s, 0, 0.1)
post_mu <- x %*% post_beta


fit1 <-  WpProj(X=x, eta=post_mu, theta = post_beta,
               power = 2.0, method = "binary program")
fit2 <-  WpProj(X=x, eta=post_mu, power = 2.0,
               options = list(penalty = "lasso")
```

```
)


out1 <- WPR2(predictions = post_mu, projected_model = fit1)
out2 <- WPR2(predictions = post_mu, projected_model = fit2)

combine <- combine.WPR2(out1, out2)
}
```

---

`distCompare` *Compares Optimal Transport Distances Between WpProj and Original Models*

---

## Description

**[Experimental]** Will compare the Wasserstein distance between the original model and the `WpProj` model.

## Usage

```
distCompare(
  models,
  target = list(parameters = NULL, predictions = NULL),
  power = 2,
  method = "exact",
  quantity = c("parameters", "predictions"),
  parallel = NULL,
  transform = function(x) {
     return(x)
 },
  ...
)
```

## Arguments

| | |
|---|---|
| `models` | A list of models from WpProj methods |
| `target` | The target to compare the methods to. Should be a list with slots "parameters" to compare the parameters and "predictions" to compare predictions |
| `power` | The power parameter of the Wasserstein distance. |
| `method` | Which approximation to the Wasserstein distance to use. Should be one of the outputs of `transport_options()`. |
| `quantity` | Should the function target the "parameters" or the "predictions". Can choose both. |
| `parallel` | Parallel backend to use for the `foreach` package. See `foreach::registerDoParallel(` for more details. |
| `transform` | Transformation function for the predictions. |
| `...` | other options passed to the `wasserstein()` distance function |

## Details

For the data frames, `dist` is the Wasserstein distance, `nactive` is the number of active variables in the model, `groups` is the name distinguishing the model, and `method` is the method used to calculate the distance (i.e., exact, sinkhorn, etc.). If the list in `models` is named, these will be used as the group names otherwise the group names will be created based on the call from the `WpProj` method.

## Value

an object of class `distcompare` with slots `parameters`, `predictions`, and `p`. The slots `parameters` and `predictions` are data frames. See the details for more info. The slot `p` is the power parameter of the Wasserstein distance used in the distance calculation.

## Examples

```
if(rlang::is_installed("stats")) {
n <- 32
p <- 10
s <- 21
x <- matrix( stats::rnorm( p * n ), nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta + stats::rnorm(n)
post_beta <- matrix(beta, nrow=p, ncol=s) + stats::rnorm(p*s, 0, 0.1)
post_mu <- x %*% post_beta

fit1 <-  WpProj(X=x, eta=post_mu, power = 2.0,
               options = list(penalty = "lasso")
)
fit2 <-  WpProj(X=x, eta=post_mu, theta = post_beta, power = 2.0,
               method = "binary program", solver = "lasso",
               options = list(solver.options = list(penalty = "mcp"))
)
dc <- distCompare(models = list("L1" = fit1, "BP" = fit2),
                 target = list(parameters = post_beta, predictions = post_mu))
if(rlang::is_installed(c("ggplot2","ggsci"))) {
plot(dc)
}
}
```

---

HC *Run the Hahn-Carvalho Method*

---

## Description

**[Experimental]** Runs the Hahn-Carvalho method but adapted to return full distributions.

## Usage

```
HC(
  X,
  Y = NULL,
  theta,
  family = "gaussian",
 penalty = c("elastic.net", "selection.lasso", "lasso", "ols", "mcp", "scad", "mcp.net",
   "scad.net", "grp.lasso", "grp.lasso.net", "grp.mcp", "grp.scad", "grp.mcp.net",
    "grp.scad.net", "sparse.grp.lasso"),
  method = c("selection.variable", "projection"),
  lambda = numeric(0),
  nlambda = 100L,
  lambda.min.ratio = NULL,
  alpha = 1,
  gamma = 1,
  tau = 0.5,
  groups = numeric(0),
  penalty.factor = NULL,
  group.weights = NULL,
  maxit = 500L,
  tol = 1e-07,
  irls.maxit = 100L,
  irls.tol = 0.001
)
```

## Arguments

| | |
|---|---|
| `X` | Covariates |
| `Y` | Predictions |
| `theta` | Parameters |
| `family` | Family for method. See oem. |
| `penalty` | Penalty function. See oem. |
| `method` | Should we run a selection variable methodology or projection? |
| `lambda` | lambda for lasso. See oem for this and all options below |
| `nlambda` | Number of lambda values. |
| `lambda.min.ratio` | Minimum lambda ratio for self selected lambda |
| `alpha` | elastic net mixing. |
| `gamma` | tuning parameters for SCAD and MCP |
| `tau` | mixing parameter for sparse group lasso |
| `groups` | A vector of grouping values |
| `penalty.factor` | Penalty factor for OEM. |
| `group.weights` | Weights for groupped lasso |
| `maxit` | Max iteration for OEM |

| | |
|---|---|
| tol | Tolerance for OEM |
| irls.maxit | IRLS max iterations for OEM |
| irls.tol | IRLS tolerance for OEM |

## Value

a WpProj object with selected covariates and their values

## References

Hahn, P. Richard and Carlos M. Carvalho. (2014) "Decoupling Shrinkage and Selection in Bayesian Linear Models: A Posterior Summary Perspective." https://arxiv.org/pdf/1408.0464

## Examples

```
n <- 32
p <- 10
s <- 99
x <- matrix( 1, nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta
post_beta <- matrix(beta, nrow=p, ncol=s)
post_mu <- x %*% post_beta

fit <-  HC(X=x, Y=post_mu, theta = post_beta,
               penalty = "lasso",
               method = "projection"
)
```

---

L0_method_options    *Options For Use With the L0 Method*

---

## Description

Options For Use With the L0 Method

## Usage

```
L0_method_options(
  method = c("binary program", "projection"),
  transport.method = transport_options(),
  epsilon = 0.05,
  OTmaxit = 0,
  parallel = NULL,
  ...
)
```

### Arguments

| | |
|---|---|
| `method` | Should covariates be selected as an approximate "binary program" or should a projection method be used. Default is the approximate binary program. |
| `transport.method` | Method for Wasserstein distance calculation. Should be one the outputs of `transport_options()`. |
| `epsilon` | A value > 0 for the penalty parameter if using the Sinkhorn method for optimal transport |
| `OTmaxit` | The number of iterations to run the Wasserstein distance solvers. |
| `parallel` | A cluster backend to be used by `foreach::foreach()` if parallelization is desired. |
| `...` | Not used |

### Value

a named list corresponding to the above arguments

### Examples

```
L0_method_options()
```

---

## `L1_method_options` *Options For Use With the L1 Method*

---

### Description

Options For Use With the L1 Method

### Usage

```
L1_method_options(
  penalty = L1_penalty_options(),
  lambda = numeric(0),
  nlambda = 500L,
  lambda.min.ratio = 1e-04,
  gamma = 1,
  alpha = 1,
  maxit = 500L,
  model.size = NULL,
  tol = 1e-07,
  display.progress = FALSE,
  solver.options = NULL
)
```

## Arguments

- `penalty` — The penalty to use. See `L1_penalty_options()` for more details.
- `lambda` — The penalty parameter to use if method is "L1".
- `nlambda` — The number of lambdas to explore for the "L1" method if `lambda` is not provided
- `lambda.min.ratio` — The minimum ratio of max to min lambda for "L1" method. Default 1e-4.
- `gamma` — Tuning parameter for SCAD and MCP penalties if method = "L1". >=1
- `alpha` — Tuning parameter for elastic net penalties `alpha` should be in `[0,1]`.
- `maxit` — The maximum iterations for optimization. Default is 500.
- `model.size` — What is the maximum number of coefficients to have in the final model. Default is NULL. If NULL, will find models from the minimum size, 0, to the number of columns in `X`.
- `tol` — The tolerance for convergence
- `display.progress` — Logical. Should intermediate progress be displayed? TRUE or FALSE. Default is FALSE.
- `solver.options` — Options to be passed on to the solver. Only used for "ecos" and "mosek" solvers.

## Value

A list with names corresponding to each argument above.

## See Also

`WpProj()`

## Examples

```
L1_method_options()
```

---

# L1_penalty_options *Recognized L1 Penalties*

---

## Description

Recognized L1 Penalties

## Usage

```
L1_penalty_options()
```

## Value

A character vector with the possible penalties for L1 methods

## Examples

```
L1_penalty_options()
# [1] "lasso"            "ols"              "mcp"              "elastic.net"      "scad"
# [6] "mcp.net"          "scad.net"         "grp.lasso"        "grp.lasso.net"    "grp.mcp"
# [11] "grp.scad"         "grp.mcp.net"      "grp.scad.net"     "sparse.grp.lasso"
```

---

plot,WPR2-method *Plot Function for W_pR^2 Objects*

---

## Description

Plot Function for $W_pR^2$ Objects

## Usage

```
## S4 method for signature 'WPR2'
plot(
  x,
  xlim = NULL,
  ylim = NULL,
  linesize = 0.5,
  pointsize = 1.5,
  facet.group = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| x | A $W_pR^2$ object |
| xlim | x-axis limits |
| ylim | y-axis limits |
| linesize | Linesize for geom_line |
| pointsize | Point size for geom_point |
| facet.group | Group to do facet_grid by |
| ... | Currently not used |

## Value

a `ggplot2::ggplot()` object

### Examples

```
n <- 128
p <- 10
s <- 99
x <- matrix( stats::rnorm( p * n ), nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta + stats::rnorm(n)
post_beta <- matrix(beta, nrow=p, ncol=s) + stats::rnorm(p*s, 0, 0.1)
post_mu <- x %*% post_beta

fit <-  WpProj(X=x, eta=post_mu, power = 2.0,
               options = list(penalty = "lasso")
)
obj <- WPR2(predictions = post_mu, projected_model = fit)
if (rlang::is_installed("ggplot2")) {
p <- plot(obj)
}
```

---

`ridgePlot` *Ridge Plots for a Range of Coefficients*

---

### Description

**[Experimental]** This function will plot the distribution of predictions for a range of active coefficients

### Usage

```
ridgePlot(
  fit,
  index = 1,
  minCoef = 1,
  maxCoef = 10,
  scale = 1,
  alpha = 0.5,
  full = NULL,
  transform = function(x) {
     x
 },
  xlab = "Predictions",
  bandwidth = NULL
)
```

### Arguments

| | |
|---|---|
| `fit` | A `WpProj` object or list of `WpProj` objects |
| `index` | The observation number to select. Can be a vector |

| | |
|---|---|
| minCoef | The minimum number of coefficients to use |
| maxCoef | The maximum number of coefficients to use |
| scale | How the densities should be scale |
| alpha | Alpha term from ggplot2 object |
| full | "True" prediction to compare to |
| transform | transform for predictions |
| xlab | x-axis label |
| bandwidth | Bandwidth for kernel |

### Value

a ggplot2::ggplot() plot

### Examples

```
if(rlang::is_installed("stats")) {
n <- 128
p <- 10
s <- 99
x <- matrix(stats::rnorm(n*p), nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta + stats::rnorm(n)
post_beta <- matrix(beta, nrow=p, ncol=s) + matrix(stats::rnorm(p*s, 0, 0.1), p, s)
post_mu <- x %*% post_beta
fit <-  WpProj(X=x, eta=post_mu,
             power = 2
)
if(rlang::is_installed(c("ggplot2","ggsci","ggridges"))) {
ridgePlot(fit)
}
}
```

---

## simulated_annealing_method_options

*Options For Use With the Simulated Annealing Selection Method*

---

### Description

Options For Use With the Simulated Annealing Selection Method

### Usage

```
simulated_annealing_method_options(
  force = NULL,
  method = c("binary program", "projection"),
  transport.method = transport_options(),
```

```
  OTmaxit = 0L,
  epsilon = 0.05,
  maxit = 1L,
  temps = 1000L,
  max.time = 3600,
  proposal.method = c("covariance", "uniform"),
  energy.distribution = c("boltzman", "bose-einstein"),
  cooling.schedule = c("Geman-Geman", "exponential"),
  model.size = NULL,
  nvars = NULL,
  display.progress = FALSE,
  parallel = NULL,
  calc.theta = TRUE,
  ...
)
```

## Arguments

`force` Any covariates to force into the model? Should be by column number or NULL if no variables to force into the model.

`method` Should covariates be selected as an approximate "binary program" or should a projection method be used. Default is the approximate binary program.

`transport.method` Method for Wasserstein distance calculation. Should be one the outputs of `transport_options()`

`OTmaxit` The number of iterations to run the Wasserstein distance solvers.

`epsilon` A value > 0 for the penalty parameter of if using the Sinkhorn method for optimal transport

`maxit` Maximum number of iterations per temperature

`temps` Number of temperatures to try

`max.time` Maximum time in seconds to run the algorithm

`proposal.method` The method to propose the next covariate to add. One of "covariance" or "random". "covariance" will randomly select from covariates with probability proportional to the absolute value of the covariance. "uniform" will select covariates uniformly at random.

`energy.distribution` The energy distribution to use for evaluating proposals. One of "boltzman" or "bose-einstein". Default is "boltzman".

`cooling.schedule` The schedule to use for cooling temperatures. One of "Geman-Geman" or "exponential". Default is "Geman-Geman".

`model.size` How many coefficients should the maximum final model have? Ignored if `nvars` set.

`nvars` What model sizes should one check? Should be a numeric vector with maximum less than number of variables or `NULL`. Default is NULL. Overrides `model.size` if is not `NULL`

display.progress
: Logical. Should intermediate progress be displayed? TRUE or FALSE. Default is `FALSE`.

parallel
: A cluster backend to be used by `foreach::foreach()`. See `foreach::foreach()` for details about how to set them up. The `WpProj` functions will register the cluster with the `doParallel::registerDoParallel()` function internally.

calc.theta
: Return the linear coefficients? Default is TRUE.

...
: Not used.

### Value

A named list with the above arguments

### Examples

```
simulated_annealing_method_options()
```

---

## stepwise_method_options *Options For Use With the Stepwise Selection Method*

---

### Description

Options For Use With the Stepwise Selection Method

### Usage

```
stepwise_method_options(
  force = NULL,
  direction = c("backward", "forward"),
  method = c("binary program", "projection"),
  transport.method = transport_options(),
  OTmaxit = 0,
  epsilon = 0.05,
  model.size = NULL,
  display.progress = FALSE,
  parallel = NULL,
  calc.theta = TRUE,
  ...
)
```

### Arguments

force
: Any covariates to force into the model? Should be by column number or NULL if no variables to force into the model.

direction
: "forward" or "backward" selection? Default is "backward"

`method` Should covariates be selected as an approximate "binary program" or should a projection method be used. Default is the approximate binary program.

`transport.method` Method for Wasserstein distance calculation. Should be one the outputs of `transport_options()`

`OTmaxit` The number of iterations to run the Wasserstein distance solvers.

`epsilon` A value > 0 for the penalty parameter of if using the Sinkhorn method for optimal transport

`model.size` How many coefficients should the maximum final model have?

`display.progress` Logical. Should intermediate progress be displayed? TRUE or FALSE. Default is FALSE.

`parallel` A cluster backend to be used by `foreach::foreach()`. See `foreach::foreach()` for details about how to set them up. The `WpProj` functions will register the cluster with the `doParallel::registerDoParallel()` function internally.

`calc.theta` Return the linear coefficients? Default is TRUE.

`...` Not used

## Value

A named list with the above arguments

## Examples

```
stepwise_method_options()
```

---

`WpProj` *p-Wasserstein Linear Projections*

---

## Description

**[Experimental]** This function will calculate linear projections from a set of predictions into the space of the covariates in terms of the p-Wasserstein distance.

## Usage

```
WpProj(
  X,
  eta = NULL,
  theta = NULL,
  power = 2,
  method = c("L1", "binary program", "stepwise", "simulated annealing", "L0"),
  solver = c("lasso", "ecos", "lpsolve", "mosek"),
  options = NULL
)
```

## Arguments

| | |
|---|---|
| `X` | An $n \times p$ matrix of covariates |
| `eta` | An $n \times s$ matrix of predictions from a model |
| `theta` | An optional An $p \times s$ parameter matrix for selection methods. Only makes sense if the original model is a linear model. |
| `power` | The power of the Wasserstein distance to use. Must be `>= 1.0`. Will default to `2.0`. |
| `method` | The algorithm to calculate the Wasserstein projections. One of "L1", "binary program", "IP", "stepwise","simulated annealing", or "L0". Will default to "L1" if not provided. See details for more information. |
| `solver` | Which solver to use? One of "lasso", "ecos", "lpsolve", or "mosek". See details for more information |
| `options` | Options passed to the particular method and desired solver. See details for more information. |

## Details

**Methods:**

The `WpProj` function is a wrapper for the various Wasserstein projection methods. It is designed to be a one-stop shop for all Wasserstein projection methods. It will automatically choose the correct method and solver based on the arguments provided. It will also return a standardized output for all methods. Each method has its own set of options that can be passed to it. See the documentation for each method for more information.

For the L1 methods, see `L1_method_options()` for more information. For the binary program methods, see `binary_program_method_options()` for more information. For the stepwise methods, see `stepwise_method_options()` for more information. For the simulated annealing methods, see `simulated_annealing_method_options()` for more information.

In most cases, we recommend using the L1 methods or binary program methods. The L1 methods are the fastest and applicable to Wasserstein powers of any value greater than 1 and function as direct linear projections into the space of the covariates. The binary program methods instead preserve the coefficients of the original model if this is of interest, such as when the original model was already a linear model. The binary program will instead function as a way of turning on and off certain coefficients in a way that minimizes the Wasserstein distance between reduced and original models. Of note, we also have available an approximate binary program method using a lasso solver. This method is faster than the exact binary program method but is not guaranteed to find the optimal solution. It is recommended to use the exact binary program method if possible. See `binary_program_method_options()` for more information on how to set up the approximate method as some arguments for the lasso solver should be specified. For more information on how this works, please also see the referenced paper.

The stepwise, simulated annealing, and L0 methods also select covariates like the binary program methods but they can be slower. They are presented merely for comparison purposes given they were used in the original paper.

**Wasserstein distances and powers:**

The Wasserstein distance is a measure of distance between two probability distributions. It is defined as:

$$W_p(\mu, \nu) = \left( \inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathbb{R}^d \times \mathbb{R}^d} \|x - y\|^p d\pi(x, y) \right)^{1/p},$$

where $\Pi(\mu, \nu)$ is the set of all joint distributions with marginals $\mu$ and $\nu$. The Wasserstein distance is a generalization of the Euclidean distance, which is the case when $p = 2$. In our function we have argument `power` that corresponds to the $p$ of the equation above. The default `power` is `2.0` but any value greater than or equal to `1.0` is allowed. For more information, see the references.

The particular implementation of the Wasserstein distance is as follows. If $\mu$ is the original prediction from the original model, then we seek to find a new prediction $\nu$ that minimizes the Wasserstein distance between the two: $\text{argmin}_\nu W_p(\mu, \nu)$.

## Value

object of class `WpProj`, which is a list with the following slots:

- `call`: The call to the function
- `theta`: A list of the final parameter matrices for each returned model
- `fitted.values`: A list of the fitted values for each returned model
- `power`: The power of the Wasserstein distance used
- `method`: The method used to calculate the Wasserstein projections
- `solver`: The solver used to calculate the Wasserstein projections
- `niter`: The number of iterations used to calculate the Wasserstein projections. Not all methods return a number of iterations so this may be `NULL`
- `nzero`: The number of non zero coefficients in the final models

## References

Dunipace, Eric and Lorenzo Trippa (2020) https://arxiv.org/abs/2012.09999.

## Examples

```
if(rlang::is_installed("stats")) {
# note we don't generate believable data with real posteriors
# these examples are just to show how to use the function
n <- 32
p <- 10
s <- 21

# covariates and coefficients
x <- matrix( stats::rnorm( p * n ), nrow = n, ncol = p )
beta <- (1:10)/10

#outcome
y <- x %*% beta + stats::rnorm(n)

# fake posterior
post_beta <- matrix(beta, nrow=p, ncol=s) + stats::rnorm(p*s, 0, 0.1)
```

```
post_mu <- x %*% post_beta #posterior predictive distributions

# fit models
## L1 model
fit.p2     <-  WpProj(X=x, eta=post_mu, power = 2.0,
                   method = "L1", #default
                   solver = "lasso" #default
)

## approximate binary program
fit.p2.bp <-  WpProj(X=x, eta=post_mu, theta = post_beta, power = 2.0,
                   method = "binary program",
                   solver = "lasso" #default because approximate algorithm is faster
)

## compare performance by measuring distance from full model
dc <- distCompare(models = list("L1" = fit.p2, "BP" = fit.p2.bp))
if(rlang::is_installed(c("ggplot2","ggsci"))) {
plot(dc)
}

## compare performance by measuring the relative distance between a null model
## and the predictions of interest as a pseudo R^2
r2.expect <- WPR2(predictions = post_mu, projected_model = dc) # can have negative values
r2.null  <- WPR2(projected_model = dc) # should be between 0 and 1
if(rlang::is_installed(c("ggplot2","ggsci"))) {
plot(r2.null)
}

## we can also examine how predictions change in the models for individual observations
if(rlang::is_installed(c("ggplot2","ggsci","ggridges"))) {
ridgePlot(fit.p2, index = 21, minCoef = 0, maxCoef = 10)
}
}
```

---

WPR2 *$W\_pR\hat{}2$ Function to Evaluate Performance*

---

## Description

**[Experimental]** This function will calculate p-Wasserstein distances between the predictions of interest and the projected model.

## Usage

```
WPR2(
  predictions = NULL,
  projected_model,
  p = 2,
  method = "exact",
```

```
  base = NULL,
  ...
)

## S4 method for signature 'ANY,matrix'
WPR2(
  predictions = NULL,
  projected_model,
  p = 2,
  method = "exact",
  base = NULL,
  ...
)

## S4 method for signature 'ANY,distcompare'
WPR2(
  predictions = NULL,
  projected_model,
  p = 2,
  method = "exact",
  base = NULL,
  ...
)

## S4 method for signature 'ANY,list'
WPR2(
  predictions = NULL,
  projected_model,
  p = 2,
  method = "exact",
  base = NULL,
  ...
)

## S4 method for signature 'ANY,WpProj'
WPR2(
  predictions = NULL,
  projected_model,
  p = 2,
  method = "exact",
  base = NULL,
  ...
)
```

## Arguments

`predictions` Predictions of interest, likely from the original model

`projected_model` A matrix of competing predictions, possibly from a WpProj fit, a WpProj fit

| | |
|---|---|
| | itself, or a list of WpProj objects |
| `p` | Power of the Wasserstein distance to use in distance calculations |
| `method` | Method for calculating Wasserstein distance |
| `base` | The baseline result to compare to. If not provided, defaults to the model with no covariates and only an intercept. |
| `...` | Arguments passed to Wasserstein distance calculation. See `wasserstein` |

## Value

$W_p R^2$ values

## Examples

```
if (rlang::is_installed("stats")) {
# this example is not a true posterior estimation, but is used for illustration
n <- 32
p <- 10
s <- 21
x <- matrix( stats::rnorm(n*p), nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta + stats::rnorm(n)
post_beta <- matrix(beta, nrow=p, ncol=s) +
    matrix(rnorm(p*s), p, s) # not a true posterior
post_mu <- x %*% post_beta

fit <-  WpProj(X=x, eta=post_mu, power = 2.0)

out <- WPR2(predictions = post_mu, projected_model = fit,
base = rowMeans(post_mu) # same as intercept only projection
)
}
```

---

`WPVI` *p-Wasserstein Variable Importance*

---

## Description

**[Experimental]** This function will measure how much removing each covariate harms prediction accuracy.

## Usage

```
WPVI(
  X,
  eta,
  theta,
  pred.fun = NULL,
  p = 2,
```

```
  ground_p = 2,
  transport.method = transport_options(),
  epsilon = 0.05,
  OTmaxit = 0,
  display.progress = FALSE,
  parallel = NULL
)
```

### Arguments

| | |
|---|---|
| X | Covariates |
| eta | Predictions from the estimated model |
| theta | Parameters from the estimated model. |
| pred.fun | A prediction function. must take variables x, theta as arguments: pred.fun(x,theta) |
| p | Power of Wasserstein distance |
| ground_p | Power of distance metric |
| transport.method | Transport methods. See transport_options() for more details. |
| epsilon | Hyperparameter for Sinkhorn iterations |
| OTmaxit | Maximum number of iterations for the Wasserstein method |
| display.progress | Display intermediate progress |
| parallel | a foreach backend if already created |

### Value

Returns an integer vector ranking covariate importance from most to least important.

### Examples

```
n <- 128
p <- 10
s <- 99
x <- matrix(1, nrow = n, ncol = p )
beta <- (1:10)/10
y <- x %*% beta
post_beta <- matrix(beta, nrow=p, ncol=s)
post_mu <- x %*% post_beta

fit <-  WpProj(X=x, eta=post_mu, power = 2.0)
WPVI(X = x, eta = post_mu, theta = post_beta, transport.method = "hilbert")
```

# Index