Diffprivlib Documentation

Release 0.3.0

Naoise Holohan

Jul 28, 2021

MODULES

This is a library dedicated to differential privacy and machine learning. Its purpose is to allow experimentation, simulation, and implementation of differentially private models using a common codebase and building blocks.

CHAPTER

ONE

DIFFPRIVLIB.ACCOUNTANT

Privacy budget accountant for differential privacy

1.1 Base class

class diffprivlib.accountant.**BudgetAccountant**(*epsilon=inf*, *delta=1.0*, *slack=0.0*, *spent_budget=None*)

Privacy budget accountant for differential privacy.

This class creates a privacy budget accountant to track privacy spend across queries and other data accesses. Once initialised, the BudgetAccountant stores each privacy spend and iteratively updates the total budget spend, raising an error when the budget ceiling (if specified) is exceeded. The accountant can be initialised without any maximum budget, to enable users track the total privacy spend of their actions without hindrance.

Diffprivlib functions can make use of a BudgetAccountant in three different ways (see examples for more details):

- Passed as an `accountant` parameter to the function (e.g., `mean(..., accountant=acc)`)
- Set as the default using the `set_default()` method (all subsequent diffprivlib functions will use the accountant by default)
- As a context manager using a `with` statement (the accountant is used for that block of code)

Implements the accountant rules as given in [KOV17].

Parameters

- **epsilon** (*float, default: infinity*) – Epsilon budget ceiling of the accountant.
- **delta** (*float, default: 1.0*) – Delta budget ceiling of the accountant.
- **slack** (*float, default: 0.0*) – Slack allowed in delta spend. Greater slack may reduce the overall epsilon spend.
- **spent_budget** (*list of tuples of the form (epsilon, delta), optional*) – List of tuples of pre-existing budget spends. Allows for a new accountant to be initialised with spends extracted from a previous instance.

epsilon

Epsilon budget ceiling of the accountant.

Type float

delta

Delta budget ceiling of the accountant.

Type float

slack
The accountant's slack. Can be modified at runtime, subject to the privacy budget not being exceeded.

Type float

spent_budget
The list of privacy spends recorded by the accountant. Can be used in the initialisation of a new accountant.

Type list of tuples of the form (epsilon, delta)

Examples

A `BudgetAccountant` is typically passed to diffprivlib functions as an `accountant` parameter. If `epsilon` and `delta` are not set, the accountant has an infinite budget by default, allowing you to track privacy spend without imposing a hard limit. By allowing a `slack` in the budget calculation, the overall epsilon privacy spend can be reduced (at the cost of extra delta spend).

```
>>> import diffprivlib as dp
>>> from numpy.random import random
>>> X = random(100)
>>> acc = dp.BudgetAccountant(epsilon=1.5, delta=0)
>>> dp.tools.mean(X, bounds=(0, 1), accountant=acc)
0.4547006207923884
>>> acc.total()
(epsilon=1.0, delta=0)
>>> dp.tools.std(X, bounds=(0, 1), epsilon=0.25, accountant=acc)
0.2630216611181259
>>> acc.total()
(epsilon=1.25, delta=0)
```

```
>>> acc2 = dp.BudgetAccountant() # infinite budget
>>> first_half = dp.tools.mean(X[:50], epsilon=0.25, bounds=(0, 1),␣
↪accountant=acc2)
>>> last_half = dp.tools.mean(X[50:], epsilon=0.25, bounds=(0, 1),␣
↪accountant=acc2)
>>> acc2.total()
(epsilon=0.5, delta=0)
>>> acc2.remaining()
(epsilon=inf, delta=1.0)
```

```
>>> acc3 = dp.BudgetAccountant(slack=1e-3)
>>> for i in range(20):
...     dp.tools.mean(X, epsilon=0.05, bounds=(0, 1), accountant=acc3)
>>> acc3.total() # Slack has reduced the epsilon spend by almost 25%
(epsilon=0.7613352285668463, delta=0.001)
```

Using `set_default()`, an accountant is used by default in all diffprivlib functions in that script. Accountants also act as context managers, allowing for use in a `with` statement. Passing an accountant as a parameter overrides all other methods.

```
>>> acc4 = dp.BudgetAccountant()
>>> acc4.set_default()
BudgetAccountant()
>>> Y = random((100, 2)) - 0.5
>>> clf = dp.models.PCA(1, centered=True, data_norm=1.4)
```

(continues on next page)

(continued from previous page)

```
>>> clf.fit(Y)
PCA(accountant=BudgetAccountant(spent_budget=[(1.0, 0)]), centered=True,␣
↪copy=True, data_norm=1.4, epsilon=1.0,
n_components=1, random_state=None, bounds=None, whiten=False)
>>> acc4.total()
(epsilon=1.0, delta=0)
```

```
>>> with dp.BudgetAccountant() as acc5:
...     dp.tools.mean(Y, bounds=(0, 1), epsilon=1/3)
>>> acc5.total()
(epsilon=0.3333333333333333, delta=0)
```

References

check (*epsilon*, *delta*)

Checks if the provided (epsilon,delta) can be spent without exceeding the accountant's budget ceiling.

Parameters

- **epsilon** (*float*) – Epsilon budget spend to check.
- **delta** (*float*) – Delta budget spend to check.

Returns True if the budget can be spent, otherwise a *BudgetError* is raised.

Return type bool

Raises ***BudgetError*** – If the specified budget spend will result in the budget ceiling being exceeded.

static load_default (*accountant*)

Loads the default privacy budget accountant if none is supplied, otherwise checks that the supplied accountant is a BudgetAccountant class.

An accountant can be set as the default using the *set_default()* method. If no default has been set, a default is created.

Parameters **accountant** (*BudgetAccountant or None*) – The supplied budget accountant. If None, the default accountant is returned.

Returns **default** – Returns a working BudgetAccountant, either the supplied *accountant* or the existing default.

Return type *BudgetAccountant*

static pop_default ()

Pops the default BudgetAccountant from the class and returns it to the user.

Returns **default** – Returns the existing default BudgetAccountant.

Return type *BudgetAccountant*

remaining (*k=1*)

Calculates the budget that remains to be spent.

Calculates the privacy budget that can be spent on *k* queries. Spending this budget on *k* queries will match the budget ceiling, assuming no floating point errors.

Parameters **k** (*int, default: 1*) – The number of queries for which to calculate the remaining budget.

Returns

- **epsilon** (*float*) – Total epsilon spend remaining for *k* queries.
- **delta** (*float*) – Total delta spend remaining for *k* queries.

set_default()

Sets the current accountant to be the default when running functions and queries with diffprivlib.

Returns self

Return type *BudgetAccountant*

spend(*epsilon*, *delta*)

Spend the given privacy budget.

Instructs the accountant to spend the given epsilon and delta privacy budget, while ensuring the target budget is not exceeded.

Parameters

- **epsilon** (*float*) – Epsilon privacy budget to spend.
- **delta** (*float*) – Delta privacy budget to spend.

Returns self

Return type *BudgetAccountant*

total(*spent_budget=None*, *slack=None*)

Returns the total current privacy spend.

spent_budget and *slack* can be specified as parameters, otherwise the class values will be used.

Parameters

- **spent_budget** (*list of tuples of the form (epsilon, delta), optional*) – List of tuples of budget spends. If not provided, the accountant's spends will be used.
- **slack** (*float, optional*) – Slack in delta for composition. If not provided, the accountant's slack will be used.

Returns

- **epsilon** (*float*) – Total epsilon spend.
- **delta** (*float*) – Total delta spend.

CHAPTER

TWO

DIFFPRIVLIB.MECHANISMS

Basic mechanisms for achieving differential privacy, the basic building blocks of the library.

2.1 Base classes

class diffprivlib.mechanisms.**DPMachine**

Parent class for *DPMechanism* and *DPTransformer*, providing and specifying basic functionality.

copy()

Produces a copy of the class.

Returns **self** – Returns the copy.

Return type class

deepcopy()

Produces a deep copy of the class.

Returns **self** – Returns the deep copy.

Return type class

abstract randomise(*value*)

Randomise *value* with the mechanism.

Parameters **value** (*int or float or str or method*) – The value to be randomised.

Returns The randomised value, same type as *value*.

Return type int or float or str or method

set_epsilon(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon* > *0*.

Returns **self**

Return type class

abstract set_epsilon_delta(*epsilon*, *delta*)

Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns self

Return type class

class diffprivlib.mechanisms.**DPMechanism**

Base class for all mechanisms. Instantiated from *DPMachine*.

Notes

- Each *DPMechanism* must define a *randomise* method, to handle the application of differential privacy
- Mechanisms that only operate in a limited window of ϵ or δ must define a *set_epsilon_delta* method. Error-checking, for example for non-zero δ should be done in *set_epsilon_delta*; *set_epsilon* should be left unchanged.
- When new methods are added, *__repr__* should be updated accordingly in the mechanism.
- Each mechanism's

check_inputs(*value*)

Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*int or float or str or method*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

copy()

Produces a copy of the class.

Returns self – Returns the copy.

Return type class

deepcopy()

Produces a deep copy of the class.

Returns self – Returns the deep copy.

Return type class

get_bias(*value*)

Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_mse(*value*)
Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)
Returns the variance of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

abstract randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*int or float or str or method*) – The value to be randomised.

Returns The randomised value, same type as *value*.

Return type int or float or str or method

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

set_epsilon_delta(*epsilon*, *delta*)
Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns **self**

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

class diffprivlib.mechanisms.**TruncationAndFoldingMixin**
Mixin for truncating or folding the outputs of a mechanism. Must be instantiated with a *DPMechanism*.

check_inputs (*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters value (*float*) –

Returns

Return type True if the mechanism is ready to be used.

set_bounds (*lower, upper*)
Sets the lower and upper bounds of the mechanism.

Must have lower <= upper.

Parameters

- **lower** (*float*) – The lower bound of the mechanism.
- **upper** (*float*) – The upper bound of the mechanism.

Returns self

Return type class

2.2 Binary mechanism

class diffprivlib.mechanisms.**Binary**
The classic binary mechanism in differential privacy.

Given a binary input value, the mechanism randomly decides to flip to the other binary value or not, in order to satisfy differential privacy.

Paper link: https://arxiv.org/pdf/1612.05568.pdf

Notes

- The binary attributes, known as *labels*, must be specified as strings. If non-string labels are required (e.g. integer-valued labels), a *DPTransformer* can be used (e.g. *IntToString*).

check_inputs (*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters value (*str*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises Exception – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

randomise (*value*)
Randomise *value* with the mechanism.

Parameters value (*str*) – The value to be randomised.

Returns The randomised value.

Return type str

set_epsilon (*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_epsilon_delta (*epsilon, delta*)
Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns self

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

set_labels (*value0, value1*)
Sets the binary labels of the mechanism.

Labels must be unique, non-empty strings. If non-string labels are required, consider using a *DPTransformer*.

Parameters

- **value0** (*str*) – 0th binary label.
- **value1** (*str*) – 1st binary label.

Returns self

Return type class

2.3 Exponential mechanisms

class diffprivlib.mechanisms.**Exponential**
The exponential mechanism for achieving differential privacy on categorical inputs, as first proposed by McSherry and Talwar.

The exponential mechanism achieves differential privacy by randomly choosing an output value for a given input value, with greater probability given to values 'closer' to the input, as measured by a given utility function.

Paper link: https://www.cs.drexel.edu/~greenie/privacy/mdviadp.pdf

check_inputs (*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **`value`** (*`str`*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **`Exception`** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

`get_utility_list`()

Gets the utility list of the mechanism, in the same form as accepted by *.set_utility_list*.

Returns **utility_list** – Returns a list of tuples of the form ("value1", "value2", utility), or *None* if the utility has not yet been set.

Return type list of tuples (str, str, float), or None

`randomise`(*value*)

Randomise *value* with the mechanism.

Parameters **`value`** (*`str`*) – The value to be randomised.

Returns The randomised value.

Return type str

`set_epsilon`(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **`epsilon`** (*`float`*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

`set_utility`(*utility_list*)

Sets the utility function of the mechanism. The utility function is used to determine the probability of selecting an output for a given input.

The utility function is set by *utility_list*, which is a list of pairwise 'distances' between values in the mechanism's domain. As the mechanisms's domain is set by the values in *utility_list*, all possible pairs in *utility_list* must be accounted for. The utility function is symmetric, meaning the distance from *a* to *b* is the same as the distance from *b* to *a*. Setting the second distance will overwrite the first.

Parameters **`utility_list`** (*`list of tuples`*) – The utility list of the mechanism. Must be specified as a list of tuples, of the form ("value1", "value2", utility), where each *value* is a string and *utility* is a strictly positive float. A *utility* must be specified for every pair of values given in the *utility_list*.

Returns **self**

Return type class

Raises

- **`TypeError`** – If the *value* components of each tuple are not strings of if the *utility* component is not a float.
- **`ValueError`** – If the *utility* component is zero or negative.

class diffprivlib.mechanisms.**ExponentialHierarchical**

Adaptation of the exponential mechanism to hierarchical data. Simplifies the process of specifying utility values, as the values can be inferred from the hierarchy.

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*str*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_utility_list()
Gets the utility list of the mechanism, in the same form as accepted by *.set_utility_list*.

Returns **utility_list** – Returns a list of tuples of the form ("value1", "value2", utility), or *None* if the utility has not yet been set.

Return type list of tuples (str, str, float), or None

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*str*) – The value to be randomised.

Returns The randomised value.

Return type str

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

set_hierarchy(*list_hierarchy*)
Sets the hierarchy of the hierarchical exponential mechanism.

The hierarchy is specified as a list of lists, where each leaf node is a string, and lies at the same depth as each other leaf node. The utility between each leaf node is then calculated as

Parameters **list_hierarchy** (*nested list of str*) – The hierarchy as specified as a nested list of string. Each string must be a leaf node, and each leaf node must lie at the same depth in the hierarchy.

Returns **self**

Return type class

2.4 Gaussian mechanisms

class diffprivlib.mechanisms.**Gaussian**

The Gaussian mechanism in differential privacy.

As first proposed by Dwork and Roth in "The algorithmic foundations of differential privacy".

Paper link: https://www.nowpublishers.com/article/DownloadSummary/TCS-042

check_inputs(*value*)

Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters value (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises Exception – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)

Returns the bias of the mechanism at a given *value*.

Parameters value (*int or float*) – The value at which the bias of the mechanism is sought.

Returns bias – The bias of the mechanism at *value*.

Return type float or None

get_mse(*value*)

Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters value (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns bias – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)

Returns the variance of the mechanism at a given *value*.

Parameters value (*float*) – The value at which the variance of the mechanism is sought.

Returns bias – The variance of the mechanism at *value*.

Return type float

randomise(*value*)

Randomise *value* with the mechanism.

Parameters value (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon_delta(*epsilon*, *delta*)

Sets the privacy parameters ϵ and δ for the mechanism.

For the Gaussian mechanism, *epsilon* cannot be greater than 1, and *delta* must be non-zero.

Parameters

- **epsilon** (*float*) – Epsilon value of the mechanism. Must satisfy 0 < *epsilon* <= 1.
- **delta** (*float*) – Delta value of the mechanism. Must satisfy 0 < *delta* <= 1.

Returns self

Return type class

set_sensitivity (*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**GaussianAnalytic**
The analytic Gaussian mechanism in differential privacy.

As first proposed by Balle and Wang in "Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising".

Paper link: https://arxiv.org/pdf/1805.06530.pdf

check_inputs (*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias (*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

get_mse (*value*)
Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance (*value*)
Returns the variance of the mechanism at a given *value*.

Parameters **value** (*float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value*.

Return type float

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon_delta(*epsilon*, *delta*)
Sets the privacy parameters ϵ and δ for the mechanism.

For the analytic Gaussian mechanism, *epsilon* and *delta* must be non-zero.

Parameters

- **epsilon** (*float*) – Epsilon value of the mechanism. Must satisfy 0 < *epsilon*.
- **delta** (*float*) – Delta value of the mechanism. Must satisfy 0 < *delta* < 1.

Returns self

Return type class

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**GaussianDiscrete**
The Discrete Gaussian mechanism in differential privacy.

As proposed by Canonne, Kamath and Steinke, re-purposed for approximate differential privacy.

Paper link: https://arxiv.org/pdf/2004.00010.pdf

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*int*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*int*) – The value to be randomised.

Returns The randomised value.

Return type int

set_epsilon_delta(*epsilon*, *delta*)
Sets the privacy parameters ϵ and δ for the mechanism.

For the discrete Gaussian mechanism, *epsilon* and *delta* must be non-zero.

Parameters

- **epsilon** (*float*) – Epsilon value of the mechanism. Must satisfy 0 < *epsilon*.
- **delta** (*float*) – Delta value of the mechanism. Must satisfy 0 < *delta* < 1.

Returns self

Return type class

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*int*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

2.5 Geometric mechanisms

class diffprivlib.mechanisms.**Geometric**
The classic geometric mechanism for differential privacy, as first proposed by Ghosh, Roughgarden and Sundararajan. Extended to allow for non-unity sensitivity.

Paper link: https://arxiv.org/pdf/0811.2841.pdf

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*int*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_mse(*value*)
Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)

Returns the variance of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

randomise(*value*)

Randomise *value* with the mechanism.

Parameters **value** (*int*) – The value to be randomised.

Returns The randomised value.

Return type int

set_epsilon(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

set_sensitivity(*sensitivity*)

Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*int*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns **self**

Return type class

class diffprivlib.mechanisms.**GeometricTruncated**

The truncated geometric mechanism, where values that fall outside a pre-described range are mapped back to the closest point within the range.

check_inputs(*value*)

Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*int*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

randomise(*value*)

Randomise *value* with the mechanism.

Parameters **value** (*int*) – The value to be randomised.

Returns The randomised value.

Return type int

set_bounds(*lower*, *upper*)
Sets the lower and upper bounds of the mechanism.

For the truncated geometric mechanism, *lower* and *upper* must be integer-valued. Must have *lower* <= *upper*.

Parameters

- **lower** (*int*) – The lower bound of the mechanism.
- **upper** (*int*) – The upper bound of the mechanism.

Returns self

Return type class

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*int*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**GeometricFolded**
The folded geometric mechanism, where values outside a pre-described range are folded back toward the domain around the closest point within the domain. Half-integer bounds are permitted.

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*int*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*int*) – The value to be randomised.

Returns The randomised value.

Return type int

set_bounds (*lower*, *upper*)
Sets the lower and upper bounds of the mechanism.

For the folded geometric mechanism, *lower* and *upper* must be integer or half-integer -valued. Must have *lower* <= *upper*.

Parameters

- **lower** (*int or float*) – The lower bound of the mechanism.
- **upper** (*int or float*) – The upper bound of the mechanism.

Returns self

Return type class

set_epsilon (*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_sensitivity (*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*int*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

2.6 Laplace mechanisms

class diffprivlib.mechanisms.**Laplace**
The classic Laplace mechanism in differential privacy, as first proposed by Dwork, McSherry, Nissim and Smith.

Paper link: https://link.springer.com/content/pdf/10.1007/11681878_14.pdf

Includes extension to (relaxed) (ϵ, δ)-differential privacy, as proposed by Holohan et al.

Paper link: https://arxiv.org/pdf/1402.6124.pdf

check_inputs (*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias (*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

get_mse(*value*)

Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)

Returns the variance of the mechanism at a given *value*.

Parameters **value** (*float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value*.

Return type float

randomise(*value*)

Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

set_epsilon_delta(*epsilon*, *delta*)

Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns **self**

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**LaplaceTruncated**
The truncated Laplace mechanism, where values outside a pre-described domain are mapped to the closest point within the domain.

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

get_mse(*value*)
Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)
Returns the variance of the mechanism at a given *value*.

Parameters **value** (*float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value*.

Return type float

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_bounds(*lower*, *upper*)
Sets the lower and upper bounds of the mechanism.

Must have lower <= upper.

Parameters

- **lower** (*float*) – The lower bound of the mechanism.
- **upper** (*float*) – The upper bound of the mechanism.

Returns self

Return type class

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_epsilon_delta(*epsilon*, *delta*)
Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ,δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ,δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns self

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**LaplaceBoundedDomain**
The bounded Laplace mechanism on a bounded domain. The mechanism draws values directly from the domain, without any post-processing.

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises `Exception` – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

get_effective_epsilon()
Gets the effective epsilon of the mechanism, only for strict ϵ-differential privacy. Returns `None` if δ is non-zero.

Returns The effective ϵ parameter of the mechanism. Returns `None` if *delta* is non-zero.

Return type float

get_mse(*value*)
Returns the mean squared error (MSE) of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the MSE of the mechanism is sought.

Returns **bias** – The MSE of the mechanism at *value* if defined, *None* otherwise.

Return type float or None

get_variance(*value*)
Returns the variance of the mechanism at a given *value*.

Parameters **value** (*float*) – The value at which the variance of the mechanism is sought.

Returns **bias** – The variance of the mechanism at *value*.

Return type float

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_bounds(*lower*, *upper*)
Sets the lower and upper bounds of the mechanism.

Must have lower <= upper.

Parameters

- **lower** (*float*) – The lower bound of the mechanism.
- **upper** (*float*) – The upper bound of the mechanism.

Returns **self**

Return type class

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_epsilon_delta(*epsilon, delta*)
Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns self

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**LaplaceBoundedNoise**
The Laplace mechanism with bounded noise, only applicable for approximate differential privacy (delta > 0).

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon_delta(*epsilon*, *delta*)
Set the privacy parameters ϵ and δ for the mechanism.

Epsilon must be strictly positive, *epsilon* > 0. *delta* must be strictly in the interval (0, 0.5).

- For zero *epsilon*, use `Uniform`.
- For zero *delta*, use `Laplace`.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon > 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 < delta < 0.5*.

Returns self

Return type class

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

class diffprivlib.mechanisms.**LaplaceFolded**
The folded Laplace mechanism, where values outside a pre-described domain are folded around the domain until they fall within.

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

randomise (*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_bounds (*lower*, *upper*)
Sets the lower and upper bounds of the mechanism.

Must have lower <= upper.

Parameters

- **lower** (*float*) – The lower bound of the mechanism.
- **upper** (*float*) – The upper bound of the mechanism.

Returns self

Return type class

set_epsilon (*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_epsilon_delta (*epsilon*, *delta*)
Sets the value of epsilon and delta to be used by the mechanism.

epsilon and *delta* cannot both be zero.

Parameters

- **epsilon** (*float*) – The value of epsilon for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *epsilon >= 0*.
- **delta** (*float*) – The value of delta for achieving (ϵ, δ)-differential privacy with the mechanism. Must have *0 <= delta <= 1*.

  delta=0 gives strict (pure) differential privacy (ϵ-differential privacy). *delta > 0* gives relaxed (approximate) differential privacy.

Returns self

Return type class

Raises **ValueError** – If *epsilon* is negative, or if *delta* falls outside [0,1], or if *epsilon* and *delta* are both zero.

set_sensitivity (*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

2.7 Staircase mechanism

class diffprivlib.mechanisms.**Staircase**

The staircase mechanism in differential privacy.

The staircase mechanism is an optimisation of the classical Laplace Mechanism (*Laplace*), described as a "geometric mixture of uniform random variables". Paper link: https://arxiv.org/pdf/1212.1186.pdf

check_inputs(*value*)

Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)

Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

randomise(*value*)

Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

set_gamma(*gamma*)

Sets the tuning parameter γ for the mechanism.

Must satisfy 0 <= *gamma* <= 1. If not set, gamma defaults to minimise the expectation of the amplitude of noise, .. math:: gamma = frac{1}{1 + e^{epsilon / 2}}

Parameters **gamma** (*float*) – Value of the tuning parameter gamma for the mechanism.

Returns **self**

Return type class

Raises

- **TypeError** – If *gamma* is not a float.

- **ValueError** – If *gamma* is does not satisfy 0 <= *gamma* <= 1.

set_sensitivity(*sensitivity*)
Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

2.8 Uniform mechanism

class diffprivlib.mechanisms.**Uniform**
The Uniform mechanism in differential privacy.

This emerges as a special case of the *LaplaceBoundedNoise* mechanism when epsilon = 0. Paper link: https://arxiv.org/pdf/1810.00877.pdf

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*float*) – The value to be checked

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

get_bias(*value*)
Returns the bias of the mechanism at a given *value*.

Parameters **value** (*int or float*) – The value at which the bias of the mechanism is sought.

Returns **bias** – The bias of the mechanism at *value*.

Return type float or None

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **value** (*float*) – The value to be randomised.

Returns The randomised value.

Return type float

set_epsilon_delta(*epsilon*, *delta*)
Set privacy parameters ϵ and δ for the mechanism.

For the uniform mechanism, *epsilon* must be strictly zero and *delta* must satisfy 0 < *delta* <= 0.5.

Parameters

- **epsilon** (*float*) – For the uniform mechanism, *epsilon* must be strictly zero.
- **delta** (*float*) – For the uniform mechanism, *delta* must satisfy 0 < *delta* <= 0.5.

Returns self

Return type class

Raises

- **ValueError** – If *epsilon* is non-zero or if *delta* does not satisfy 0 < *delta* <= 0.5.
- **TypeError** – If *epsilon* or *delta* cannot be cast as floats.

set_sensitivity(*sensitivity*)

Sets the sensitivity of the mechanism.

Parameters **sensitivity** (*float*) – The sensitivity of the mechanism. Must satisfy *sensitivity* > 0.

Returns self

Return type class

2.9 Vector mechanism

class diffprivlib.mechanisms.**Vector**

The vector mechanism in differential privacy.

The vector mechanism is used when perturbing convex objective functions. Full paper: http://www.jmlr.org/papers/volume12/chaudhuri11a/chaudhuri11a.pdf

check_inputs(*value*)

Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*method*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

randomise(*value*)

Randomise *value* with the mechanism.

If *value* is a method of two outputs, they are taken as *f* and *fprime* (i.e., its gradient), and both are perturbed accordingly.

Parameters **value** (*method*) – The function to be randomised.

Returns The randomised method.

Return type method

set_alpha(*alpha*)

Set the regularisation parameter α for the mechanism.

alpha must be strictly positive. Default is 0.01.

Parameters **alpha** (*float*) – Regularisation parameter.

Returns self

Return type class

set_dimension(*vector_dim*)
Sets the dimension *vector_dim* of the domain of the mechanism.

This dimension relates to the size of the input vector of the function being considered by the mechanism. This corresponds to the size of the random vector produced by the mechanism.

Parameters **vector_dim** (*int*) – Function input dimension.

Returns self

Return type class

set_epsilon(*epsilon*)
Sets the value of epsilon to be used by the mechanism.

Parameters **epsilon** (*float*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns self

Return type class

set_sensitivity(*function_sensitivity*, *data_sensitivity=1*)
Sets the sensitivity of the function and data being processed by the mechanism.

- The sensitivity of the function relates to the max of its second derivative. Must be strictly positive.
- The sensitivity of the data relates to the max 2-norm of each row. Must be strictly positive.

Parameters

- **function_sensitivity** (*float*) – The function sensitivity of the mechanism.
- **data_sensitivity** (*float, default: 1.0*) – The data sensitivity of the mechanism.

Returns self

Return type class

2.10 Wishart mechanism

class diffprivlib.mechanisms.**Wishart**
The Wishart mechanism in differential privacy.

Used to achieve differential privacy on 2nd moment matrices.

Paper link: https://ieeexplore.ieee.org/abstract/document/7472095/

check_inputs(*value*)
Checks that all parameters of the mechanism have been initialised correctly, and that the mechanism is ready to be used.

Parameters **value** (*method*) – The value to be checked.

Returns

Return type True if the mechanism is ready to be used.

Raises **Exception** – If parameters have not been set correctly, or if *value* falls outside the domain of the mechanism.

randomise(*value*)
Randomise *value* with the mechanism.

Parameters **`value`** (*`numpy array`*) – The data to be randomised.

Returns The randomised array.

Return type numpy array

`set_epsilon`(*epsilon*)

Sets the value of epsilon to be used by the mechanism.

Parameters **`epsilon`** (*`float`*) – The value of epsilon for achieving ϵ-differential privacy with the mechanism. Must have *epsilon > 0*.

Returns **self**

Return type class

`set_sensitivity`(*sensitivity*)

Sets the l2-norm sensitivity of the data being processed by the mechanism.

Parameters **`sensitivity`** (*`float`*) – The maximum l2-norm of the data.

Returns **self**

Return type class

CHAPTER

THREE

DIFFPRIVLIB.MECHANISMS.TRANSFORMS

Transform wrappers for differential privacy mechanisms to extend their use to alternative data types.

Notes

The naming convention for new transforms is to describe the *pre-transform* action, i.e. the action performed on the data to be ingested by the mechanism. For transforms without a *pre-transform*, the *post-transform* action should be described.

3.1 Base class

class diffprivlib.mechanisms.transforms.**DPTransformer**(*parent*)

Base class for DP transformers. DP Transformers are simple wrappers for DP Mechanisms to allow mechanisms to be used with data types and structures outside their scope.

A *DPTransformer* must be initiated with a *DPMachine* (either another *DPTransformer*, or a *DPMechanism*). This allows many instances of *DPTransformer* to be chained together, but the chain must terminate with a *DPMechanism*.

copy()

Produces a copy of the class.

Returns **self** – Returns the copy.

Return type class

deepcopy()

Produces a deep copy of the class.

Returns **self** – Returns the deep copy.

Return type class

post_transform(*value*)

Performs no transformation on the output of the mechanism, and is returned as-is.

Parameters **value** (*float or string*) – Mechanism output to be transformed.

Returns Transformed output value.

Return type float or string

pre_transform(*value*)

Performs no transformation on the input data, and is ingested by the mechanism as-is.

Parameters **value** (*float or string*) – Input value to be transformed.

Returns Transformed input value

Return type float or string

randomise(*value*)

Randomise the given value using the *DPMachine*.

Parameters **value** (*float or string*) – Value to be randomised.

Returns Randomised value, same type as *value*.

Return type float or string

set_epsilon(*epsilon*)

Sets the value of epsilon to be used by the mechanism. For further details see *set_epsilon* of the mechanism.

Parameters **epsilon** (*float*) – Epsilon value for differential privacy.

Returns self

Return type class

set_epsilon_delta(*epsilon*, *delta*)

Sets the value of epsilon and delta to be used by the mechanism. For further details see *set_epsilon_delta* of the mechanism.

Parameters

- **epsilon** (*float*) – Epsilon value for differential privacy.
- **delta** (*float*) – Delta value for differential privacy.

Returns self

Return type class

3.2 Type casting transforms

class diffprivlib.mechanisms.transforms.**IntToString**(*parent*)

IntToString DP transformer, for using integer-valued data with string-valued mechanisms.

Useful when using integer-valued data with *Binary* or *Exponential*.

post_transform(*value*)

Transforms the output of the mechanism to be integer-valued.

Parameters **value** (*float or string*) – Mechanism output to be transformed.

Returns Transformed output value.

Return type int

pre_transform(*value*)

Transforms the input to be string-valued for ingestion by the mechanism.

Parameters **value** (*float or string*) – Input value to be transformed.

Returns Transformed input value

Return type string

randomise(*value*)

Randomise the given value using the *DPMachine*.

Parameters **value** (*float or string*) – Value to be randomised.

Returns Randomised value, same type as *value*.

Return type float or string

class `diffprivlib.mechanisms.transforms.`**`StringToInt`**(*parent*)

StringToInt DP transformer, for using string-valued data with integer-valued mechanisms.

Useful when using ordered, string-valued data with `Geometric`.

`post_transform`(*value*)

Transforms the output of the mechanism to be string-valued.

Parameters **value** (`float or string`) – Mechanism output to be transformed.

Returns Transformed output value.

Return type string

`pre_transform`(*value*)

Transforms the input to be integer-valued for ingestion by the mechanism.

Parameters **value** (`float or string`) – Input value to be transformed.

Returns Transformed input value

Return type int

`randomise`(*value*)

Randomise the given value using the `DPMachine`.

Parameters **value** (`float or string`) – Value to be randomised.

Returns Randomised value, same type as *value*.

Return type float or string

3.3 Other transforms

class `diffprivlib.mechanisms.transforms.`**`RoundedInteger`**(*parent*)

Rounded integer transform. Rounds the (float) output of the given mechanism to the nearest integer.

`post_transform`(*value*)

Transforms the (float) output of the mechanism to be a rounded integer.

Parameters **value** (`float`) – Mechanism output to be transformed.

Returns Transformed output value.

Return type int

`pre_transform`(*value*)

Performs no transformation on the input data, and is ingested by the mechanism as-is.

Parameters **value** (`float or string`) – Input value to be transformed.

Returns Transformed input value

Return type float or string

`randomise`(*value*)

Randomise the given value using the `DPMachine`.

Parameters **value** (`float or string`) – Value to be randomised.

Returns Randomised value, same type as *value*.

Return type float or string

CHAPTER

FOUR

DIFFPRIVLIB.TOOLS

Tools for data analysis with differential privacy.

4.1 Histogram functions

`diffprivlib.tools.`**`histogram`**(*sample*, *epsilon=1.0*, *bins=10*, *range=None*, *weights=None*, *density=None*, *accountant=None*, ***unused_args*)

Compute the differentially private histogram of a set of data.

The histogram is computed using `numpy.histogram`, and noise added using *`GeometricTruncated`* to satisfy differential privacy. If the *range* parameter is not specified correctly, a *`PrivacyLeakWarning`* is thrown. Users are referred to `numpy.histogram` for more usage notes.

Parameters

- **`sample`** (*array_like*) – Input data. The histogram is computed over the flattened array.
- **`epsilon`** (*float, default: 1.0*) – Privacy parameter ϵ to be applied.
- **`bins`** (*int or sequence of scalars or str, default: 10*) – If *bins* is an int, it defines the number of equal-width bins in the given range (10, by default). If *bins* is a sequence, it defines a monotonically increasing array of bin edges, including the rightmost edge, allowing for non-uniform bin widths.

  If *bins* is a string, it defines the method used to calculate the optimal bin width, as defined by *histogram_bin_edges*.
- **`range`** (*(float, float), optional*) – The lower and upper range of the bins. If not provided, range is simply `(a.min(), a.max())`. Values outside the range are ignored. The first element of the range must be less than or equal to the second. *range* affects the automatic bin computation as well. While bin width is computed to be optimal based on the actual data within *range*, the bin count will fill the entire range including portions containing no data.
- **`weights`** (*array_like, optional*) – An array of weights, of the same shape as *a*. Each value in *a* only contributes its associated weight towards the bin count (instead of 1). If *density* is True, the weights are normalized, so that the integral of the density over the range remains 1.
- **`density`** (*bool, optional*) – If `False`, the result will contain the number of samples in each bin. If `True`, the result is the value of the probability *density* function at the bin, normalized such that the *integral* over the range is 1. Note that the sum of the histogram values will not be equal to 1 unless bins of unity width are chosen; it is not a probability *mass* function.

- **accountant** (`BudgetAccountant`, *optional*) – Accountant to keep track of privacy budget.

Returns

- **hist** (*array*) – The values of the histogram. See *density* and *weights* for a description of the possible semantics.
- **bin_edges** (*array of dtype float*) – Return the bin edges `(length(hist)+1)`.

See also:

histogramdd, *histogram2d*

Notes

All but the last (righthand-most) bin is half-open. In other words, if *bins* is:

```
[1, 2, 3, 4]
```

then the first bin is `[1, 2)` (including 1, but excluding 2) and the second `[2, 3)`. The last bin, however, is `[3, 4]`, which *includes* 4.

`diffprivlib.tools.`**`histogramdd`**(*sample*, *epsilon=1.0*, *bins=10*, *range=None*, *weights=None*, *density=None*, *accountant=None*, ***unused_args*)

Compute the differentially private multidimensional histogram of some data.

The histogram is computed using `numpy.histogramdd`, and noise added using *GeometricTruncated* to satisfy differential privacy. If the *range* parameter is not specified correctly, a *PrivacyLeakWarning* is thrown. Users are referred to `numpy.histogramdd` for more usage notes.

Parameters

- **sample** (*(N, D) array, or (D, N) array_like*) – The data to be histogrammed.

  Note the unusual interpretation of sample when an array_like:

  – When an array, each row is a coordinate in a D-dimensional space - such as `histogramgramdd(np.array([p1, p2, p3]))`.

  – When an array_like, each element is the list of values for single coordinate - such as `histogramgramdd((X, Y, Z))`.

  The first form should be preferred.

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ to be applied.
- **bins** (*sequence or int, default: 10*) – The bin specification:

  – A sequence of arrays describing the monotonically increasing bin edges along each dimension.

  – The number of bins for each dimension (nx, ny, … =bins)

  – The number of bins for all dimensions (nx=ny=…=bins).

- **range** (*sequence, optional*) – A sequence of length D, each an optional (lower, upper) tuple giving the outer bin edges to be used if the edges are not given explicitly in *bins*. An entry of None in the sequence results in the minimum and maximum values being used for the corresponding dimension. The default, None, is equivalent to passing a tuple of D None values.

- **density** (*bool, optional*) – If False, the default, returns the number of samples in each bin. If True, returns the probability *density* function at the bin, `bin_count / sample_count / bin_volume`.
- **weights** (*(N,) array_like, optional*) – An array of values *w_i* weighing each sample *(x_i, y_i, z_i, ...)*. Weights are normalized to 1 if normed is True. If normed is False, the values of the returned histogram are equal to the sum of the weights belonging to the samples falling into each bin.
- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.

Returns

- **H** (*ndarray*) – The multidimensional histogram of sample x. See normed and weights for the different possible semantics.
- **edges** (*list*) – A list of D arrays describing the bin edges for each dimension.

See also:

histogram 1-D differentially private histogram

histogram2d 2-D differentially private histogram

`diffprivlib.tools.`**`histogram2d`**(*array_x, array_y, epsilon=1.0, bins=10, range=None, weights=None, density=None, accountant=None, **unused_args*)

Compute the differentially private bi-dimensional histogram of two data samples.

Parameters

- **array_x** (*array_like, shape (N,)*) – An array containing the x coordinates of the points to be histogrammed.
- **array_y** (*array_like, shape (N,)*) – An array containing the y coordinates of the points to be histogrammed.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ to be applied.
- **bins** (*int or array_like or [int, int] or [array, array], default: 10*) – The bin specification:
  - If int, the number of bins for the two dimensions (nx=ny=bins).
  - If array_like, the bin edges for the two dimensions (x_edges=y_edges=bins).
  - If [int, int], the number of bins in each dimension (nx, ny = bins).
  - If [array, array], the bin edges in each dimension (x_edges, y_edges = bins).
  - A combination [int, array] or [array, int], where int is the number of bins and array is the bin edges.
- **range** (*array_like, shape(2,2), optional*) – The leftmost and rightmost edges of the bins along each dimension (if not specified explicitly in the *bins* parameters): `[[xmin, xmax], [ymin, ymax]]`. All values outside of this range will be considered outliers and not tallied in the histogram.
- **density** (*bool, optional*) – If False, the default, returns the number of samples in each bin. If True, returns the probability *density* function at the bin, `bin_count / sample_count / bin_area`.

- **weights** (*array_like, shape(N,), optional*) – An array of values `w_i` weighing each sample `(x_i, y_i)`. Weights are normalized to 1 if *normed* is True. If *normed* is False, the values of the returned histogram are equal to the sum of the weights belonging to the samples falling into each bin.
- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.

Returns

- **H** (*ndarray, shape(nx, ny)*) – The bi-dimensional histogram of samples *x* and *y*. Values in *x* are histogrammed along the first dimension and values in *y* are histogrammed along the second dimension.
- **xedges** (*ndarray, shape(nx+1,)*) – The bin edges along the first dimension.
- **yedges** (*ndarray, shape(ny+1,)*) – The bin edges along the second dimension.

See also:

histogram 1D differentially private histogram

histogramdd Differentially private Multidimensional histogram

Notes

When *normed* is True, then the returned histogram is the sample density, defined such that the sum over bins of the product `bin_value * bin_area` is 1.

Please note that the histogram does not follow the Cartesian convention where *x* values are on the abscissa and *y* values on the ordinate axis. Rather, *x* is histogrammed along the first dimension of the array (vertical), and *y* along the second dimension of the array (horizontal). This ensures compatibility with *histogramdd*.

4.2 General Utilities

diffprivlib.tools.**count_nonzero**(*array*, *epsilon=1.0*, *accountant=None*, *axis=None*, *keepdims=False*)

Counts the number of non-zero values in the array `array` with differential privacy.

The word "non-zero" is in reference to the Python 2.x built-in method `__nonzero__()` (renamed `__bool__()` in Python 3.x) of Python objects that tests an object's "truthfulness". For example, any number is considered truthful if it is nonzero, whereas any string is considered truthful if it is not the empty string. Thus, this function (recursively) counts how many elements in `array` (and in sub-arrays thereof) have their `__nonzero__()` or `__bool__()` method evaluated to `True`.

Parameters

- **array** (*array_like*) – The array for which to count non-zeros.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.
- **axis** (*int or tuple, optional*) – Axis or tuple of axes along which to count non-zeros. Default is None, meaning that non-zeros will be counted along a flattened version of `array`.

- **keepdims** (*bool, optional*) – If this is set to True, the axes that are counted are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

Returns **count** – Differentially private number of non-zero values in the array along a given axis. Otherwise, the total number of non-zero values in the array is returned.

Return type int or array of int

`diffprivlib.tools.`**`mean`**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the differentially private arithmetic mean along the specified axis.

Returns the average of the array elements with differential privacy. The average is taken over the flattened array by default, otherwise over the specified axis. Noise is added using `Laplace` to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of `numpy.mean` for further details, as the behaviour of *mean* closely follows its Numpy variant.

Parameters

- **array** (*array_like*) – Array containing numbers whose mean is desired. If *array* is not an array, a conversion is attempted.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

  If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- **dtype** (*data-type, optional*) – Type to use in computing the mean. For integer inputs, the default is *float64*; for floating point inputs, it is the same as the input dtype.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *mean* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.
- **accountant** (`BudgetAccountant`, *optional*) – Accountant to keep track of privacy budget.

Returns **m** – Returns a new array containing the mean values.

Return type ndarray, see dtype parameter above

See also:

std, *var*, *nanmean*

`diffprivlib.tools.`**`nanmean`**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the differentially private arithmetic mean along the specified axis, ignoring NaNs.

Returns the average of the array elements with differential privacy. The average is taken over the flattened array by default, otherwise over the specified axis. Noise is added using `Laplace` to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of `numpy.mean` for further details, as the behaviour of *mean* closely follows its Numpy variant.

For all-NaN slices, NaN is returned and a *RuntimeWarning* is raised.

Parameters

- **array** (*array_like*) – Array containing numbers whose mean is desired. If *array* is not an array, a conversion is attempted.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the means are computed. The default is to compute the mean of the flattened array.

  If this is a tuple of ints, a mean is performed over multiple axes, instead of a single axis or all the axes as before.
- **dtype** (*data-type, optional*) – Type to use in computing the mean. For integer inputs, the default is *float64*; for floating point inputs, it is the same as the input dtype.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *mean* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.
- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.

Returns **m** – Returns a new array containing the mean values.

Return type ndarray, see dtype parameter above

See also:

std, *var*, *mean*

`diffprivlib.tools.`**`std`**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the standard deviation along the specified axis.

Returns the standard deviation of the array elements, a measure of the spread of a distribution, with differential privacy. The standard deviation is computed for the flattened array by default, otherwise over the specified axis. Noise is added using *LaplaceBoundedDomain* to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of `numpy.std` for further details, as the behaviour of *std* closely follows its Numpy variant.

Parameters

- **array** (*array_like*) – Calculate the standard deviation of these values.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the standard deviation is computed. The default is to compute the standard deviation of the flattened array.

  If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before.

- **dtype** (*dtype, optional*) – Type to use in computing the standard deviation. For arrays of integer type the default is float64, for arrays of float types it is the same as the array type.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *std* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.
- **accountant** (BudgetAccountant, *optional*) – Accountant to keep track of privacy budget.

Returns **standard_deviation** – Return a new array containing the standard deviation.

Return type ndarray, see dtype parameter above.

See also:

var, *mean*, *nanstd*

diffprivlib.tools.**nanstd**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the standard deviation along the specified axis, ignoring NaNs.

Returns the standard deviation of the array elements, a measure of the spread of a distribution, with differential privacy. The standard deviation is computed for the flattened array by default, otherwise over the specified axis. Noise is added using *LaplaceBoundedDomain* to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of numpy.std for further details, as the behaviour of *std* closely follows its Numpy variant.

For all-NaN slices, NaN is returned and a *RuntimeWarning* is raised.

Parameters

- **array** (*array_like*) – Calculate the standard deviation of these values.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the standard deviation is computed. The default is to compute the standard deviation of the flattened array.

  If this is a tuple of ints, a standard deviation is performed over multiple axes, instead of a single axis or all the axes as before.
- **dtype** (*dtype, optional*) – Type to use in computing the standard deviation. For arrays of integer type the default is float64, for arrays of float types it is the same as the array type.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *std* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.

- **accountant** (`BudgetAccountant, optional`) – Accountant to keep track of privacy budget.

Returns **standard_deviation** – Return a new array containing the standard deviation.

Return type ndarray, see dtype parameter above.

See also:

var, *mean*, *std*

`diffprivlib.tools.`**`sum`**(*array*, *epsilon=1.0*, *bounds=None*, *accountant=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, ***unused_args*)

Sum of array elements over a given axis with differential privacy.

Parameters

- **array** (*array_like*) – Elements to sum.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **accountant** (`BudgetAccountant, optional`) – Accountant to keep track of privacy budget.
- **axis** (*None or int or tuple of ints, optional*) – Axis or axes along which a sum is performed. The default, axis=None, will sum all of the elements of the input array. If axis is negative it counts from the last to the first axis.

  If axis is a tuple of ints, a sum is performed on all of the axes specified in the tuple instead of a single axis or all the axes as before.
- **dtype** (*dtype, optional*) – The type of the returned array and of the accumulator in which the elements are summed. The dtype of *array* is used by default unless *array* has an integer dtype of less precision than the default platform integer. In that case, if *array* is signed then the platform integer is used while if *array* is unsigned then an unsigned integer of the same precision as the platform integer is used.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *sum* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.

Returns **sum_along_axis** – An array with the same shape as *array*, with the specified axis removed. If *array* is a 0-d array, or if *axis* is None, a scalar is returned.

Return type ndarray

See also:

`ndarray.sum` Equivalent non-private method.

mean, *nansum*

`diffprivlib.tools.`**`nansum`**(*array*, *epsilon=1.0*, *bounds=None*, *accountant=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, ***unused_args*)

Sum of array elements over a given axis with differential privacy, ignoring NaNs.

Parameters

- **array** (*array_like*) – Elements to sum.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).
- **accountant** (BudgetAccountant*, optional*) – Accountant to keep track of privacy budget.
- **axis** (*None or int or tuple of ints, optional*) – Axis or axes along which a sum is performed. The default, axis=None, will sum all of the elements of the input array. If axis is negative it counts from the last to the first axis.

  If axis is a tuple of ints, a sum is performed on all of the axes specified in the tuple instead of a single axis or all the axes as before.
- **dtype** (*dtype, optional*) – The type of the returned array and of the accumulator in which the elements are summed. The dtype of *array* is used by default unless *array* has an integer dtype of less precision than the default platform integer. In that case, if *array* is signed then the platform integer is used while if *array* is unsigned then an unsigned integer of the same precision as the platform integer is used.
- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *sum* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.

Returns **sum_along_axis** – An array with the same shape as *array*, with the specified axis removed. If *array* is a 0-d array, or if *axis* is None, a scalar is returned. If an output array is specified, a reference to *out* is returned.

Return type ndarray

See also:

ndarray.sum Equivalent non-private method.

mean, *sum*

diffprivlib.tools.**var**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the differentially private variance along the specified axis.

Returns the variance of the array elements, a measure of the spread of a distribution, with differential privacy. The variance is computer for the flattened array by default, otherwise over the specified axis. Noise is added using *LaplaceBoundedDomain* to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of numpy.var for further details, as the behaviour of *var* closely follows its Numpy variant.

Parameters

- **array** (*array_like*) – Array containing numbers whose variance is desired. If *array* is not an array, a conversion is attempted.
- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).

- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the variance is computed. The default is to compute the variance of the flattened array.

  If this is a tuple of ints, a variance is performed over multiple axes, instead of a single axis or all the axes as before.

- **dtype** (*data-type, optional*) – Type to use in computing the variance. For arrays of integer type the default is *float32*; for arrays of float types it is the same as the array type.

- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

  If the default value is passed, then *keepdims* will not be passed through to the *var* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.

- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.

Returns **variance** – Returns a new array containing the variance.

Return type ndarray, see dtype parameter above

See also:

std, *mean*, *nanvar*

diffprivlib.tools.**nanvar**(*array*, *epsilon=1.0*, *bounds=None*, *axis=None*, *dtype=None*, *keepdims=<no value>*, *accountant=None*, ***unused_args*)

Compute the differentially private variance along the specified axis, ignoring NaNs.

Returns the variance of the array elements, a measure of the spread of a distribution, with differential privacy. The variance is computer for the flattened array by default, otherwise over the specified axis. Noise is added using *LaplaceBoundedDomain* to satisfy differential privacy, where sensitivity is calculated using *bounds*. Users are advised to consult the documentation of `numpy.var` for further details, as the behaviour of *var* closely follows its Numpy variant.

For all-NaN slices, NaN is returned and a *RuntimeWarning* is raised.

Parameters

- **array** (*array_like*) – Array containing numbers whose variance is desired. If *array* is not an array, a conversion is attempted.

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.

- **bounds** (*tuple, optional*) – Bounds of the values of the array, of the form (min, max).

- **axis** (*int or tuple of ints, optional*) – Axis or axes along which the variance is computed. The default is to compute the variance of the flattened array.

  If this is a tuple of ints, a variance is performed over multiple axes, instead of a single axis or all the axes as before.

- **dtype** (*data-type, optional*) – Type to use in computing the variance. For arrays of integer type the default is *float32*; for arrays of float types it is the same as the array type.

- **keepdims** (*bool, optional*) – If this is set to True, the axes which are reduced are left in the result as dimensions with size one. With this option, the result will broadcast correctly against the input array.

If the default value is passed, then *keepdims* will not be passed through to the *var* method of sub-classes of *ndarray*, however any non-default value will be. If the sub-class' method does not implement *keepdims* any exceptions will be raised.

- **`accountant`** (`BudgetAccountant`, *`optional`*) – Accountant to keep track of privacy budget.

Returns **variance** – If `out=None`, returns a new array containing the variance; otherwise, a reference to the output array is returned.

Return type ndarray, see dtype parameter above

See also:

`std`, *`mean`*, *`var`*

CHAPTER

FIVE

DIFFPRIVLIB.MODELS

Machine learning models with differential privacy

5.1 Classification models

5.1.1 Gaussian Naive Bayes

class diffprivlib.models.**GaussianNB**(*epsilon=1.0*, *bounds=None*, *priors=None*, *var_smoothing=1e-09*, *accountant=None*)

Gaussian Naive Bayes (GaussianNB) with differential privacy

Inherits the `sklearn.naive_bayes.GaussianNB` class from Scikit Learn and adds noise to satisfy differential privacy to the learned means and variances. Adapted from the work presented in [VSB13].

Parameters

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ for the model.
- **bounds** (*tuple, optional*) – Bounds of the data, provided as a tuple of the form (min, max). *min* and *max* can either be scalars, covering the min/max of the entire data, or vectors with one entry per feature. If not provided, the bounds are computed on the data when `.fit()` is first called, resulting in a *PrivacyLeakWarning*.
- **priors** (*array-like, shape (n_classes,)*) – Prior probabilities of the classes. If specified the priors are not adjusted according to the data.
- **var_smoothing** (*float, default: 1e-9*) – Portion of the largest variance of all features that is added to variances for calculation stability.
- **accountant** (*BudgetAccountant, optional*) – Accountant to keep track of privacy budget.

class_prior_

probability of each class.

Type array, shape (n_classes,)

class_count_

number of training samples observed in each class.

Type array, shape (n_classes,)

theta_

mean of each feature per class

Type array, shape (n_classes, n_features)

sigma_
variance of each feature per class

Type array, shape (n_classes, n_features)

epsilon_
absolute additive value to variances (unrelated to `epsilon` parameter for differential privacy)

Type float

References

fit (*X*, *y*, *sample_weight=None*)
Fit Gaussian Naive Bayes according to X, y

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (*array-like, shape (n_samples,)*) – Target values.
- **sample_weight** (*array-like, shape (n_samples,), optional (default=None)*) – Weights applied to individual samples (1. for unweighted).

  New in version 0.17: Gaussian Naive Bayes supports fitting with *sample_weight*.

Returns self

Return type object

get_params (*deep=True*)
Get parameters for this estimator.

Parameters deep (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns params – Parameter names mapped to their values.

Return type mapping of string to any

partial_fit (*X*, *y*, *classes=None*, *sample_weight=None*)
Incremental fit on a batch of samples.

This method is expected to be called several times consecutively on different chunks of a dataset so as to implement out-of-core or online learning.

This is especially useful when the whole dataset is too big to fit in memory at once.

This method has some performance and numerical stability overhead, hence it is better to call partial_fit on chunks of data that are as large as possible (as long as fitting in the memory budget) to hide the overhead.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training vectors, where n_samples is the number of samples and n_features is the number of features.
- **y** (*array-like, shape (n_samples,)*) – Target values.
- **classes** (*array-like, shape (n_classes,), optional (default=None)*) – List of all the classes that can possibly appear in the y vector.

  Must be provided at the first call to partial_fit, can be omitted in subsequent calls.

- **sample_weight** (*array-like, shape (n_samples,), optional (default=None)*) – Weights applied to individual samples (1. for unweighted).

  New in version 0.17.

Returns self

Return type object

predict(*X*)

Perform classification on an array of test vectors X.

Parameters X (*array-like of shape (n_samples, n_features)*) –

Returns C – Predicted target values for X

Return type ndarray of shape (n_samples,)

predict_log_proba(*X*)

Return log-probability estimates for the test vector X.

Parameters X (*array-like of shape (n_samples, n_features)*) –

Returns C – Returns the log-probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute classes_.

Return type array-like of shape (n_samples, n_classes)

predict_proba(*X*)

Return probability estimates for the test vector X.

Parameters X (*array-like of shape (n_samples, n_features)*) –

Returns C – Returns the probability of the samples for each class in the model. The columns correspond to the classes in sorted order, as they appear in the attribute classes_.

Return type array-like of shape (n_samples, n_classes)

score(*X*, *y*, *sample_weight=None*)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for X.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns score – Mean accuracy of self.predict(X) wrt. y.

Return type float

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters **params (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

5.1.2 Logistic Regression

class diffprivlib.models.**LogisticRegression**(*epsilon=1.0*, *data_norm=None*, *tol=0.0001*, *C=1.0*, *fit_intercept=True*, *max_iter=100*, *verbose=0*, *warm_start=False*, *n_jobs=None*, *accountant=None*, ***unused_args*)

Logistic Regression (aka logit, MaxEnt) classifier with differential privacy.

This class implements regularised logistic regression using Scipy's L-BFGS-B algorithm. ϵ-Differential privacy is achieved relative to the maximum norm of the data, as determined by *data_norm*, by the `Vector` mechanism, which adds a Laplace-distributed random vector to the objective. Adapted from the work presented in [CMS11].

This class is a child of `sklearn.linear_model.LogisticRegression`, with amendments to allow for the implementation of differential privacy. Some parameters of *Scikit Learn*'s model have therefore had to be fixed, including:

- The only permitted solver is 'lbfgs'. Specifying the `solver` option will result in a warning.
- Consequently, the only permitted penalty is 'l2'. Specifying the `penalty` option will result in a warning.
- In the multiclass case, only the one-vs-rest (OvR) scheme is permitted. Specifying the `multi_class` option will result in a warning.

Parameters

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **data_norm** (*float, optional*) – The max l2 norm of any row of the data. This defines the spread of data that will be protected by differential privacy.

  If not specified, the max norm is taken from the data when `.fit()` is first called, but will result in a `PrivacyLeakWarning`, as it reveals information about the data. To preserve differential privacy fully, *data_norm* should be selected independently of the data, i.e. with domain knowledge.
- **tol** (*float, default: 1e-4*) – Tolerance for stopping criteria.
- **C** (*float, default: 1.0*) – Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.
- **fit_intercept** (*bool, default: True*) – Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.
- **max_iter** (*int, default: 100*) – Maximum number of iterations taken for the solver to converge. For smaller *epsilon* (more noise), *max_iter* may need to be increased.
- **verbose** (*int, default: 0*) – Set to any positive number for verbosity.
- **warm_start** (*bool, default: False*) – When set to `True`, reuse the solution of the previous call to fit as initialization, otherwise, just erase the previous solution.
- **n_jobs** (*int, optional*) – Number of CPU cores used when parallelising over classes. `None` means 1 unless in a context. `-1` means using all processors.
- **accountant** (`BudgetAccountant`, *optional*) – Accountant to keep track of privacy budget.

classes_
A list of class labels known to the classifier.

Type array, shape (n_classes,)

coef_
Coefficient of the features in the decision function.

*coef_* is of shape (1, n_features) when the given problem is binary.

Type array, shape (1, n_features) or (n_classes, n_features)

intercept_
Intercept (a.k.a. bias) added to the decision function.

If *fit_intercept* is set to False, the intercept is set to zero. *intercept_* is of shape (1,) when the given problem is binary.

Type array, shape (1,) or (n_classes,)

n_iter_
Actual number of iterations for all classes. If binary, it returns only 1 element.

Type array, shape (n_classes,) or (1,)

Examples

```
>>> from sklearn.datasets import load_iris
>>> from diffprivlib.models import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(data_norm=12, epsilon=2).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[7.35362932e-01, 2.16667422e-14, 2.64637068e-01],
       [9.08384378e-01, 3.47767052e-13, 9.16156215e-02]])
>>> clf.score(X, y)
0.5266666666666666
```

See also:

sklearn.linear_model.LogisticRegression The implementation of logistic regression in scikit-learn, upon which this implementation is built.

Vector The mechanism used by the model to achieve differential privacy.

References

decision_function(*X*)
Predict confidence scores for samples.

The confidence score for a sample is the signed distance of that sample to the hyperplane.

Parameters X (*array_like or sparse matrix, shape (n_samples, n_features)*) – Samples.

Returns Confidence scores per (sample, class) combination. In the binary case, confidence score for self.classes_[1] where >0 means this class would be predicted.

Return type array, shape=(n_samples,) if n_classes == 2 else (n_samples, n_classes)

densify()
Convert coefficient matrix to dense array format.

Converts the `coef_` member (back) to a numpy.ndarray. This is the default format of `coef_` and is required for fitting, so calling this method is only required on models that have previously been sparsified; otherwise, it is a no-op.

Returns Fitted estimator.

Return type self

fit(*X*, *y*, *sample_weight=None*)
Fit the model according to the given training data.

Parameters

- **X** (*{array-like, sparse matrix}, shape (n_samples, n_features)*) – Training vector, where n_samples is the number of samples and n_features is the number of features.
- **y** (*array-like, shape (n_samples,)*) – Target vector relative to X.
- **sample_weight** (*ignored*) – Ignored by diffprivlib. Present for consistency with sklearn API.

Returns **self**

Return type class

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict(*X*)
Predict class labels for samples in X.

Parameters **X** (*array_like or sparse matrix, shape (n_samples, n_features)*) – Samples.

Returns **C** – Predicted class label per sample.

Return type array, shape [n_samples]

predict_log_proba(*X*)
Predict logarithm of probability estimates.

The returned estimates for all classes are ordered by the label of classes.

Parameters **X** (*array-like of shape (n_samples, n_features)*) – Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns **T** – Returns the log-probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Return type array-like of shape (n_samples, n_classes)

predict_proba(*X*)
Probability estimates.

The returned estimates for all classes are ordered by the label of classes.

For a multi_class problem, if multi_class is set to be "multinomial" the softmax function is used to find the predicted probability of each class. Else use a one-vs-rest approach, i.e calculate the probability of each class assuming it to be positive using the logistic function. and normalize these values across all the classes.

Parameters **X** (*array-like of shape (n_samples, n_features)*) – Vector to be scored, where *n_samples* is the number of samples and *n_features* is the number of features.

Returns **T** – Returns the probability of the sample for each class in the model, where classes are ordered as they are in `self.classes_`.

Return type array-like of shape (n_samples, n_classes)

score (*X*, *y*, *sample_weight=None*)

Return the mean accuracy on the given test data and labels.

In multi-label classification, this is the subset accuracy which is a harsh metric since you require for each sample that each label set be correctly predicted.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True labels for X.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns **score** – Mean accuracy of self.predict(X) wrt. y.

Return type float

set_params (***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

sparsify ()

Convert coefficient matrix to sparse format.

Converts the `coef_` member to a scipy.sparse matrix, which for L1-regularized models can be much more memory- and storage-efficient than the usual numpy.ndarray representation.

The `intercept_` member is not converted.

Returns Fitted estimator.

Return type self

Notes

For non-sparse models, i.e. when there are not many zeros in `coef_`, this may actually *increase* memory usage, so use this method with care. A rule of thumb is that the number of zero elements, which can be computed with `(coef_ == 0).sum()`, must be more than 50% for this to provide significant benefits.

After calling this method, further fitting with the partial_fit method (if any) will not work until you call densify.

5.2 Regression models

5.2.1 Linear Regression

class diffprivlib.models.**LinearRegression**(*epsilon=1.0*, *data_norm=None*, *bounds_X=None*, *bounds_y=None*, *fit_intercept=True*, *copy_X=True*, *accountant=None*, ***unused_args*)

Ordinary least squares Linear Regression with differential privacy.

LinearRegression fits a linear model with coefficients w = (w1, . . . , wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Differential privacy is guaranteed with respect to the training sample.

Differential privacy is achieved by adding noise to the second moment matrix using the `Wishart` mechanism. This method is demonstrated in [She15], but our implementation takes inspiration from the use of the Wishart distribution in [IS16] to achieve a strict differential privacy guarantee.

Parameters

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **data_norm** (*float, optional*) – The max l2 norm of any row of the concatenated dataset A = [X; y]. This defines the spread of data that will be protected by differential privacy.

  If not specified, the max norm is taken from the data when `.fit()` is first called, but will result in a `PrivacyLeakWarning`, as it reveals information about the data. To preserve differential privacy fully, *data_norm* should be selected independently of the data, i.e. with domain knowledge.
- **bounds_X** (*tuple, optional*) – Bounds of the data, provided as a tuple of the form (min, max). *min* and *max* can either be scalars, covering the min/max of the entire data, or vectors with one entry per feature. If not provided, the bounds are computed on the data when `.fit()` is first called, resulting in a `PrivacyLeakWarning`.
- **bounds_y** (*tuple*) – Same as *bounds_X*, but for the training label set *y*.
- **fit_intercept** (*bool, default: True*) – Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered).
- **copy_X** (*bool, default: True*) – If True, X will be copied; else, it may be overwritten.
- **accountant** (`BudgetAccountant`*, optional*) – Accountant to keep track of privacy budget.

coef_
Estimated coefficients for the linear regression problem. If multiple targets are passed during the fit (y 2D), this is a 2D array of shape (n_targets, n_features), while if only one target is passed, this is a 1D array of length n_features.

Type array of shape (n_features,) or (n_targets, n_features)

rank_
Rank of matrix *X*.

Type int

singular_
Singular values of *X*.

Type array of shape (min(X, y),)

intercept_
Independent term in the linear model. Set to 0.0 if *fit_intercept = False*.

Type float or array of shape of (n_targets,)

References

fit (*X*, *y*, *sample_weight=None*)
Fit linear model.

Parameters

- **X** (*array-like or sparse matrix, shape (n_samples, n_features)*) – Training data
- **y** (*array_like, shape (n_samples, n_targets)*) – Target values. Will be cast to X's dtype if necessary
- **sample_weight** (*ignored*) – Ignored by diffprivlib. Present for consistency with sklearn API.

Returns self

Return type returns an instance of self.

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*)
Predict using the linear model.

Parameters **X** (*array_like or sparse matrix, shape (n_samples, n_features)*) – Samples.

Returns **C** – Returns predicted values.

Return type array, shape (n_samples,)

score (*X*, *y*, *sample_weight=None*)

Return the coefficient of determination R^2 of the prediction.

The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

Parameters

- **X** (*array-like of shape (n_samples, n_features)*) – Test samples. For some estimators this may be a precomputed kernel matrix or a list of generic objects instead, shape = (n_samples, n_samples_fitted), where n_samples_fitted is the number of samples used in the fitting for the estimator.
- **y** (*array-like of shape (n_samples,) or (n_samples, n_outputs)*) – True values for X.
- **sample_weight** (*array-like of shape (n_samples,), default=None*) – Sample weights.

Returns **score** – R^2 of self.predict(X) wrt. y.

Return type float

Notes

The R2 score used when calling `score` on a regressor will use `multioutput='uniform_average'` from version 0.23 to keep consistent with `r2_score()`. This will influence the `score` method of all the multioutput regressors (except for `MultiOutputRegressor`). To specify the default value manually and avoid the warning, please either call `r2_score()` directly or make a custom scorer with `make_scorer()` (the built-in scorer `'r2'` uses `multioutput='uniform_average'`).

set_params (***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

5.3 Clustering models

5.3.1 K-Means

class diffprivlib.models.**KMeans** (*epsilon=1.0*, *bounds=None*, *n_clusters=8*, *accountant=None*, ***unused_args*)

K-Means clustering with differential privacy.

Implements the DPLloyd approach presented in [SCL16], leveraging the `sklearn.cluster.KMeans` class for full integration with Scikit Learn.

Parameters

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ.
- **bounds** (*tuple, optional*) – Bounds of the data, provided as a tuple of the form (min, max). *min* and *max* can either be scalars, covering the min/max of the entire data, or vectors with one entry per feature. If not provided, the bounds are computed on the data when `.fit()` is first called, resulting in a *PrivacyLeakWarning*.
- **n_clusters** (*int, default: 8*) – The number of clusters to form as well as the number of centroids to generate.
- **accountant** (BudgetAccountant, *optional*) – Accountant to keep track of privacy budget.(

cluster_centers_
Coordinates of cluster centers. If the algorithm stops before fully converging, these will not be consistent with `labels_`.

Type array, [n_clusters, n_features]

labels_
Labels of each point

inertia_
Sum of squared distances of samples to their closest cluster center.

Type float

n_iter_
Number of iterations run.

Type int

References

fit (*X*, *y=None*, *sample_weight=None*)
Computes k-means clustering with differential privacy.

Parameters

- **X** (*array-like, shape=(n_samples, n_features)*) – Training instances to cluster.
- **y** (*Ignored*) – not used, present here for API consistency by convention.
- **sample_weight** (*ignored*) – Ignored by diffprivlib. Present for consistency with sklearn API.

Returns self

Return type class

fit_predict (*X*, *y=None*, *sample_weight=None*)
Compute cluster centers and predict cluster index for each sample.

Convenience method; equivalent to calling fit(X) followed by predict(X).

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – New data to transform.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

- **sample_weight** (*array-like, shape (n_samples,), optional*) – The weights for each observation in X. If None, all observations are assigned equal weight (default: None).

Returns **labels** – Index of the cluster each sample belongs to.

Return type array, shape [n_samples,]

fit_transform (*X*, *y=None*, *sample_weight=None*)
Compute clustering and transform X to cluster-distance space.

Equivalent to fit(X).transform(X), but more efficiently implemented.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – New data to transform.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.
- **sample_weight** (*array-like, shape (n_samples,), optional*) – The weights for each observation in X. If None, all observations are assigned equal weight (default: None).

Returns **X_new** – X transformed in the new space.

Return type array, shape [n_samples, k]

get_params (*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

predict (*X*, *sample_weight=None*)
Predict the closest cluster each sample in X belongs to.

In the vector quantization literature, *cluster_centers_* is called the code book and each value returned by *predict* is the index of the closest code in the code book.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – New data to predict.
- **sample_weight** (*array-like, shape (n_samples,), optional*) – The weights for each observation in X. If None, all observations are assigned equal weight (default: None).

Returns **labels** – Index of the cluster each sample belongs to.

Return type array, shape [n_samples,]

score (*X*, *y=None*, *sample_weight=None*)
Opposite of the value of X on the K-means objective.

Parameters

- **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – New data.
- **y** (*Ignored*) – Not used, present here for API consistency by convention.

- **sample_weight** (*array-like, shape (n_samples,), optional*) – The weights for each observation in X. If None, all observations are assigned equal weight (default: None).

Returns **score** – Opposite of the value of X on the K-means objective.

Return type float

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form `<component>__<parameter>` so that it's possible to update each component of a nested object.

Parameters ****params** (*dict*) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*)

Transform X to a cluster-distance space.

In the new space, each dimension is the distance to the cluster centers. Note that even if X is sparse, the array returned by *transform* will typically be dense.

Parameters **X** (*{array-like, sparse matrix} of shape (n_samples, n_features)*) – New data to transform.

Returns **X_new** – X transformed in the new space.

Return type array, shape [n_samples, k]

5.4 Dimensionality reduction models

5.4.1 PCA

class diffprivlib.models.**PCA**(*n_components=None, centered=False, epsilon=1.0, data_norm=None, bounds=None, copy=True, whiten=False, random_state=None, accountant=None, **unused_args*)

Principal component analysis (PCA) with differential privacy.

This class is a child of `sklearn.decomposition.PCA`, with amendments to allow for the implementation of differential privacy as given in [IS16b]. Some parameters of *Scikit Learn*'s model have therefore had to be fixed, including:

- The only permitted *svd_solver* is 'full'. Specifying the `svd_solver` option will result in a warning;
- The parameters `tol` and `iterated_power` are not applicable (as a consequence of fixing `svd_solver = 'full'`).

Parameters

- **n_components** (*int, float, None or str*) – Number of components to keep. If n_components is not set all components are kept:

```
n_components == min(n_samples, n_features)
```

If `n_components == 'mle'`, Minka's MLE is used to guess the dimension.

If `0 < n_components < 1`, select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by n_components.

Hence, the None case results in:

```
n_components == min(n_samples, n_features) - 1
```

- **centered** (*bool, default: False*) – If False, the data will be centered before calculating the principal components. This will be calculated with differential privacy, consuming privacy budget from epsilon.

  If True, the data is assumed to have been centered previously (e.g. using *StandardScaler*), and therefore will not require the consumption of privacy budget to calculate the mean.

- **epsilon** (*float, default: 1.0*) – Privacy parameter ϵ. If `centered=False`, half of epsilon is used to calculate the differentially private mean to center the data prior to the calculation of principal components.

- **data_norm** (*float, optional*) – The max l2 norm of any row of the data. This defines the spread of data that will be protected by differential privacy.

  If not specified, the max norm is taken from the data when `.fit()` is first called, but will result in a *PrivacyLeakWarning*, as it reveals information about the data. To preserve differential privacy fully, *data_norm* should be selected independently of the data, i.e. with domain knowledge.

- **bounds** (*tuple, optional*) – Bounds of the data, provided as a tuple of the form (min, max). *min* and *max* can either be scalars, covering the min/max of the entire data, or vectors with one entry per feature. If not provided, the bounds are computed on the data when `.fit()` is first called, resulting in a *PrivacyLeakWarning*.

- **copy** (*bool, default: True*) – If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

- **whiten** (*bool, default: False*) – When True (False by default) the *components_* vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

  Whitening will remove some information from the transformed signal (the relative variance scales of the components) but can sometime improve the predictive accuracy of the downstream estimators by making their data respect some hard-wired assumptions.

- **random_state** (*int or RandomState instance, optional*) – If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator.

- **accountant** (BudgetAccountant, *optional*) – Accountant to keep track of privacy budget.

components_

Principal axes in feature space, representing the directions of maximum variance in the data. The components are sorted by `explained_variance_`.

Type array, shape (n_components, n_features)

explained_variance_

The amount of variance explained by each of the selected components.

Equal to n_components largest eigenvalues of the covariance matrix of X.

Type array, shape (n_components,)

explained_variance_ratio_
Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

Type array, shape (n_components,)

singular_values_
The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

Type array, shape (n_components,)

mean_
Per-feature empirical mean, estimated from the training set.

Equal to *X.mean(axis=0)*.

Type array, shape (n_features,)

n_components_
The estimated number of components. When n_components is set to 'mle' or a number between 0 and 1 (with svd_solver == 'full') this number is estimated from input data. Otherwise it equals the parameter n_components, or the lesser value of n_features and n_samples if n_components is None.

Type int

n_features_
Number of features in the training data.

Type int

n_samples_
Number of samples in the training data.

Type int

noise_variance_
The estimated noise covariance following the Probabilistic PCA model from Tipping and Bishop 1999. See "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf. It is required to compute the estimated data covariance and score samples.

Equal to the average of (min(n_features, n_samples) - n_components) smallest eigenvalues of the covariance matrix of X.

Type float

See also:

sklearn.decomposition.PCA Scikit-learn implementation Principal Component Analysis.

References

fit(*X*, *y=None*)
Fit the model with X.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **y** (*None*) – Ignored variable.

Returns **self** – Returns the instance itself.

Return type object

fit_transform(*X*, *y=None*)
Fit the model with X and apply the dimensionality reduction on X.

Parameters

- **X** (*array-like, shape (n_samples, n_features)*) – Training data, where n_samples is the number of samples and n_features is the number of features.
- **y** (*None*) – Ignored variable.

Returns **X_new** – Transformed values.

Return type array-like, shape (n_samples, n_components)

Notes

This method returns a Fortran-ordered array. To convert it to a C-ordered array, use 'np.ascontiguousarray'.

get_covariance()
Compute data covariance with the generative model.

`cov = components_.T * S**2 * components_ + sigma2 * eye(n_features)` where S**2 contains the explained variances, and sigma2 contains the noise variances.

Returns **cov** – Estimated covariance of data.

Return type array, shape=(n_features, n_features)

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

get_precision()
Compute data precision matrix with the generative model.

Equals the inverse of the covariance but computed with the matrix inversion lemma for efficiency.

Returns **precision** – Estimated precision of data.

Return type array, shape=(n_features, n_features)

inverse_transform(*X*)

Transform data back to its original space.

In other words, return an input X_original whose transform would be X.

Parameters X (*array-like, shape (n_samples, n_components)*) – New data, where n_samples is the number of samples and n_components is the number of components.

Returns

Return type X_original array-like, shape (n_samples, n_features)

Notes

If whitening is enabled, inverse_transform will compute the exact inverse operation, which includes reversing whitening.

score(*X*, *y=None*)

Return the average log-likelihood of all samples.

See. "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf

Parameters

- **X** (*array, shape(n_samples, n_features)*) – The data.
- **y** (*None*) – Ignored variable.

Returns ll – Average log-likelihood of the samples under the current model.

Return type float

score_samples(*X*)

Return the log-likelihood of each sample.

See. "Pattern Recognition and Machine Learning" by C. Bishop, 12.2.1 p. 574 or http://www.miketipping.com/papers/met-mppca.pdf

Parameters X (*array, shape(n_samples, n_features)*) – The data.

Returns ll – Log-likelihood of each sample under the current model.

Return type array, shape (n_samples,)

set_params(***params*)

Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters **params (*dict*) – Estimator parameters.

Returns self – Estimator instance.

Return type object

transform(*X*)

Apply dimensionality reduction to X.

X is projected on the first principal components previously extracted from a training set.

Parameters X (*array-like, shape (n_samples, n_features)*) – New data, where n_samples is the number of samples and n_features is the number of features.

Returns X_new

Return type array-like, shape (n_samples, n_components)

Examples

```
>>> import numpy as np
>>> from sklearn.decomposition import IncrementalPCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> ipca = IncrementalPCA(n_components=2, batch_size=3)
>>> ipca.fit(X)
IncrementalPCA(batch_size=3, n_components=2)
>>> ipca.transform(X)
```

5.5 Preprocessing

5.5.1 Standard Scaler

class diffprivlib.models.**StandardScaler**(*epsilon=1.0*, *bounds=None*, *copy=True*, *with_mean=True*, *with_std=True*, *accountant=None*)

Standardize features by removing the mean and scaling to unit variance, calculated with differential privacy guarantees. Differential privacy is guaranteed on the learned scaler with respect to the training sample; the transformed output will certainly not satisfy differential privacy.

The standard score of a sample *x* is calculated as:

z = (x - u) / s

where *u* is the (differentially private) mean of the training samples or zero if *with_mean=False*, and *s* is the (differentially private) standard deviation of the training samples or one if *with_std=False*.

Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using the *transform* method.

For further information, users are referred to `sklearn.preprocessing.StandardScaler`.

Parameters

- **epsilon** (*float, default: 1.0*) – The privacy budget to be allocated to learning the mean and variance of the training sample. If *with_std=True*, the privacy budget is split evenly between mean and variance (the mean must be calculated even when *with_mean=False*, as it is used in the calculation of the variance.
- **bounds** (*tuple, optional*) – Bounds of the data, provided as a tuple of the form (min, max). *min* and *max* can either be scalars, covering the min/max of the entire data, or vectors with one entry per feature. If not provided, the bounds are computed on the data when `.fit()` is first called, resulting in a `PrivacyLeakWarning`.
- **copy** (*boolean, default: True*) – If False, try to avoid a copy and do inplace scaling instead. This is not guaranteed to always work inplace; e.g. if the data is not a NumPy array, a copy may still be returned.
- **with_mean** (*boolean, True by default*) – If True, center the data before scaling.

- **with_std** (*boolean, True by default*) – If True, scale the data to unit variance (or equivalently, unit standard deviation).
- **accountant** (BudgetAccountant, *optional*) – Accountant to keep track of privacy budget.

scale_
Per feature relative scaling of the data. This is calculated using *np.sqrt(var_)*. Equal to `None` when `with_std=False`.

Type ndarray or None, shape (n_features,)

mean_
The mean value for each feature in the training set. Equal to `None` when `with_mean=False`.

Type ndarray or None, shape (n_features,)

var_
The variance for each feature in the training set. Used to compute *scale_*. Equal to `None` when `with_std=False`.

Type ndarray or None, shape (n_features,)

n_samples_seen_
The number of samples processed by the estimator for each feature. If there are not missing samples, the `n_samples_seen` will be an integer, otherwise it will be an array. Will be reset on new calls to fit, but increments across `partial_fit` calls.

Type int or array, shape (n_features,)

See also:

sklearn.preprocessing.StandardScaler Vanilla scikit-learn version, without differential privacy.

PCA Further removes the linear correlation across features with 'whiten=True'.

Notes

NaNs are treated as missing values: disregarded in fit, and maintained in transform.

fit (*X*, *y=None*)
Compute the mean and std to be used for later scaling.

Parameters

- **X** (*{array-like, sparse matrix}, shape [n_samples, n_features]*) – The data used to compute the mean and standard deviation used for later scaling along the features axis.
- **y** – Ignored

fit_transform (*X*, *y=None*, ***fit_params*)
Fit to data, then transform it.

Fits transformer to X and y with optional parameters fit_params and returns a transformed version of X.

Parameters

- **X** (*numpy array of shape [n_samples, n_features]*) – Training set.
- **y** (*numpy array of shape [n_samples]*) – Target values.
- ****fit_params** (*dict*) – Additional fit parameters.

Returns **X_new** – Transformed array.

Return type numpy array of shape [n_samples, n_features_new]

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (`bool, default=True`) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

inverse_transform(*X*, *copy=None*)
Scale back the data to the original representation

Parameters

- **X** (`array-like, shape [n_samples, n_features]`) – The data used to scale along the features axis.
- **copy** (`bool, optional (default: None)`) – Copy the input X or not.

Returns **X_tr** – Transformed array.

Return type array-like, shape [n_samples, n_features]

partial_fit(*X*, *y=None*)
Online computation of mean and std with differential privacy on X for later scaling. All of X is processed as a single batch. This is intended for cases when *fit* is not feasible due to very large number of *n_samples* or because X is read from a continuous stream.

The algorithm for incremental mean and std is given in Equation 1.5a,b in Chan, Tony F., Gene H. Golub, and Randall J. LeVeque. "Algorithms for computing the sample variance: Analysis and recommendations." The American Statistician 37.3 (1983): 242-247:

Parameters

- **X** (`{array-like}, shape [n_samples, n_features]`) – The data used to compute the mean and standard deviation used for later scaling along the features axis.
- **y** – Ignored

set_params(***params*)
Set the parameters of this estimator.

The method works on simple estimators as well as on nested objects (such as pipelines). The latter have parameters of the form <component>__<parameter> so that it's possible to update each component of a nested object.

Parameters ****params** (`dict`) – Estimator parameters.

Returns **self** – Estimator instance.

Return type object

transform(*X*, *copy=None*)
Perform standardization by centering and scaling

Parameters

- **X** (`array-like, shape [n_samples, n_features]`) – The data used to scale along the features axis.
- **copy** (`bool, optional (default: None)`) – Copy the input X or not.

CHAPTER SIX

UTILITIES AND GENERAL FUNCTIONS

Basic functions and other utilities for the differential privacy library

6.1 Exceptions and warnings

exception `diffprivlib.utils.`**`PrivacyLeakWarning`**

Custom warning to capture privacy leaks resulting from incorrect parameter setting.

For example, this warning may occur when the user:

- fails to specify the bounds or range of data to a model where required (e.g., *bounds=None* to `GaussianNB`).
- inputs data to a model that falls outside the bounds or range originally specified.

exception `diffprivlib.utils.`**`DiffprivlibCompatibilityWarning`**

Custom warning to capture inherited class arguments that are not compatible with diffprivlib.

The purpose of the warning is to alert the user of the incompatibility, but to continue execution having fixed the incompatibility at runtime.

For example, this warning may occur when the user:

- passes a parameter value that is not compatible with diffprivlib (e.g., *solver='liblinear'* to `LogisticRegression`)
- specifies a non-default value for a parameter that is ignored by diffprivlib (e.g., *intercept_scaling=0.5* to `LogisticRegression`.

exception `diffprivlib.utils.`**`BudgetError`**

Custom exception to capture the privacy budget being exceeded, typically controlled by a `BudgetAccountant`.

For example, this exception may be raised when the user:

- Attempts to execute a query which would exceed the privacy budget of the accountant.
- Attempts to change the slack of the accountant in such a way that the existing budget spends would exceed the accountant's budget.

6.2 General classes

class diffprivlib.utils.**Budget**(*epsilon*, *delta*)
Custom tuple subclass for privacy budgets of the form (epsilon, delta).

The `Budget` class allows for correct comparison/ordering of privacy budget, ensuring that both epsilon and delta satisfy the comparison (tuples are compared lexicographically). Additionally, tuples are represented with added verbosity, labelling epsilon and delta appropriately.

Examples

```
>>> from diffprivlib.utils import Budget
>>> Budget(1, 0.5)
(epsilon=1, delta=0.5)
>>> Budget(2, 0) >= Budget(1, 0.5)
False
>>> (2, 0) >= (1, 0.5) # Tuples are compared with lexicographic ordering
True
```

6.3 General functions

diffprivlib.utils.**copy_docstring**(*source*)
Decorator function to copy a docstring from a *source* function to a *target* function.

The docstring is only copied if a docstring is present in *source*, and if none is present in *target*. Takes inspiration from similar in *matplotlib*.

Parameters **source** (*method*) – Source function from which to copy the docstring. If `source.__doc__` is empty, do nothing.

Returns **target** – Target function with new docstring.

Return type method

diffprivlib.utils.**global_seed**(*seed*)
Sets the seed for all random number generators, to guarantee reproducibility in experiments.

Parameters **seed** (*int*) – The seed value for the random number generators.

Returns

Return type None

diffprivlib.utils.**warn_unused_args**(*args*)
Warn the user about supplying unused *args* to a diffprivlib model.

Arguments can be supplied as a string, a list of strings, or a dictionary as supplied to kwargs.

Parameters **args** (*str or list or dict*) – Arguments for which warnings should be thrown.

Returns

Return type None

CHAPTER

SEVEN

VALIDATION FUNCTIONS

Validation functions for the differential privacy library

7.1 General functions

`diffprivlib.validation.`**`check_epsilon_delta`**(*epsilon*, *delta*, *allow_zero=False*)

Checks that epsilon and delta are valid values for differential privacy. Throws an error if checks fail, otherwise returns nothing.

As well as the requirements of epsilon and delta separately, both cannot be simultaneously zero, unless `allow_zero` is set to `True`.

Parameters

- **epsilon** (*float*) – Epsilon parameter for differential privacy. Must be non-negative.
- **delta** (*float*) – Delta parameter for differential privacy. Must be on the unit interval, [0, 1].
- **allow_zero** (*bool, default: False*) – Allow epsilon and delta both be zero.

`diffprivlib.validation.`**`check_bounds`**(*bounds*, *shape=0*, *min_separation=0.0*, *dtype=<class 'float'>*)

Input validation for the `bounds` parameter.

Checks that `bounds` is composed of a list of tuples of the form (lower, upper), where lower <= upper and both are numeric. Also checks that `bounds` contains the appropriate number of dimensions, and that there is a `min_separation` between the bounds.

Parameters

- **bounds** (*tuple*) – Tuple of bounds of the form (min, max). *min* and *max* can either be scalars or 1-dimensional arrays.
- **shape** (*int, default: 0*) – Number of dimensions to be expected in `bounds`.
- **min_separation** (*float, default: 0.0*) – The minimum separation between *lower* and *upper* of each dimension. This separation is enforced if not already satisfied.
- **dtype** (*data-type, default: float*) – Data type of the returned bounds.

Returns **bounds**

Return type tuple

`diffprivlib.validation.`**`clip_to_norm`**(*array*, *clip*)

Clips the examples of a 2-dimensional array to a given maximum norm.

Parameters

- **array** (*np.ndarray*) – Array to be clipped. After clipping, all examples have a 2-norm of at most *clip*.
- **clip** (*float*) – Norm at which to clip each example

Returns **array** – The clipped array.

Return type np.ndarray

diffprivlib.validation.**clip_to_bounds**(*array*, *bounds*)

Clips the examples of a 2-dimensional array to given bounds.

Parameters

- **array** (*np.ndarray*) – Array to be clipped. After clipping, all examples have a 2-norm of at most *clip*.
- **bounds** (*tuple*) – Tuple of bounds of the form (min, max) which the array is to be clipped to. *min* and *max* must be scalar, unless array is 2-dimensional.

Returns **array** – The clipped array.

Return type np.ndarray

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[KOV17] Kairouz, Peter, Sewoong Oh, and Pramod Viswanath. "The composition theorem for differential privacy." IEEE Transactions on Information Theory 63.6 (2017): 4037-4049.

[VSB13] Vaidya, Jaideep, Basit Shafiq, Anirban Basu, and Yuan Hong. "Differentially private naive bayes classification." In 2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), vol. 1, pp. 571-576. IEEE, 2013.

[CMS11] Chaudhuri, Kamalika, Claire Monteleoni, and Anand D. Sarwate. "Differentially private empirical risk minimization." Journal of Machine Learning Research 12, no. Mar (2011): 1069-1109.

[She15] Sheffet, Or. "Private approximations of the 2nd-moment matrix using existing techniques in linear regression." arXiv preprint arXiv:1507.00056 (2015).

[IS16] Imtiaz, Hafiz, and Anand D. Sarwate. "Symmetric matrix perturbation for differentially-private principal component analysis." In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2339-2343. IEEE, 2016.

[SCL16] Su, Dong, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. "Differentially private k-means clustering." In Proceedings of the sixth ACM conference on data and application security and privacy, pp. 26-37. ACM, 2016.

[IS16b] Imtiaz, Hafiz, and Anand D. Sarwate. "Symmetric matrix perturbation for differentially-private principal component analysis." In 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 2339-2343. IEEE, 2016.

PYTHON MODULE INDEX

d

INDEX

H

I

K

L

M

N

P

R

S

T

U

V

W