Package: DPpack (via r-universe)

September 17, 2024

Type Package

Title Differentially Private Statistical Analysis and Machine Learning

Version 0.2.0

Author Spencer Giddens <giddens2spencer@gmail.com> with contributions from Fang Liu <fliu2@nd.edu>

Maintainer Spencer Giddens <giddens2spencer@gmail.com>

Description An implementation of common statistical analysis and models with differential privacy (Dwork et al., 2006a) <doi:10.1007/11681878_14> guarantees. The package contains, for example, functions providing differentially private computations of mean, variance, median, histograms, and contingency tables. It also implements some statistical models and machine learning algorithms such as linear regression (Kifer et al., 2012) <https://proceedings.mlr.press/v23/kifer12.html> and SVM (Chaudhuri et al., 2011) <https://jmlr.org/papers/v12/chaudhuri11a.html>. In addition, it implements some popular randomization mechanisms, including the Laplace mechanism (Dwork et al., 2006a) <doi:10.1007/11681878_14>, Gaussian mechanism (Dwork et al., 2006b) <doi:10.1007/11761679_29>, analytic Gaussian mechanism (Balle & Wang, 2018) <https://proceedings.mlr.press/v80/balle18a.html>, and exponential mechanism (McSherry & Talwar, 2007) <doi:10.1109/FOCS.2007.66>.

License GPL-3 | file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Imports rmutil (>= 1.1.5), Rdpack (>= 2.1.2), R6 (>= 2.5.1), dplyr (>= 1.0.1), MASS (>= 7.3-51.6), nloptr (>= 1.2.2.2), e1071 (>= 1.7-9), stats (>= 4.0.2), graphics (>= 4.0.2), ggplot2 (>= 3.3.2)

RdMacros Rdpack

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

Repository https://sgiddens.r-universe.dev

RemoteUrl https://github.com/sgiddens/dppack

RemoteRef HEAD

RemoteSha 7401779c9f10f538d301d924f7bed1223cefb6f6

Contents

`AnalyticGaussianMechanism`

Analytic Gaussian Mechanism

Description

This function implements the analytic Gaussian mechanism for differential privacy by adding noise to the true value(s) of a function according to specified values of epsilon, delta, and l2-global sensitivity(-ies). The noise scale is for the analytic Gaussian mechanism (Balle and Wang 2018). Global sensitivity calculated based either on bounded or unbounded differential privacy can be used (Kifer and Machanavajjhala 2011). If true.values is a vector, the provided epsilon and delta are divided such that (epsilon, delta)-level differential privacy is satisfied across all function values. In the case that each element of true.values comes from its own function with different corresponding

sensitivities, a vector of sensitivities may be provided. In this case, if desired, the user can specify how to divide epsilon and delta among the function values using alloc.proportions.

Usage

```
AnalyticGaussianMechanism(
  true.values,
  eps,
  delta,
  sensitivities,
  alloc.proportions = NULL,
  tol = 1e-12
)
```

Arguments

`true.values`	Real number or numeric vector corresponding to the true value(s) of the desired function.
`eps`	Positive real number defining the epsilon privacy parameter.
`delta`	Positive real number defining the delta privacy parameter.
`sensitivities`	Real number or numeric vector corresponding to the l2-global sensitivity(-ies) of the function(s) generating true.values. This value must be of length 1 or of the same length as true.values. If it is of length 1 and true.values is a vector, this indicates that the given sensitivity applies simultaneously to all elements of true.values and that the privacy budget need not be allocated (alloc.proportions is unused in this case). If it is of the same length as true.values, this indicates that each element of true.values comes from its own function with different corresponding sensitivities. In this case, the l2-norm of the provided sensitivities is used to generate the Gaussian noise.
`alloc.proportions`	Optional numeric vector giving the allocation proportions of epsilon and delta to the function values in the case of vector-valued sensitivities. For example, if sensitivities is of length two and alloc.proportions = c(.75, .25), then 75% of the privacy budget eps (and 75% of delta) is allocated to the noise computation for the first element of true.values, and the remaining 25% is allocated to the noise computation for the second element of true.values. This ensures (eps, delta)-level privacy across all computations. Input does not need to be normalized, meaning alloc.proportions = c(3,1) produces the same result as the example above.
`tol`	Optional error tolerance for binary search used in determining the noise parameter for the analytic Gaussian mechanism. Defaults to 1e-12.

Value

Sanitized function values based on the bounded and/or unbounded definitions of differential privacy, sanitized via the analytic Gaussian mechanism.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Balle B, Wang Y (2018). "Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising." In Dy J, Krause A (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, 394–403. https://proceedings.mlr.press/v80/balle18a.html.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Liu F (2019). "Generalized Gaussian Mechanism for Differential Privacy." *IEEE Transactions on Knowledge and Data Engineering*, **31**(4), 747-756. https://doi.org/10.1109/TKDE.2018.2845388.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Examples

```
# Simulate dataset
n <- 100
c0 <- 5 # Lower bound
c1 <- 10 # Upper bound
D1 <- stats::runif(n, c0, c1)

# Privacy budget
epsilon <- 1.1 # epsilon can be > 1 for analytic Gaussian mechanism
delta <- 0.01
sensitivity <- (c1-c0)/n

private.mean <- AnalyticGaussianMechanism(mean(D1), epsilon,
                                          delta, sensitivity)
private.mean


# Simulate second dataset
d0 <- 3 # Lower bound
d1 <- 6 # Upper bound
D2 <- stats::runif(n, d0, d1)
D <- matrix(c(D1,D2),ncol=2)
sensitivities <- c((c1-c0)/n, (d1-d0)/n)
epsilon <- 0.9 # Total privacy budget for all means
delta <- 0.01

# Here, sensitivities are summed and the result is used to generate Laplace
# noise. This is essentially the same as allocating epsilon proportional to
# the corresponding sensitivity. The results satisfy (0.9,0.01)-approximate
# differential privacy.
```

```
private.means <- AnalyticGaussianMechanism(apply(D, 2, mean), epsilon,
                                           delta, sensitivities)
private.means

# Here, privacy budget is explicitly split so that 75% is given to the first
# vector element and 25% is given to the second.
private.means <- AnalyticGaussianMechanism(apply(D, 2, mean), epsilon,
                                           delta, sensitivities,
                                           alloc.proportions = c(0.75, 0.25))
private.means
```

`covDP` *Differentially Private Covariance*

Description

This function computes the differentially private covariance of a pair of vectors at user-specified privacy levels of epsilon and delta.

Usage

```
covDP(
  x1,
  x2,
  eps,
  lower.bound1,
  upper.bound1,
  lower.bound2,
  upper.bound2,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP"
)
```

Arguments

`x1`, `x2`	Numeric vectors whose covariance is desired.
`eps`	Positive real number defining the epsilon privacy budget.
`lower.bound1`, `lower.bound2`	Real numbers giving the global or public lower bounds of x1 and x2, respectively.
`upper.bound1`, `upper.bound2`	Real numbers giving the global or public upper bounds of x1 and x2, respectively.

`which.sensitivity`
: String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.

`mechanism`
: String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.

`delta`
: Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP`
: String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

Value

Sanitized covariance based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Liu F (2019). "Statistical Properties of Sanitized Results from Differentially Private Laplace Mechanism with Univariate Bounding Constraints." *Transactions on Data Privacy*, **12**(3), 169-195. http://www.tdp.cat/issues16/tdp.a316a18.pdf.

Examples

```
D1 <- sort(stats::rnorm(500, mean=3, sd=2))
D2 <- sort(stats::rnorm(500, mean=-1,sd=0.5))
lb1 <- -3 # 3 std devs below mean
lb2 <- -2.5 # 3 std devs below mean
ub1 <- 9 # 3 std devs above mean
```

```
ub2 <- .5 # 3 std devs above mean
covDP(D1, D2, 1, lb1, ub1, lb2, ub2)
covDP(D1, D2, .5, lb1, ub1, lb2, ub2, which.sensitivity='unbounded',
  mechanism='Gaussian', delta=0.01)
```

`ExponentialMechanism` *Exponential Mechanism*

Description

This function implements the exponential mechanism for differential privacy by selecting the index of a vector of candidates to return according to a user-specified vector of utility function values, epsilon, and global sensitivity. Sensitivity calculated based either on bounded or unbounded differential privacy can be used (Kifer and Machanavajjhala 2011). If measure is provided, the probabilities of selecting each value are scaled according to the values in measure. If candidates is provided, the function returns the value of candidates at the selected index, rather than the index itself.

Usage

```
ExponentialMechanism(
  utility,
  eps,
  sensitivity,
  measure = NULL,
  candidates = NULL
)
```

Arguments

`utility`	Numeric vector giving the utilities of the possible values.
`eps`	Positive real number defining the epsilon privacy budget.
`sensitivity`	Real number corresponding to the l1-global sensitivity of the function generating utility.
`measure`	Optional numeric vector of scaling measures for the probabilities of selecting each value. Should be same size as utility. Defaults to uniform scaling.
`candidates`	Optional vector of candidates of same size as utility. If given, the function returns the candidate at the selected index rather than the index itself.

Value

Indices (or values if candidates given) selected by the mechanism based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

McSherry F, Talwar K (2007). "Mechanism Design via Differential Privacy." In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, 94-103. doi:10.1109/FOCS.2007.66.

Examples

```
candidates <- c('a','b','c','d','e','f','g')
# Release index
idx <- ExponentialMechanism(c(0,1,2,3,2,1,0), 1, 1)
candidates[idx] # Randomly chosen candidate

# Release candidate
ExponentialMechanism(c(0,1,2,3,2,1,0), 1, .5, measure=c(1,1,2,1,2,1,1),
  candidates=candidates)
```

`GaussianMechanism` *Gaussian Mechanism*

Description

This function implements the Gaussian mechanism for differential privacy by adding noise to the true value(s) of a function according to specified values of epsilon, delta, and l2-global sensitivity(-ies). Global sensitivity calculated based either on bounded or unbounded differential privacy can be used (Kifer and Machanavajjhala 2011). If true.values is a vector, the provided epsilon and delta are divided such that (epsilon, delta)-level differential privacy is satisfied across all function values. In the case that each element of true.values comes from its own function with different corresponding sensitivities, a vector of sensitivities may be provided. In this case, if desired, the user can specify how to divide epsilon and delta among the function values using alloc.proportions.

Usage

```
GaussianMechanism(
  true.values,
  eps,
  delta,
  sensitivities,
  type.DP = "aDP",
  alloc.proportions = NULL
)
```

Arguments

`true.values`	Real number or numeric vector corresponding to the true value(s) of the desired function.
`eps`	Positive real number defining the epsilon privacy parameter.
`delta`	Positive real number defining the delta privacy parameter.
`sensitivities`	Real number or numeric vector corresponding to the l2-global sensitivity(-ies) of the function(s) generating true.values. This value must be of length 1 or of the same length as true.values. If it is of length 1 and true.values is a vector, this indicates that the given sensitivity applies simultaneously to all elements of true.values and that the privacy budget need not be allocated (alloc.proportions is unused in this case). If it is of the same length as true.values, this indicates that each element of true.values comes from its own function with different corresponding sensitivities. In this case, the l2-norm of the provided sensitivities is used to generate the Gaussian noise.
`type.DP`	String indicating the type of differential privacy desired for the Gaussian mechanism. Can be either 'pDP' for probabilistic DP (Liu 2019) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.
`alloc.proportions`	Optional numeric vector giving the allocation proportions of epsilon and delta to the function values in the case of vector-valued sensitivities. For example, if sensitivities is of length two and alloc.proportions = c(.75, .25), then 75% of the privacy budget eps (and 75% of delta) is allocated to the noise computation for the first element of true.values, and the remaining 25% is allocated to the noise computation for the second element of true.values. This ensures (eps, delta)-level privacy across all computations. Input does not need to be normalized, meaning alloc.proportions = c(3,1) produces the same result as the example above.

Value

Sanitized function values based on the bounded and/or unbounded definitions of differential privacy, sanitized via the Gaussian mechanism.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Liu F (2019). "Generalized Gaussian Mechanism for Differential Privacy." *IEEE Transactions on Knowledge and Data Engineering*, **31**(4), 747-756. https://doi.org/10.1109/TKDE.2018.2845388.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Examples

```
# Simulate dataset
n <- 100
c0 <- 5 # Lower bound
c1 <- 10 # Upper bound
D1 <- stats::runif(n, c0, c1)

# Privacy budget
epsilon <- 0.9 # eps must be in (0, 1) for approximate differential privacy
delta <- 0.01
sensitivity <- (c1-c0)/n

# Approximate differential privacy
private.mean.approx <- GaussianMechanism(mean(D1), epsilon, delta,
                                         sensitivity)
private.mean.approx

# Probabilistic differential privacy
private.mean.prob <- GaussianMechanism(mean(D1), epsilon, delta, sensitivity,
                                       type.DP = 'pDP')
private.mean.prob

# Simulate second dataset
d0 <- 3 # Lower bound
d1 <- 6 # Upper bound
D2 <- stats::runif(n, d0, d1)
D <- matrix(c(D1,D2),ncol=2)
sensitivities <- c((c1-c0)/n, (d1-d0)/n)
epsilon <- 0.9 # Total privacy budget for all means
delta <- 0.01

# Here, sensitivities are summed and the result is used to generate Laplace
# noise. This is essentially the same as allocating epsilon proportional to
# the corresponding sensitivity. The results satisfy (0.9,0.01)-approximate
# differential privacy.
private.means <- GaussianMechanism(apply(D, 2, mean), epsilon, delta,
                                   sensitivities)
private.means

# Here, privacy budget is explicitly split so that 75% is given to the first
# vector element and 25% is given to the second.
private.means <- GaussianMechanism(apply(D, 2, mean), epsilon, delta,
                                   sensitivities,
                                   alloc.proportions = c(0.75, 0.25))
private.means
```

`histogramDP` *Differentially Private Histogram*

Description

This function computes a differentially private histogram from a vector at user-specified privacy levels of epsilon and delta. A histogram object is returned with sanitized values for the counts for easy plotting.

Usage

```
histogramDP(
  x,
  eps,
  breaks = "Sturges",
  normalize = FALSE,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP",
  allow.negative = FALSE
)
```

Arguments

`x`	Numeric vector from which the histogram will be formed.
`eps`	Positive real number defining the epsilon privacy budget.
`breaks`	Identical to the argument with the same name from `hist`.
`normalize`	Logical value. If FALSE (default), returned histogram counts correspond to frequencies. If TRUE, returned histogram counts correspond to densities (i.e. area of histogram is one).
`which.sensitivity`	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.
`mechanism`	String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.
`delta`	Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP` String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

`allow.negative` Logical value. If FALSE (default), any negative values in the sanitized histogram due to the added noise will be set to 0. If TRUE, the negative values (if any) will be returned.

Value

Sanitized histogram based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Examples

```
x <- stats::rnorm(500)
graphics::hist(x) # Non-private histogram
result <- histogramDP(x, 1)
plot(result) # Private histogram

graphics::hist(x, freq=FALSE) # Normalized non-private histogram
result <- histogramDP(x, .5, normalize=TRUE, which.sensitivity='unbounded',
  mechanism='Gaussian',delta=0.01, allow.negative=TRUE)
plot(result) # Normalized private histogram (note negative values allowed)
```

`LaplaceMechanism` *Laplace Mechanism*

Description

This function implements the Laplace mechanism for differential privacy by adding noise to the true value(s) of a function according to specified values of epsilon and l1-global sensitivity(-ies). Global sensitivity calculated based either on bounded or unbounded differential privacy can be used (Kifer and Machanavajjhala 2011). If true.values is a vector, the provided epsilon is divided such that epsilon-differential privacy is satisfied across all function values. In the case that each element of true.values comes from its own function with different corresponding sensitivities, a vector of sensitivities may be provided. In this case, if desired, the user can specify how to divide epsilon among the function values using alloc.proportions.

Usage

```
LaplaceMechanism(true.values, eps, sensitivities, alloc.proportions = NULL)
```

Arguments

`true.values`	Real number or numeric vector corresponding to the true value(s) of the desired function.
`eps`	Positive real number defining the epsilon privacy parameter.
`sensitivities`	Real number or numeric vector corresponding to the l1-global sensitivity(-ies) of the function(s) generating true.values. This value must be of length 1 or of the same length as true.values. If it is of length 1 and true.values is a vector, this indicates that the given sensitivity applies simultaneously to all elements of true.values and that the privacy budget need not be allocated (alloc.proportions is unused in this case). If it is of the same length as true.values, this indicates that each element of true.values comes from its own function with different corresponding sensitivities. In this case, the l1-norm of the provided sensitivities is used to generate the Laplace noise.
`alloc.proportions`	Optional numeric vector giving the allocation proportions of epsilon to the function values in the case of vector-valued sensitivities. For example, if sensitivities is of length two and alloc.proportions = c(.75, .25), then 75% of the privacy budget eps is allocated to the noise computation for the first element of true.values, and the remaining 25% is allocated to the noise computation for the second element of true.values. This ensures eps-level privacy across all computations. Input does not need to be normalized, meaning alloc.proportions = c(3,1) produces the same result as the example above.

Value

Sanitized function values based on the bounded and/or unbounded definitions of differential privacy, sanitized via the Laplace mechanism.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Examples

```
# Simulate dataset
n <- 100
c0 <- 5 # Lower bound
c1 <- 10 # Upper bound
D1 <- stats::runif(n, c0, c1)
epsilon <- 1 # Privacy budget
sensitivity <- (c1-c0)/n

private.mean <- LaplaceMechanism(mean(D1), epsilon, sensitivity)
private.mean

# Simulate second dataset
d0 <- 3 # Lower bound
d1 <- 6 # Upper bound
D2 <- stats::runif(n, d0, d1)
D <- matrix(c(D1,D2),ncol=2)
sensitivities <- c((c1-c0)/n, (d1-d0)/n)
epsilon <- 1 # Total privacy budget for all means

# Here, sensitivities are summed and the result is used to generate Laplace
# noise. This is essentially the same as allocating epsilon proportional to
# the corresponding sensitivity. The results satisfy 1-differential privacy.
private.means <- LaplaceMechanism(apply(D, 2, mean), epsilon, sensitivities)
private.means

# Here, privacy budget is explicitly split so that 75% is given to the first
# vector element and 25% is given to the second.
private.means <- LaplaceMechanism(apply(D, 2, mean), epsilon, sensitivities,
                                  alloc.proportions = c(0.75, 0.25))
private.means
```

`LinearRegressionDP` *Privacy-preserving Linear Regression*

Description

This class implements differentially private linear regression using the objective perturbation technique (Kifer et al. 2012).

Details

To use this class for linear regression, first use the `new` method to construct an object of this class with the desired function values and hyperparameters. After constructing the object, the `fit` method

can be applied with a provided dataset and data bounds to fit the model. In fitting, the model stores a vector of coefficients `coeff` which satisfy differential privacy. These can be released directly, or used in conjunction with the `predict` method to privately predict the outcomes of new datapoints.

Note that in order to guarantee differential privacy for linear regression, certain constraints must be satisfied for the values used to construct the object, as well as for the data used to fit. The regularizer must be convex. Additionally, it is assumed that if x represents a single row of the dataset X, then the l2-norm of x is at most p for all x, where p is the number of predictors (including any possible intercept term). In order to ensure this constraint is satisfied, the dataset is preprocessed and scaled, and the resulting coefficients are postprocessed and un-scaled so that the stored coefficients correspond to the original data. Due to this constraint on x, it is best to avoid using an intercept term in the model whenever possible. If an intercept term must be used, the issue can be partially circumvented by adding a constant column to X before fitting the model, which will be scaled along with the rest of X. The `fit` method contains functionality to add a column of constant 1s to X before scaling, if desired.

Super class

`DPpack::EmpiricalRiskMinimizationDP.KST` -> `LinearRegressionDP`

Methods

Public methods:

- `LinearRegressionDP$new()`
- `LinearRegressionDP$fit()`
- `LinearRegressionDP$clone()`

Method `new()`**:** Create a new LinearRegressionDP object.

Usage:

```
LinearRegressionDP$new(regularizer, eps, delta, gamma, regularizer.gr = NULL)
```

Arguments:

`regularizer` String or regularization function. If a string, must be 'l2', indicating to use l2 regularization. If a function, must have form `regularizer(coeff)`, where `coeff` is a vector or matrix, and return the value of the regularizer at `coeff`. See `regularizer.l2` for an example. Additionally, in order to ensure differential privacy, the function must be convex.

`eps` Positive real number defining the epsilon privacy budget. If set to Inf, runs algorithm without differential privacy.

`delta` Nonnegative real number defining the delta privacy parameter. If 0, reduces to pure eps-DP.

`gamma` Nonnegative real number representing the regularization constant.

`regularizer.gr` Optional function representing the gradient of the regularization function with respect to `coeff` and of the form `regularizer.gr(coeff)`. Should return a vector. See `regularizer.gr.l2` for an example. If `regularizer` is given as a string, this value is ignored. If not given and `regularizer` is a function, non-gradient based optimization methods are used to compute the coefficient values in fitting the model.

Returns: A new LinearRegressionDP object.

Method `fit()`**:** Fit the differentially private linear regression model. The function runs the objective perturbation algorithm (Kifer et al. 2012) to generate an objective function. A numerical optimization method is then run to find optimal coefficients for fitting the model given the training data and hyperparameters. The `nloptr` function is used. If `regularizer` is given as 'l2' or if `regularizer.gr` is given in the construction of the object, the gradient of the objective function and the Jacobian of the constraint function are utilized for the algorithm, and the NLOPT_LD_MMA method is used. If this is not the case, the NLOPT_LN_COBYLA method is used. The resulting privacy-preserving coefficients are stored in coeff.

Usage:

```
LinearRegressionDP$fit(X, y, upper.bounds, lower.bounds, add.bias = FALSE)
```

Arguments:

`X` Dataframe of data to be fit.

`y` Vector or matrix of true values for each row of `X`.

`upper.bounds` Numeric vector of length `ncol(X)+1` giving upper bounds on the values in each column of `X` and the values of `y`. The last value in the vector is assumed to be the upper bound on `y`, while the first `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` and in `y` larger than the corresponding upper bound is clipped at the bound.

`lower.bounds` Numeric vector of length `ncol(X)+1` giving lower bounds on the values in each column of `X` and the values of `y`. The last value in the vector is assumed to be the lower bound on `y`, while the first `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` and in `y` larger than the corresponding lower bound is clipped at the bound.

`add.bias` Boolean indicating whether to add a bias term to `X`. Defaults to FALSE.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LinearRegressionDP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Kifer D, Smith A, Thakurta A (2012). "Private Convex Empirical Risk Minimization and High-dimensional Regression." In Mannor S, Srebro N, Williamson RC (eds.), *Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, 25.1–25.40. https://proceedings.mlr.press/v23/kifer12.html.

Examples

```
# Build example dataset
n <- 500
X <- data.frame(X=seq(-1,1,length.out = n))
true.theta <- c(-.3,.5) # First element is bias term
p <- length(true.theta)
y <- true.theta[1] + as.matrix(X)%*%true.theta[2:p] + stats::rnorm(n=n,sd=.1)
```

```
# Construct object for linear regression
regularizer <- 'l2' # Alternatively, function(coeff) coeff%*%coeff/2
eps <- 1
delta <- 0 # Indicates to use pure eps-DP
gamma <- 1
regularizer.gr <- function(coeff) coeff

lrdp <- LinearRegressionDP$new('l2', eps, delta, gamma, regularizer.gr)

# Fit with data
# We must assume y is a matrix with values between -p and p (-2 and 2
#   for this example)
upper.bounds <- c(1, 2) # Bounds for X and y
lower.bounds <- c(-1,-2) # Bounds for X and y
lrdp$fit(X, y, upper.bounds, lower.bounds, add.bias=TRUE)
lrdp$coeff # Gets private coefficients

# Predict new data points
# Build a test dataset
Xtest <- data.frame(X=c(-.5, -.25, .1, .4))
predicted.y <- lrdp$predict(Xtest, add.bias=TRUE)
```

`LogisticRegressionDP` *Privacy-preserving Logistic Regression*

Description

This class implements differentially private logistic regression (Chaudhuri et al. 2011). Either the output or the objective perturbation method can be used.

Details

To use this class for logistic regression, first use the `new` method to construct an object of this class with the desired function values and hyperparameters. After constructing the object, the `fit` method can be applied with a provided dataset and data bounds to fit the model. In fitting, the model stores a vector of coefficients `coeff` which satisfy differential privacy. These can be released directly, or used in conjunction with the `predict` method to privately predict the outcomes of new datapoints.

Note that in order to guarantee differential privacy for logistic regression, certain constraints must be satisfied for the values used to construct the object, as well as for the data used to fit. These conditions depend on the chosen perturbation method. The regularizer must be 1-strongly convex and differentiable. It also must be doubly differentiable if objective perturbation is chosen. Additionally, it is assumed that if x represents a single row of the dataset X, then the l2-norm of x is at most 1 for all x. In order to ensure this constraint is satisfied, the dataset is preprocessed and scaled, and the resulting coefficients are postprocessed and un-scaled so that the stored coefficients correspond to the original data. Due to this constraint on x, it is best to avoid using a bias term in the model whenever possible. If a bias term must be used, the issue can be partially circumvented by adding a constant column to X before fitting the model, which will be scaled along with the rest of X. The `fit` method contains functionality to add a column of constant 1s to X before scaling, if desired.

Super class

DPpack::EmpiricalRiskMinimizationDP.CMS -> LogisticRegressionDP

Methods

Public methods:

- LogisticRegressionDP$new()
- LogisticRegressionDP$fit()
- LogisticRegressionDP$predict()
- LogisticRegressionDP$clone()

Method `new()`**:** Create a new LogisticRegressionDP object.

Usage:

```
LogisticRegressionDP$new(
  regularizer,
  eps,
  gamma,
  perturbation.method = "objective",
  regularizer.gr = NULL
)
```

Arguments:

`regularizer` String or regularization function. If a string, must be 'l2', indicating to use l2 regularization. If a function, must have form `regularizer(coeff)`, where `coeff` is a vector or matrix, and return the value of the regularizer at `coeff`. See `regularizer.l2` for an example. Additionally, in order to ensure differential privacy, the function must be 1-strongly convex and doubly differentiable.

`eps` Positive real number defining the epsilon privacy budget. If set to Inf, runs algorithm without differential privacy.

`gamma` Nonnegative real number representing the regularization constant.

`perturbation.method` String indicating whether to use the 'output' or the 'objective' perturbation methods (Chaudhuri et al. 2011). Defaults to 'objective'.

`regularizer.gr` Optional function representing the gradient of the regularization function with respect to `coeff` and of the form `regularizer.gr(coeff)`. Should return a vector. See `regularizer.gr.l2` for an example. If `regularizer` is given as a string, this value is ignored. If not given and `regularizer` is a function, non-gradient based optimization methods are used to compute the coefficient values in fitting the model.

Returns: A new LogisticRegressionDP object.

Method `fit()`**:** Fit the differentially private logistic regression model. This method runs either the output perturbation or the objective perturbation algorithm (Chaudhuri et al. 2011), depending on the value of perturbation.method used to construct the object, to generate an objective function. A numerical optimization method is then run to find optimal coefficients for fitting the model given the training data and hyperparameters. The built-in `optim` function using the "BFGS" optimization method is used. If `regularizer` is given as 'l2' or if `regularizer.gr` is given in the construction of the object, the gradient of the objective function is utilized by `optim` as well. Otherwise, non-gradient based optimization methods are used. The resulting privacy-preserving coefficients are stored in `coeff`.

Usage:

```
LogisticRegressionDP$fit(X, y, upper.bounds, lower.bounds, add.bias = FALSE)
```

Arguments:

`X` Dataframe of data to be fit.

`y` Vector or matrix of true labels for each row of `X`.

`upper.bounds` Numeric vector of length `ncol(X)` giving upper bounds on the values in each column of X. The `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` larger than the corresponding upper bound is clipped at the bound.

`lower.bounds` Numeric vector of length `ncol(X)` giving lower bounds on the values in each column of `X`. The `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` larger than the corresponding upper bound is clipped at the bound.

`add.bias` Boolean indicating whether to add a bias term to `X`. Defaults to FALSE.

Method `predict()`**:** Predict label(s) for given `X` using the fitted coefficients.

Usage:

```
LogisticRegressionDP$predict(X, add.bias = FALSE, raw.value = FALSE)
```

Arguments:

`X` Dataframe of data on which to make predictions. Must be of same form as `X` used to fit coefficients.

`add.bias` Boolean indicating whether to add a bias term to `X`. Defaults to FALSE. If add.bias was set to TRUE when fitting the coefficients, add.bias should be set to TRUE for predictions.

`raw.value` Boolean indicating whether to return the raw predicted value or the rounded class label. If FALSE (default), outputs the predicted labels 0 or 1. If TRUE, returns the raw score from the logistic regression.

Returns: Matrix of predicted labels or scores corresponding to each row of `X`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LogisticRegressionDP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Chaudhuri K, Monteleoni C, Sarwate AD (2011). "Differentially Private Empirical Risk Minimization." *Journal of Machine Learning Research*, **12**(29), 1069-1109. https://jmlr.org/papers/v12/chaudhuri11a.html.

Chaudhuri K, Monteleoni C (2009). "Privacy-preserving logistic regression." In Koller D, Schuurmans D, Bengio Y, Bottou L (eds.), *Advances in Neural Information Processing Systems*, volume 21. https://proceedings.neurips.cc/paper/2008/file/8065d07da4a77621450aa84fee5656d9-Paper.pdf.

Examples

```
# Build train dataset X and y, and test dataset Xtest and ytest
N <- 200
K <- 2
X <- data.frame()
y <- data.frame()
for (j in (1:K)){
  t <- seq(-.25, .25, length.out = N)
  if (j==1) m <- stats::rnorm(N,-.2, .1)
  if (j==2) m <- stats::rnorm(N, .2, .1)
  Xtemp <- data.frame(x1 = 3*t , x2 = m - t)
  ytemp <- data.frame(matrix(j-1, N, 1))
  X <- rbind(X, Xtemp)
  y <- rbind(y, ytemp)
}
Xtest <- X[seq(1,(N*K),10),]
ytest <- y[seq(1,(N*K),10),,drop=FALSE]
X <- X[-seq(1,(N*K),10),]
y <- y[-seq(1,(N*K),10),,drop=FALSE]

# Construct object for logistic regression
regularizer <- 'l2' # Alternatively, function(coeff) coeff%*%coeff/2
eps <- 1
gamma <- 1
lrdp <- LogisticRegressionDP$new(regularizer, eps, gamma)

# Fit with data
# Bounds for X based on construction
upper.bounds <- c( 1, 1)
lower.bounds <- c(-1,-1)
lrdp$fit(X, y, upper.bounds, lower.bounds) # No bias term
lrdp$coeff # Gets private coefficients

# Predict new data points
predicted.y <- lrdp$predict(Xtest)
n.errors <- sum(predicted.y!=ytest)
```

`meanDP` *Differentially Private Mean*

Description

This function computes the differentially private mean of a given dataset at user-specified privacy levels of epsilon and delta.

Usage

```
meanDP(
```

```
  x,
  eps,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP"
)
```

Arguments

x	Dataset whose mean is desired.
eps	Positive real number defining the epsilon privacy budget.
lower.bound	Scalar representing the global or public lower bound on values of x.
upper.bound	Scalar representing the global or public upper bound on values of x.
which.sensitivity	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.
mechanism	String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See LaplaceMechanism, GaussianMechanism, and AnalyticGaussianMechanism for descriptions of the supported mechanisms.
delta	Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.
type.DP	String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

Value

Sanitized mean based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Examples

```
D <- stats::rnorm(500, mean=3, sd=2)
lb <- -3 # 3 std devs below mean
ub <- 9 # 3 std devs above mean
meanDP(D,  1, lb, ub)
meanDP(D, .5, lb, ub, which.sensitivity='unbounded', mechanism='Gaussian',
  delta=0.01)
```

`medianDP` *Differentially Private Median*

Description

This function computes the differentially private median of an input vector at a user-specified privacy level of epsilon.

Usage

```
medianDP(
  x,
  eps,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "exponential",
  uniform.sampling = TRUE
)
```

Arguments

`x`	Numeric vector of which the median will be taken.
`eps`	Positive real number defining the epsilon privacy budget.
`lower.bound`	Real number giving the global or public lower bound of x.
`upper.bound`	Real number giving the global or public upper bound of x.

`which.sensitivity`
: String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, 0)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.

`mechanism`
: String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'exponential'. See `ExponentialMechanism` for a description of the supported mechanisms.

`uniform.sampling`
: Boolean indicating whether to sample uniformly between sorted dataset values when returning the private quantile. If TRUE, it is possible for this function to return any number between lower.bound and upper.bound. If FALSE, only a value present in the dataset or the lower bound can be returned.

Value

Sanitized median based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Smith A (2011). "Privacy-Preserving Statistical Estimation with Optimal Convergence Rates." In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, 813–822. ISBN 9781450306911, doi:10.1145/1993636.1993743.

Examples

```
D <- stats::rnorm(500)
lower.bound <- -3 # 3 standard deviations below mean
upper.bound <- 3 # 3 standard deviations above mean

eps <- 1
# Get median satisfying pure 1-differential privacy
private.median <- medianDP(D, eps, lower.bound, upper.bound)
private.median

# Require released value to be in dataset
private.median <- medianDP(c(1,0,3,3,2), eps, 0, 4, uniform.sampling = FALSE)
private.median
```

`pooledCovDP` *Differentially Private Pooled Covariance*

Description

This function computes the differentially private pooled covariance from two or more two-column matrices of data at user-specified privacy levels of epsilon and delta.

Usage

```
pooledCovDP(
  ...,
  eps = 1,
  lower.bound1,
  upper.bound1,
  lower.bound2,
  upper.bound2,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP",
  approx.n.max = FALSE
)
```

Arguments

`...`	Two or more matrices, each with two columns from which to compute the pooled covariance.
`eps`	Positive real number defining the epsilon privacy budget.
`lower.bound1, lower.bound2`	Real numbers giving the global or public lower bounds over the first and second columns of all input data, respectively.
`upper.bound1, upper.bound2`	Real numbers giving the global or public upper bounds over the first and second columns of all input data, respectively.
`which.sensitivity`	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.
`mechanism`	String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.

`delta` Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP` String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

`approx.n.max` Logical indicating whether to approximate n.max (defined to be the length of the largest input vector) in the computation of the global sensitivity based on the upper and lower bounds of the data (Liu 2019). Approximation is best if n.max is very large.

Value

Sanitized pooled covariance based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Liu F (2019). "Statistical Properties of Sanitized Results from Differentially Private Laplace Mechanism with Univariate Bounding Constraints." *Transactions on Data Privacy*, **12**(3), 169-195. http://www.tdp.cat/issues16/tdp.a316a18.pdf.

Examples

```
# Build datasets
D1 <- sort(stats::rnorm(500, mean=3, sd=2))
D2 <- sort(stats::rnorm(500, mean=-1, sd=0.5))
D3 <- sort(stats::rnorm(200, mean=3, sd=2))
D4 <- sort(stats::rnorm(200, mean=-1, sd=0.5))
M1 <- matrix(c(D1, D2), ncol=2)
M2 <- matrix(c(D3, D4), ncol=2)

lb1 <- -3 # 3 std devs below mean
lb2 <- -2.5 # 3 std devs below mean
ub1 <- 9 # 3 std devs above mean
ub2 <- .5 # 3 std devs above mean
```

```
# Pooled covariance satisfying (1,0)-differential privacy
private.pooled.cov <- pooledCovDP(M1, M2, eps = 1, lower.bound1 = lb1,
                                  lower.bound2 = lb2, upper.bound1 = ub1,
                                  upper.bound2 = ub2)
private.pooled.cov

# Pooled covariance satisfying approximate (0.9, 0.01)-differential privacy
# and approximating n.max in the sensitivity calculation
private.pooled.cov <- pooledCovDP(M1, M2, eps = 0.9, lower.bound1 = lb1,
                                  lower.bound2 = lb2, upper.bound1 = ub1,
                                  upper.bound2 = ub2, mechanism = 'Gaussian',
                                  delta = 0.01, approx.n.max = TRUE)
private.pooled.cov
```

`pooledVarDP` *Differentially Private Pooled Variance*

Description

This function computes the differentially private pooled variance from two or more vectors of data at user-specified privacy levels of epsilon and delta.

Usage

```
pooledVarDP(
  ...,
  eps = 1,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP",
  approx.n.max = FALSE
)
```

Arguments

`...`	Two or more vectors from which to compute the pooled variance.
`eps`	Positive real number defining the epsilon privacy budget.
`lower.bound`	Real number giving the global or public lower bound of the input data.
`upper.bound`	Real number giving the global or public upper bound of the input data.
`which.sensitivity`	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded

definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.

`mechanism` String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.

`delta` Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP` String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

`approx.n.max` Logical indicating whether to approximate n.max (defined to be the length of the largest input vector) in the computation of the global sensitivity based on the upper and lower bounds of the data (Liu 2019). Approximation is best if n.max is very large.

Value

Sanitized pooled variance based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Liu F (2019). "Statistical Properties of Sanitized Results from Differentially Private Laplace Mechanism with Univariate Bounding Constraints." *Transactions on Data Privacy*, **12**(3), 169-195. http://www.tdp.cat/issues16/tdp.a316a18.pdf.

Examples

```
# Build datasets
D1 <- stats::rnorm(500, mean=3, sd=2)
```

```
D2 <- stats::rnorm(200, mean=3, sd=2)
D3 <- stats::rnorm(100, mean=3, sd=2)
lower.bound <- -3 # 3 standard deviations below mean
upper.bound <- 9 # 3 standard deviations above mean

# Get private pooled variance without approximating n.max
private.pooled.var <- pooledVarDP(D1, D2, D3, eps=1, lower.bound=lower.bound,
                                  upper.bound = upper.bound)
private.pooled.var

# If n.max is sensitive, we can also use
private.pooled.var <- pooledVarDP(D1, D2, D3, eps=1, lower.bound=lower.bound,
                                  upper.bound = upper.bound,
                                  approx.n.max = TRUE)
private.pooled.var
```

---

quantileDP *Differentially Private Quantile*

Description

This function computes the differentially private quantile of an input vector at a user-specified privacy level of epsilon.

Usage

```
quantileDP(
  x,
  quant,
  eps,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "exponential",
  uniform.sampling = TRUE
)
```

Arguments

x	Numeric vector of which the quantile will be taken.
quant	Real number between 0 and 1 indicating which quantile to return.
eps	Positive real number defining the epsilon privacy budget.
lower.bound	Real number giving the global or public lower bound of x.
upper.bound	Real number giving the global or public upper bound of x.

`which.sensitivity`
: String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, 0)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.

`mechanism`
: String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'exponential'. See `ExponentialMechanism` for a description of the supported mechanisms.

`uniform.sampling`
: Boolean indicating whether to sample uniformly between sorted dataset values when returning the private quantile. If TRUE, it is possible for this function to return any number between lower.bound and upper.bound. If FALSE, only a value present in the dataset or the lower bound can be returned.

Value

Sanitized quantile based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Smith A (2011). "Privacy-Preserving Statistical Estimation with Optimal Convergence Rates." In *Proceedings of the Forty-Third Annual ACM Symposium on Theory of Computing*, STOC '11, 813–822. ISBN 9781450306911, doi:10.1145/1993636.1993743.

Examples

```
D <- stats::rnorm(500)
lower.bound <- -3 # 3 standard deviations below mean
upper.bound <- 3 # 3 standard deviations above mean

quant <- 0.25
eps <- 1
# Get 25th quantile satisfying pure 1-differential privacy
private.quantile <- quantileDP(D, quant, eps, lower.bound, upper.bound)
private.quantile

# Get 75th quantile requiring released value to be in dataset
quant <- 0.75
private.quantile <- quantileDP(D, quant, eps, lower.bound, upper.bound,
                               uniform.sampling = FALSE)
```

private.quantile

---

sdDP *Differentially Private Standard Deviation*

Description

This function computes the differentially private standard deviation of a given dataset at user-specified privacy levels of epsilon and delta.

Usage

```
sdDP(
  x,
  eps,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP"
)
```

Arguments

x	Numeric vector whose variance is desired.
eps	Positive real number defining the epsilon privacy budget.
lower.bound	Scalar representing the global or public lower bound on values of x.
upper.bound	Scalar representing the global or public upper bound on values of x.
which.sensitivity	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.
mechanism	String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See LaplaceMechanism, GaussianMechanism, and AnalyticGaussianMechanism for descriptions of the supported mechanisms.
delta	Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP` String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

Value

Sanitized standard deviation based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Liu F (2019). "Statistical Properties of Sanitized Results from Differentially Private Laplace Mechanism with Univariate Bounding Constraints." *Transactions on Data Privacy*, **12**(3), 169-195. http://www.tdp.cat/issues16/tdp.a316a18.pdf.

Examples

```
D <- stats::rnorm(500, mean=3, sd=2)
lb <- -3 # 3 std devs below mean
ub <- 9 # 3 std devs above mean
sdDP(D, 1, lb, ub)
sdDP(D,.5, lb, ub, which.sensitivity='unbounded', mechanism='Gaussian',
  delta=0.01)
```

---

svmDP *Privacy-preserving Support Vector Machine*

Description

This class implements differentially private support vector machine (SVM) (Chaudhuri et al. 2011). It can be either weighted (Yang et al. 2005) or unweighted. Either the output or the objective perturbation method can be used for unweighted SVM, though only the output perturbation method is currently supported for weighted SVM.

Details

To use this class for SVM, first use the `new` method to construct an object of this class with the desired function values and hyperparameters, including a choice of the desired kernel. After constructing the object, the `fit` method can be applied to fit the model with a provided dataset, data bounds, and optional observation weights and weight upper bound. In fitting, the model stores a vector of coefficients `coeff` which satisfy differential privacy. Additionally, if a nonlinear kernel is chosen, the models stores a mapping function from the input data X to a higher dimensional embedding V in the form of a method `XtoV` as required (Chaudhuri et al. 2011). These can be released directly, or used in conjunction with the `predict` method to privately predict the label of new datapoints. Note that the mapping function `XtoV` is based on an approximation method via Fourier transforms (Rahimi and Recht 2007; Rahimi and Recht 2008).

Note that in order to guarantee differential privacy for the SVM model, certain constraints must be satisfied for the values used to construct the object, as well as for the data used to fit. These conditions depend on the chosen perturbation method. First, the loss function is assumed to be differentiable (and doubly differentiable if the objective perturbation method is used). The hinge loss, which is typically used for SVM, is not differentiable at 1. Thus, to satisfy this constraint, this class utilizes the Huber loss, a smooth approximation to the hinge loss (Chapelle 2007). The level of approximation to the hinge loss is determined by a user-specified constant, h, which defaults to 0.5, a typical value. Additionally, the regularizer must be 1-strongly convex and differentiable. It also must be doubly differentiable if objective perturbation is chosen. If weighted SVM is desired, the provided weights must be nonnegative and bounded above by a global or public value, which must also be provided.

Finally, it is assumed that if x represents a single row of the dataset X, then the l2-norm of x is at most 1 for all x. In order to ensure this constraint is satisfied, the dataset is preprocessed and scaled, and the resulting coefficients are postprocessed and un-scaled so that the stored coefficients correspond to the original data. Due to this constraint on x, it is best to avoid using a bias term in the model whenever possible. If a bias term must be used, the issue can be partially circumvented by adding a constant column to X before fitting the model, which will be scaled along with the rest of X. The `fit` method contains functionality to add a column of constant 1s to X before scaling, if desired.

Super classes

`DPpack::EmpiricalRiskMinimizationDP.CMS` -> `DPpack::WeightedERMDP.CMS` -> `svmDP`

Methods

Public methods:

- `svmDP$new()`
- `svmDP$fit()`
- `svmDP$XtoV()`
- `svmDP$predict()`
- `svmDP$clone()`

Method `new()`**:** Create a new `svmDP` object.

Usage:

```
svmDP$new(
  regularizer,
  eps,
  gamma,
  perturbation.method = "objective",
  kernel = "linear",
  D = NULL,
  kernel.param = NULL,
  regularizer.gr = NULL,
  huber.h = 0.5
)
```

Arguments:

`regularizer` String or regularization function. If a string, must be 'l2', indicating to use l2 regularization. If a function, must have form `regularizer(coeff)`, where `coeff` is a vector or matrix, and return the value of the regularizer at `coeff`. See `regularizer.l2` for an example. Additionally, in order to ensure differential privacy, the function must be 1-strongly convex and doubly differentiable.

`eps` Positive real number defining the epsilon privacy budget. If set to Inf, runs algorithm without differential privacy.

`gamma` Nonnegative real number representing the regularization constant.

`perturbation.method` String indicating whether to use the 'output' or the 'objective' perturbation methods (Chaudhuri et al. 2011). Defaults to 'objective'.

`kernel` String indicating which kernel to use for SVM. Must be one of 'linear', 'Gaussian'. If 'linear' (default), linear SVM is used. If 'Gaussian,' uses the sampling function corresponding to the Gaussian (radial) kernel approximation.

`D` Nonnegative integer indicating the dimensionality of the transform space approximating the kernel if a nonlinear kernel is used. Higher values of D provide better kernel approximations at a cost of computational efficiency. This value must be specified if a nonlinear kernel is used.

`kernel.param` Positive real number corresponding to the Gaussian kernel parameter. Defaults to 1/p, where p is the number of predictors.

`regularizer.gr` Optional function representing the gradient of the regularization function with respect to `coeff` and of the form `regularizer.gr(coeff)`. Should return a vector. See `regularizer.gr.l2` for an example. If `regularizer` is given as a string, this value is ignored. If not given and `regularizer` is a function, non-gradient based optimization methods are used to compute the coefficient values in fitting the model.

`huber.h` Positive real number indicating the degree to which the Huber loss approximates the hinge loss. Defaults to 0.5 (Chapelle 2007).

Returns: A new svmDP object.

Method `fit()`**:** Fit the differentially private SVM model. This method runs either the output perturbation or the objective perturbation algorithm (Chaudhuri et al. 2011), depending on the value of perturbation.method used to construct the object, to generate an objective function. A numerical optimization method is then run to find optimal coefficients for fitting the model given the training data, weights, and hyperparameters. The built-in `optim` function using the "BFGS" optimization method is used. If `regularizer` is given as 'l2' or if `regularizer.gr` is given in the construction of the object, the gradient of the objective function is utilized by `optim` as well.

Otherwise, non-gradient based optimization methods are used. The resulting privacy-preserving coefficients are stored in `coeff`.

Usage:

```
svmDP$fit(
  X,
  y,
  upper.bounds,
  lower.bounds,
  add.bias = FALSE,
  weights = NULL,
  weights.upper.bound = NULL
)
```

Arguments:

`X` Dataframe of data to be fit.

`y` Vector or matrix of true labels for each row of `X`.

`upper.bounds` Numeric vector of length `ncol(X)` giving upper bounds on the values in each column of X. The `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` larger than the corresponding upper bound is clipped at the bound.

`lower.bounds` Numeric vector of length `ncol(X)` giving lower bounds on the values in each column of `X`. The `ncol(X)` values are assumed to be in the same order as the corresponding columns of `X`. Any value in the columns of `X` larger than the corresponding upper bound is clipped at the bound.

`add.bias` Boolean indicating whether to add a bias term to `X`. Defaults to FALSE.

`weights` Numeric vector of observation weights of the same length as `y`. If not given, no observation weighting is performed.

`weights.upper.bound` Numeric value representing the global or public upper bound on the weights. Required if weights are given.

Method `XtoV()`**:** Convert input data X into transformed data V. Uses sampled pre-filter values and a mapping function based on the chosen kernel to produce D-dimensional data V on which to train the model or predict future values. This method is only used if the kernel is nonlinear. See Chaudhuri et al. (2011) for more details.

Usage:

```
svmDP$XtoV(X)
```

Arguments:

`X` Matrix corresponding to the original dataset.

Returns: Matrix V of size n by D representing the transformed dataset, where n is the number of rows of X, and D is the provided transformed space dimension.

Method `predict()`**:** Predict label(s) for given `X` using the fitted coefficients.

Usage:

```
svmDP$predict(X, add.bias = FALSE, raw.value = FALSE)
```

Arguments:

`X` Dataframe of data on which to make predictions. Must be of same form as `X` used to fit coefficients.

`add.bias` Boolean indicating whether to add a bias term to `X`. Defaults to FALSE. If add.bias was set to TRUE when fitting the coefficients, add.bias should be set to TRUE for predictions.

`raw.value` Boolean indicating whether to return the raw predicted value or the rounded class label. If FALSE (default), outputs the predicted labels 0 or 1. If TRUE, returns the raw score from the SVM model.

Returns: Matrix of predicted labels or scores corresponding to each row of `X`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
svmDP$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Chaudhuri K, Monteleoni C, Sarwate AD (2011). "Differentially Private Empirical Risk Minimization." *Journal of Machine Learning Research*, **12**(29), 1069-1109. https://jmlr.org/papers/v12/chaudhuri11a.html.

Yang X, Song Q, Cao A (2005). "Weighted support vector machine for data classification." In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, volume 2, 859-864 vol. 2. doi:10.1109/IJCNN.2005.1555965.

Chapelle O (2007). "Training a Support Vector Machine in the Primal." *Neural Computation*, **19**(5), 1155-1178. doi:10.1162/neco.2007.19.5.1155.

Rahimi A, Recht B (2007). "Random Features for Large-Scale Kernel Machines." In Platt J, Koller D, Singer Y, Roweis S (eds.), *Advances in Neural Information Processing Systems*, volume 20. https://proceedings.neurips.cc/paper/2007/file/013a006f03dbc5392effeb8f18fda755-Paper.pdf.

Rahimi A, Recht B (2008). "Weighted Sums of Random Kitchen Sinks: Replacing minimization with randomization in learning." In Koller D, Schuurmans D, Bengio Y, Bottou L (eds.), *Advances in Neural Information Processing Systems*, volume 21. https://proceedings.neurips.cc/paper/2008/file/0efe32849d230d7f53049ddc4a4b0c60-Paper.pdf.

Examples

```
# Build train dataset X and y, and test dataset Xtest and ytest
N <- 400
X <- data.frame()
y <- data.frame()
for (i in (1:N)){
  Xtemp <- data.frame(x1 = stats::rnorm(1,sd=.28) , x2 = stats::rnorm(1,sd=.28))
  if (sum(Xtemp^2)<.15) ytemp <- data.frame(y=0)
  else ytemp <- data.frame(y=1)
  X <- rbind(X, Xtemp)
  y <- rbind(y, ytemp)
```

```
}
Xtest <- X[seq(1,N,10),]
ytest <- y[seq(1,N,10),,drop=FALSE]
X <- X[-seq(1,N,10),]
y <- y[-seq(1,N,10),,drop=FALSE]

# Construct object for SVM
regularizer <- 'l2' # Alternatively, function(coeff) coeff%*%coeff/2
eps <- 1
gamma <- 1
perturbation.method <- 'output'
kernel <- 'Gaussian'
D <- 20
svmdp <- svmDP$new(regularizer, eps, gamma, perturbation.method,
                   kernel=kernel, D=D)

# Fit with data
# Bounds for X based on construction
upper.bounds <- c( 1, 1)
lower.bounds <- c(-1,-1)
weights <- rep(1, nrow(y)) # Uniform weighting
weights[nrow(y)] <- 0.5 # Half weight for last observation
wub <- 1 # Public upper bound for weights
svmdp$fit(X, y, upper.bounds, lower.bounds, weights=weights,
          weights.upper.bound=wub) # No bias term

# Predict new data points
predicted.y <- svmdp$predict(Xtest)
n.errors <- sum(predicted.y!=ytest)
```

---

tableDP *Differentially Private Contingency Table*

Description

This function computes a differentially private contingency table from given vectors of data at user-specified privacy levels of epsilon and delta.

Usage

```
tableDP(
  ...,
  eps = 1,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP",
  allow.negative = FALSE
)
```

Arguments

`...`	Vectors of data from which to create the contingency table.
`eps`	Positive real number defining the epsilon privacy budget.
`which.sensitivity`	String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.
`mechanism`	String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.
`delta`	Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.
`type.DP`	String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.
`allow.negative`	Logical value. If FALSE (default), any negative values in the sanitized table due to the added noise will be set to 0. If TRUE, the negative values (if any) will be returned.

Value

Sanitized contingency table based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Examples

```
x <- MASS::Cars93$Type
y <- MASS::Cars93$Origin
z <- MASS::Cars93$AirBags
tableDP(x,y,eps=1,which.sensitivity='bounded',mechanism='Laplace',
  type.DP='pDP')
tableDP(x,y,z,eps=.5,which.sensitivity='unbounded',mechanism='Gaussian',
  delta=0.01)
```

`tune_classification_model` *Privacy-preserving Hyperparameter Tuning for Binary Classification Models*

Description

This function implements the privacy-preserving hyperparameter tuning function for binary classification (Chaudhuri et al. 2011) using the exponential mechanism. It accepts a list of models with various chosen hyperparameters, a dataset X with corresponding labels y, upper and lower bounds on the columns of X, and a boolean indicating whether to add bias in the construction of each of the models. The data are split into m+1 equal groups, where m is the number of models being compared. One group is set aside as the validation group, and each of the other m groups are used to train each of the given m models. The number of errors on the validation set is counted for each model and used as the utility values in the exponential mechanism (`ExponentialMechanism`) to select a tuned model in a privacy-preserving way.

Usage

```
tune_classification_model(
  models,
  X,
  y,
  upper.bounds,
  lower.bounds,
  add.bias = FALSE,
  weights = NULL,
  weights.upper.bound = NULL
)
```

Arguments

`models`	Vector of binary classification model objects, each initialized with a different combination of hyperparameter values from the search space for tuning. Each model should be initialized with the same epsilon privacy parameter value eps. The tuned model satisfies eps-level differential privacy.

X	Dataframe of data to be used in tuning the model. Note it is assumed the data rows and corresponding labels are randomly shuffled.
y	Vector or matrix of true labels for each row of X.
upper.bounds	Numeric vector giving upper bounds on the values in each column of X. Should be of length ncol(X). The values are assumed to be in the same order as the corresponding columns of X. Any value in the columns of X larger than the corresponding upper bound is clipped at the bound.
lower.bounds	Numeric vector giving lower bounds on the values in each column of X. Should be of length ncol(X). The values are assumed to be in the same order as the corresponding columns of X. Any value in the columns of X smaller than the corresponding lower bound is clipped at the bound.
add.bias	Boolean indicating whether to add a bias term to X. Defaults to FALSE.
weights	Numeric vector of observation weights of the same length as y.
weights.upper.bound	Numeric value representing the global or public upper bound on the weights.

Value

Single model object selected from the input list models with tuned parameters.

References

Chaudhuri K, Monteleoni C, Sarwate AD (2011). "Differentially Private Empirical Risk Minimization." *Journal of Machine Learning Research*, **12**(29), 1069-1109. https://jmlr.org/papers/v12/chaudhuri11a.html.

Examples

```
# Build train dataset X and y, and test dataset Xtest and ytest
N <- 200
K <- 2
X <- data.frame()
y <- data.frame()
for (j in (1:K)){
  t <- seq(-.25,.25,length.out = N)
  if (j==1) m <- stats::rnorm(N,-.2,.1)
  if (j==2) m <- stats::rnorm(N, .2,.1)
  Xtemp <- data.frame(x1 = 3*t , x2 = m - t)
  ytemp <- data.frame(matrix(j-1, N, 1))
  X <- rbind(X, Xtemp)
  y <- rbind(y, ytemp)
}
Xtest <- X[seq(1,(N*K),10),]
ytest <- y[seq(1,(N*K),10),,drop=FALSE]
X <- X[-seq(1,(N*K),10),]
y <- y[-seq(1,(N*K),10),,drop=FALSE]
y <- as.matrix(y)
weights <- rep(1, nrow(y)) # Uniform weighting
weights[nrow(y)] <- 0.5 # half weight for last observation
```

```
wub <- 1 # Public upper bound for weights

# Grid of possible gamma values for tuning logistic regression model
grid.search <- c(100, 1, .0001)

# Construct objects for SVM parameter tuning
eps <- 1 # Privacy budget should be the same for all models
svmdp1 <- svmDP$new("l2", eps, grid.search[1], perturbation.method='output')
svmdp2 <- svmDP$new("l2", eps, grid.search[2], perturbation.method='output')
svmdp3 <- svmDP$new("l2", eps, grid.search[3], perturbation.method='output')
models <- c(svmdp1, svmdp2, svmdp3)

# Tune using data and bounds for X based on its construction
upper.bounds <- c( 1, 1)
lower.bounds <- c(-1,-1)
tuned.model <- tune_classification_model(models, X, y, upper.bounds,
                                         lower.bounds, weights=weights,
                                         weights.upper.bound=wub)
tuned.model$gamma # Gives resulting selected hyperparameter

# tuned.model result can be used the same as a trained LogisticRegressionDP model
# Predict new data points
predicted.y <- tuned.model$predict(Xtest)
n.errors <- sum(predicted.y!=ytest)
```

`tune_linear_regression_model` *Privacy-preserving Hyperparameter Tuning for Linear Regression Models*

Description

This function implements the privacy-preserving hyperparameter tuning function for linear regression (Kifer et al. 2012) using the exponential mechanism. It accepts a list of models with various chosen hyperparameters, a dataset X with corresponding values y, upper and lower bounds on the columns of X and the values of y, and a boolean indicating whether to add bias in the construction of each of the models. The data are split into m+1 equal groups, where m is the number of models being compared. One group is set aside as the validation group, and each of the other m groups are used to train each of the given m models. The negative of the sum of the squared error for each model on the validation set is used as the utility values in the exponential mechanism (`ExponentialMechanism`) to select a tuned model in a privacy-preserving way.

Usage

```
tune_linear_regression_model(
  models,
  X,
  y,
```

```
  upper.bounds,
  lower.bounds,
  add.bias = FALSE
)
```

Arguments

models	Vector of linear regression model objects, each initialized with a different combination of hyperparameter values from the search space for tuning. Each model should be initialized with the same epsilon privacy parameter value eps. The tuned model satisfies eps-level differential privacy.
X	Dataframe of data to be used in tuning the model. Note it is assumed the data rows and corresponding labels are randomly shuffled.
y	Vector or matrix of true values for each row of X.
upper.bounds	Numeric vector giving upper bounds on the values in each column of X and the values in y. Should be length ncol(X)+1. The first ncol(X) values are assumed to be in the same order as the corresponding columns of X, while the last value in the vector is assumed to be the upper bound on y. Any value in the columns of X and y larger than the corresponding upper bound is clipped at the bound.
lower.bounds	Numeric vector giving lower bounds on the values in each column of X and the values in y. Should be length ncol(X)+1. The first ncol(X) values are assumed to be in the same order as the corresponding columns of X, while the last value in the vector is assumed to be the lower bound on y. Any value in the columns of X and y smaller than the corresponding lower bound is clipped at the bound.
add.bias	Boolean indicating whether to add a bias term to X. Defaults to FALSE.

Value

Single model object selected from the input list models with tuned parameters.

References

Kifer D, Smith A, Thakurta A (2012). "Private Convex Empirical Risk Minimization and High-dimensional Regression." In Mannor S, Srebro N, Williamson RC (eds.), *Proceedings of the 25th Annual Conference on Learning Theory*, volume 23 of *Proceedings of Machine Learning Research*, 25.1–25.40. https://proceedings.mlr.press/v23/kifer12.html.

Examples

```
# Build example dataset
n <- 500
X <- data.frame(X=seq(-1,1,length.out = n))
true.theta <- c(-.3,.5) # First element is bias term
p <- length(true.theta)
y <- true.theta[1] + as.matrix(X)%*%true.theta[2:p] + stats::rnorm(n=n,sd=.1)

# Grid of possible gamma values for tuning linear regression model
grid.search <- c(100, 1, .0001)
```

```
# Construct objects for logistic regression parameter tuning
# Privacy budget should be the same for all models
eps <- 1
delta <- 0.01
linrdp1 <- LinearRegressionDP$new("l2", eps, delta, grid.search[1])
linrdp2 <- LinearRegressionDP$new("l2", eps, delta, grid.search[2])
linrdp3 <- LinearRegressionDP$new("l2", eps, delta, grid.search[3])
models <- c(linrdp1, linrdp2, linrdp3)

# Tune using data and bounds for X and y based on their construction
upper.bounds <- c( 1, 2) # Bounds for X and y
lower.bounds <- c(-1,-2) # Bounds for X and y
tuned.model <- tune_linear_regression_model(models, X, y, upper.bounds,
                                            lower.bounds, add.bias=TRUE)
tuned.model$gamma # Gives resulting selected hyperparameter

# tuned.model result can be used the same as a trained LogisticRegressionDP model
tuned.model$coeff # Gives coefficients for tuned model

# Build a test dataset for prediction
Xtest <- data.frame(X=c(-.5, -.25, .1, .4))
predicted.y <- tuned.model$predict(Xtest, add.bias=TRUE)
```

---

varDP *Differentially Private Variance*

Description

This function computes the differentially private variance of a given dataset at user-specified privacy levels of epsilon and delta.

Usage

```
varDP(
  x,
  eps,
  lower.bound,
  upper.bound,
  which.sensitivity = "bounded",
  mechanism = "Laplace",
  delta = 0,
  type.DP = "aDP"
)
```

Arguments

x	Numeric vector whose variance is desired.
eps	Positive real number defining the epsilon privacy budget.

`lower.bound` Scalar representing the global or public lower bound on values of x.

`upper.bound` Scalar representing the global or public upper bound on values of x.

`which.sensitivity`
String indicating which type of sensitivity to use. Can be one of 'bounded', 'unbounded', 'both'. If 'bounded' (default), returns result based on bounded definition for differential privacy. If 'unbounded', returns result based on unbounded definition. If 'both', returns result based on both methods (Kifer and Machanavajjhala 2011). Note that if 'both' is chosen, each result individually satisfies (eps, delta)-differential privacy, but may not do so collectively and in composition. Care must be taken not to violate differential privacy in this case.

`mechanism` String indicating which mechanism to use for differential privacy. Currently the following mechanisms are supported: 'Laplace', 'Gaussian', 'analytic'. Default is Laplace. See `LaplaceMechanism`, `GaussianMechanism`, and `AnalyticGaussianMechanism` for descriptions of the supported mechanisms.

`delta` Nonnegative real number defining the delta privacy parameter. If 0 (default), reduces to eps-DP.

`type.DP` String indicating the type of differential privacy desired for the Gaussian mechanism (if selected). Can be either 'pDP' for probabilistic DP (Machanavajjhala et al. 2008) or 'aDP' for approximate DP (Dwork et al. 2006). Note that if 'aDP' is chosen, epsilon must be strictly less than 1.

Value

Sanitized variance based on the bounded and/or unbounded definitions of differential privacy.

References

Dwork C, McSherry F, Nissim K, Smith A (2006). "Calibrating Noise to Sensitivity in Private Data Analysis." In Halevi S, Rabin T (eds.), *Theory of Cryptography*, 265–284. ISBN 978-3-540-32732-5, https://doi.org/10.1007/11681878_14.

Kifer D, Machanavajjhala A (2011). "No Free Lunch in Data Privacy." In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, 193–204. ISBN 9781450306614, doi:10.1145/1989323.1989345.

Machanavajjhala A, Kifer D, Abowd J, Gehrke J, Vilhuber L (2008). "Privacy: Theory meets Practice on the Map." In *2008 IEEE 24th International Conference on Data Engineering*, 277-286. doi:10.1109/ICDE.2008.4497436.

Dwork C, Kenthapadi K, McSherry F, Mironov I, Naor M (2006). "Our Data, Ourselves: Privacy Via Distributed Noise Generation." In Vaudenay S (ed.), *Advances in Cryptology - EUROCRYPT 2006*, 486–503. ISBN 978-3-540-34547-3, doi:10.1007/11761679_29.

Liu F (2019). "Statistical Properties of Sanitized Results from Differentially Private Laplace Mechanism with Univariate Bounding Constraints." *Transactions on Data Privacy*, **12**(3), 169-195. http://www.tdp.cat/issues16/tdp.a316a18.pdf.

Examples

```
D <- stats::rnorm(500, mean=3, sd=2)
lb <- -3 # 3 std devs below mean
```

```
ub <- 9 # 3 std devs above mean
varDP(D, 1, lb, ub)
varDP(D,.5, lb, ub, which.sensitivity='unbounded', mechanism='Gaussian',
  delta=0.01)
```

Index