SUFTware Documentation

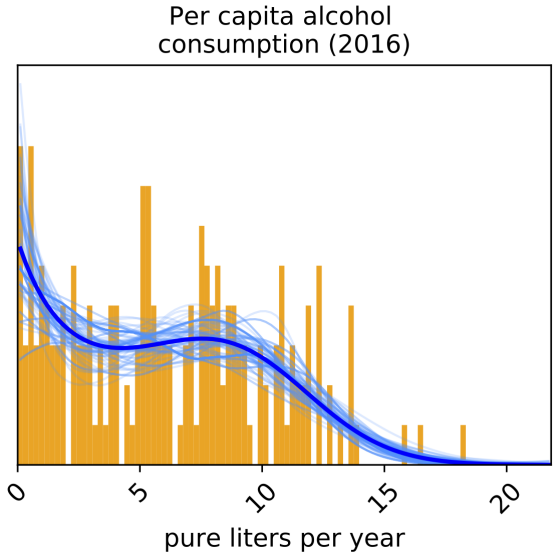
Release 0.15

Wei-Chia Chen, Ammar Tareen, Justin B. Kinney

Aug 19, 2019

Contents

Written by Wei-Chia Chen, Ammar Tareen, and Justin B. Kinney.



SUFTware (Statistics Using Field Theory) provides fast and lightweight Python implementations of Bayesian Field Theory algorithms for low-dimensional statistical inference. SUFTware currently supports the one-dimensional density estimation algorithm DEFT, described in[1],[2], and[3]. The image on the right shows DEFT applied to alcohol consumption data from the World Health Organization. This computation took about 0.25 seconds on a standard laptop computer.

Code for this and other examples can be found on the *Examples* page. The *Tutorial* page contains a short tutorial on how to use SUFTware. The *Documentation* page details the SUFTware API.

[1] Chen W, Tareen A, Kinney JB (2018) Density estimation on small datasets. *Phys Rev Lett 121:160605*. PDF.
[2] Kinney JB (2015) Unification of field theory and maximum entropy methods for learning probability densities. *Phys Rev E* 92:032107. PDF.
[3] Kinney JB (2014) Estimation of probability densities using scale-free field theories. *Phys Rev E* 90:011301(R). PDF.

CHAPTER 1

Installation

SUFTware can be installed from PyPI using the pip package manager (version 9.0.0 or higher). At the command line:

```
pip install suftware
```

The code for SUFTware is open source and available on GitHub.

CHAPTER 2

Quick Start

To make the figure shown above, do this from within Python:

```
import software as sw
sw.demo()
```

CHAPTER 3

Resources

3.1 Tutorial

We being by loading NumPy, PyPlot, SUFTware:

```
import numpy as np
import matplotlib.pyplot as plt
import suftware as sw

# Enable interactive plotting
plt.ion()
```

Next we simulate data from a Gamma distribution:

```
# Generate data from a Gamma distribution
np.random.seed(0)
data = np.random.gamma(shape=5, scale=1, size=100)
```

To estimate the probability density, do this:

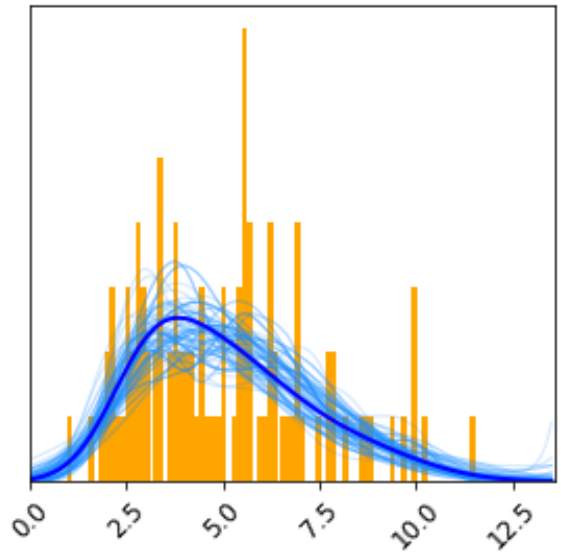
```
# Perform DEFT density estimation using SUFTware
density = sw.DensityEstimator(data)
```

This creates an instance of the `sw.DensityEstimator` class. The DEFT density estimation algorithm is run as part as part of this object's initialization process.

To quickly view the estimated probability density, use `density.plot()`:

```
# Plot density estimate using built-in plotting routine and save to file
density.plot(save_as='tutorial_1.png')
```

This will create a matplotlib figure resembling the one below. A histogram of the data is shown in orange, the optimal density estimate is shown in blue, and plausible densities are shown in light blue. Because the optional `save_as` argument is set, this plot is also saved to a PNG file. Other optional arguments to `density.plot()` can be used to specify styling options. See *Documentation* for more information.



DEFT estimates each probability distribution on a grid contained within a finite bounding box. Both the bounding box and the grid were set automatically in this example, but these as well as other grid characteristics can be set by the user by passing additional parameters to the `DensityEstimator` constructor. See *Documentation* for more information.

Information about the grid and bounding box are stored in the attributes of `density`:

- `density.bounding_box`: Lower and upper edges of the bounding box.
- `density.grid`: Locations of the gridpoints used.
- `density.grid_spacing`: Distance between neighboring grid points.
- `density.num_grid_points`: Number of grid points used.

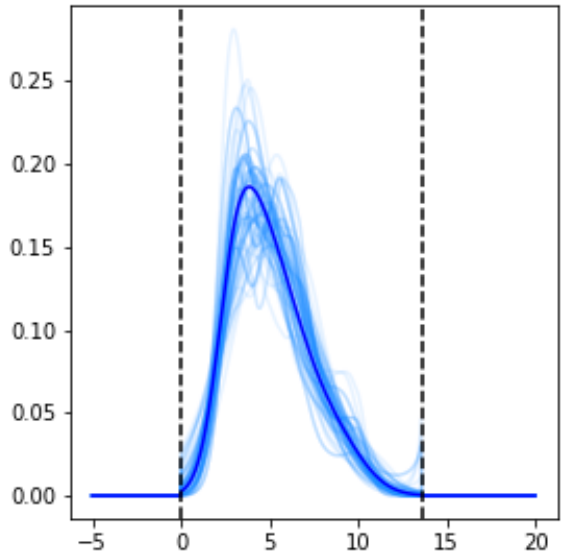
The values of the optimal density estimate at each grid point are stored in `density.values`. The `density.evaluate()` method allows this density to be evaluated at any other set of locations. The ensemble of posterior-sampled densities can be evaluated in a similar manner. Note that all estimated distributions evaluate to zero outside of the bounding box:

```
# Create new grid
new_grid = np.linspace(-5,20,10000)

# Evaluate optimal density on new grid
new_values = density.evaluate(new_grid)

# Evaluate sampled densities on new grid
new_sampled_values = density.evaluate_samples(new_grid)

# Create figure
plt.figure(figsize=[4,4])

# Plot optimal and posterior-sampled densities
plt.plot(new_grid, new_sampled_values, color='dodgerblue', alpha=.1)
plt.plot(new_grid, new_values, color='blue')

# Draw lines indicating bounding box
plt.axvline(density.bounding_box[0], linestyle='--', color='black')
```

(continues on next page)

(continued from previous page)

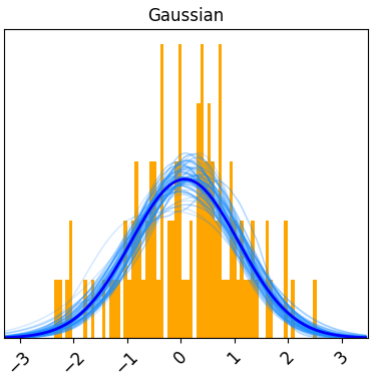
```
plt.axvline(density.bounding_box[1], linestyle='--', color='black')

# Show plot
plt.tight_layout()
plt.savefig('tutorial_2.png')
plt.show()
```



See *Documentation* for more information on the SUFTware API.

3.2 Examples

3.2.1 Custom data



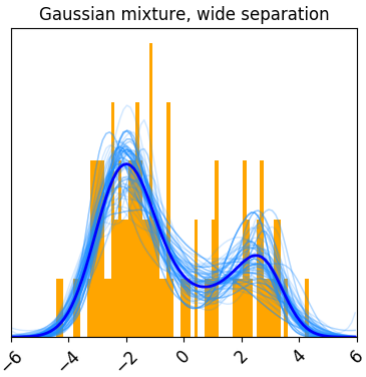

```
import numpy as np
import suftware as sw

# Generate random data
data = np.random.randn(100)
```

(continues on next page)

(continued from previous page)

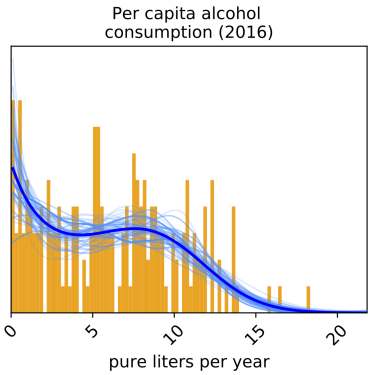
```
# Perform one-dimensional density estimation
density = sw.DensityEstimator(data)

# Plot results and save to file
density.plot(title='Gaussian')
```

3.2.2 Simulated data




```
import suftware as sw

# Simulate data using a pre-specified distribution
dataset = sw.SimulatedDataset(distribution='wide', num_data_points=100)

# Perform one-dimensional density estimation
density = sw.DensityEstimator(dataset.data, bounding_box=dataset.bounding_box)

# Plot results and save to file
density.plot(title='Gaussian mixture, wide separation')
```

3.2.3 Real data



```
import suftware as sw

# Retrieve data included with SUFTware
dataset = sw.ExampleDataset('who.alcohol_consumption')

# Perform one-dimensional density estimation
density = sw.DensityEstimator(dataset.data)

# Plot results and annotate with metadata
density.plot(title=dataset.description, xlabel=dataset.units)
```

3.3 Documentation

3.3.1 sw.DensityEstimator

class software.**DensityEstimator**(*data*, *grid=None*, *grid_spacing=None*, *num_grid_points=None*, *bounding_box=None*, *alpha=3*, *periodic=False*, *num_posterior_samples=100*, *max_t_step=1.0*, *tolerance=1e-06*, *resolution=0.1*, *sample_only_at_l_star=False*, *max_log_evidence_ratio_drop=20*, *evaluation_method_for_Z='Lap'*, *num_samples_for_Z=1000*, *seed=None*, *print_t=False*)

Estimates a 1D probability density from sampled data.

Parameters

data: (set, list, or np.array of numbers) An array of data from which the probability density will be estimated. Infinite or NaN values will be discarded.

grid: (1D np.array) An array of evenly spaced grid points on which the probability density will be estimated. Default value is `None`, in which case the grid is set automatically.

grid_spacing: (float > 0) The distance at which to space neighboring grid points. Default value is `None`, in which case this spacing is set automatically.

num_grid_points: (int) The number of grid points to draw within the data domain. Restricted to `2*alpha <= num_grid_points <= 1000`. Default value is `None`, in which case the number of grid points is chosen automatically.

bounding_box: ([float, float]) The boundaries of the data domain, within which the probability density will be estimated. Default value is `None`, in which case the bounding box is set automatically to encompass all of the data.

alpha: (int) The order of derivative constrained in the definition of smoothness. Restricted to `1 <= alpha <= 4`. Default value is 3.

periodic: (bool) Whether or not to impose periodic boundary conditions on the estimated probability density. Default False, in which case no boundary conditions are imposed.

num_posterior_samples: (int >= 0) Number of samples to draw from the Bayesian posterior. Restricted to 0 <= num_posterior_samples <= MAX_NUM_POSTERIOR_SAMPLES.

max_t_step: (float > 0) Upper bound on the amount by which the parameter `t` in the DEFT algorithm is incremented when tracing the MAP curve. Default value is 1.0.

tollerance: (float > 0) Sets the convergence criterion for the corrector algorithm used in tracing the MAP curve.

resolution: (float > 0) The maximum geodesic distance allowed for neighboring points on the MAP curve.

sample_only_at_l_star: (boolean) Specifies whether to let l vary when sampling from the Bayesian posterior.

max_log_evidence_ratio_drop: (float > 0) If set, MAP curve tracing will terminate prematurely when max_log_evidence - current_log_evidence > max_log_evidence_ratio_drop.

evaluation_method_for_Z: (string) Method of evaluation of partition function Z. Possible values: 'Lap' : Laplace approximation (default). 'Lap+Imp' : Laplace approximation + importance sampling. 'Lap+Fey' : Laplace approximation + Feynman diagrams.

num_samples_for_Z: (int >= 0) Number of posterior samples to use when evaluating the paritation function Z. Only has an affect when `evaluation_method_for_Z = 'Lap+Imp'`.

seed: (int) Seed provided to the random number generator before density estimation commences. For development purposes only.

print_t: (bool) Whether to print the values of `t` while tracing the MAP curve. For development purposes only.

Attributes

grid: The grid points at which the probability density was be estimated. (1D np.array)

grid_spacing: The distance between neighboring grid points. (float > 0)

num_grid_points: The number of grid points used. (int)

bounding_box: The boundaries of the data domain within which the probability density was be estimated. ([float, float])

histogram: A histogram of the data using `grid` for the centers of each bin. (1D np.array)

values: The values of the optimal (i.e., MAP) density at each grid point. (1D np.array)

sample_values: The values of the posterior sampled densities at each grid point. The first index specifies grid points, the second posterior samples. (2D np.array)

sample_weights: The importance weights corresponding to each posterior sample. (1D np.array)

Methods

evaluate(x)	Evaluate the optimal (i.e.
evaluate_samples(x[, resample])	Evaluate sampled densities at specified locations.
get_stats([use_weights, show_samples])	Computes summary statistics for the estimated density
plot([ax, save_as, resample, figsize, ...])	Plot the MAP density, the posterior sampled densities, and the data histogram.

evaluate(*x*)

Evaluate the optimal (i.e. MAP) density at the supplied value(s) of x.

Parameters

x: (number or list-like collection of numbers) The locations in the data domain at which to evaluate the MAP density.

Returns

A float or 1D np.array representing the values of the MAP density at

the specified locations.

evaluate_samples(*x*, *resample=True*)
Evaluate sampled densities at specified locations.

Parameters

x: (number or list-like collection of numbers) The locations in the data domain at which to evaluate sampled density.

resample: (bool) Whether to use importance resampling, i.e., should the values returned be from the original samples (obtained using a Laplace approximated posterior) or should they be resampled to account for the deviation between the true Bayesian posterior and its Laplace approximation.

Returns

A 1D np.array (if x is a number) or a 2D np.array (if x is list-like),

representing the values of the posterior sampled densities at the

specified locations. The first index corresponds to values in x, the

second to sampled densities.

get_stats(*use_weights=True*, *show_samples=False*)
Computes summary statistics for the estimated density

Parameters

show_samples: (bool) If True, summary stats are computed for each posterior sample. If False, summary stats are returned for the "star" estimate, the histogram, and the maxent estimate, along with the mean and RMSD values of these stats across posterior samples.

use_weights: (bool) If True, mean and RMSD are computed using importance weights.

Returns

df: (pd.DataFrame) A pandas data frame listing summary statistics for the estimated probability densities. These summary statistics include "entropy" (in bits), "mean", "variance", "skewness", and "kurtosis". If `show_samples = False`, results will be shown for the best estimate, as well as mean and RMDS values across all samples. If `show_samples = True`, results will be shown for each sample. A column showing column weights will also be included.

plot(*ax=None*, *save_as=None*, *resample=True*, *figsize=(4, 4)*, *fontsize=12*, *title=''*, *xlabel=''*, *tight_layout=False*, *show_now=True*, *show_map=True*, *map_color='blue'*, *map_linewidth=2*, *map_alpha=1*, *num_posterior_samples=None*, *posterior_color='dodgerblue'*, *posterior_linewidth=1*, *posterior_alpha=0.2*, *show_histogram=True*, *histogram_color='orange'*, *histogram_alpha=1*, *show_maxent=False*, *maxent_color='maroon'*, *maxent_linewidth=1*, *maxent_alpha=1*, *backend='TkAgg'*)
Plot the MAP density, the posterior sampled densities, and the data histogram.

Parameters

ax: (plt.Axes) A matplotlib axes object on which to draw. If None, one will be created

save_as: (str) Name of file to save plot to. File type is determined by file extension.

resample: (bool) If True, sampled densities will be ploted only after importance resampling.

figsize: ([float, float]) Figure size as (width, height) in inches.

fontsize: (float) Size of font to use in plot annotation.

title: (str) Plot title.

xlabel: (str) Plot xlabel.

tight_layout: (bool) Whether to call plt.tight_layout() after rendering graphics.

show_now: (bool) Whether to show the plot immediately by calling plt.show().

show_map: (bool) Whether to show the MAP density.

map_color: (color spec) MAP density color.

map_linewidth: (float) MAP density linewidth.

map_alpha: (float) Map density opacity (between 0 and 1).

num_posterior_samples: (int) Number of posterior samples to display. If this is greater than the number of posterior samples taken, all of the samples taken will be shown.

posterior_color: (color spec) Sampled density color.

posterior_linewidth: (float) Sampled density linewidth.

posterior_alpha: (float) Sampled density opactity (between 0 and 1).

show_histogram: (bool) Whether to show the (normalized) data histogram.

histogram_color: (color spec) Face color of the data histogram.

histogram_alpha: (float) Data histogram opacity (between 0 and 1).

show_maxent: (bool) Whether to show the MaxEnt density estimate.

maxent_color: (color spect) Line color of the MaxEnt density estimate.

maxent_alpha: (float) MaxEnt opacity (between 0 and 1).

backend: (str) Backend specification to send to sw.enable_graphics().

Returns

None.

3.3.2 sw.ExampleDataset

class suftware.**ExampleDataset**(*dataset='old_faithful_eruption_times'*)

Provides an interface to example data provided with the SUFTware package.

Parameters

dataset: (str) Name of dataset to load. Run sw.ExampleDataset.list() to see which datasets are available.

Attributes

data: (np.array) An array containing sampled data

details: (np.array, optional) Optional return value containing meta information

Methods

list()	Return list of available datasets.

static list()
Return list of available datasets.

3.3.3 sw.SimulatedDataset

class software.**SimulatedDataset**(*distribution='gaussian'*, *num_data_points=100*, *seed=None*)
Simulates data from a variety of distributions.

Parameters

distribution: (str) The distribution from which to draw data. Run sw.SimulatedDataset.list() to which distributions are available.

num_data_points: (int > 0) The number of data points to simulate. Must satisfy 0 <= N <= MAX_DATASET_SIZE.

seed: (int) Seed passed to random number generator.

Attributes

data: (np.array) The simulated dataset

bounding_box: ([float, float]) Bounding box within which data is generated.

distribution: (str) Name of the simualted distribution

pdf_js: (str) Formula for probability density in JavaScript

pdf_py: (str) Formula for probaiblity density in Python.

periodic: (bool) Whether the simulated distribution is periodic within bounding_box.

Methods

list()	Return list of valid distributions.

static list()
Return list of valid distributions.

3.3.4 sw.enable_graphics

software.**enable_graphics**(*backend='TkAgg'*)
Enable graphical output by software.

This function should be _run before any calls are made to DensityEstimator.plot(). This is not always necessary, since DensityEstimator.plot() itself will call this function if necessary. However, when plotting inline using the iPython notebook, this function must be called before the magic function `%matplotlib inline`, e.g.:

```
import suftware as sw
sw.enable_graphics()
%matplotlib inline
```

If this function is never called, suftware can be _run without importing matplotlib. This can be useful, for instance, when distributing jobs across the nodes of a high performance computing cluster.

Parameters

backend: (str) Graphical backend to be passed to matplotlib.use(). See the matplotlib documentation for more information on graphical backends.

Returns

None.

3.3.5 sw.demo

`suftware.demo(example='real_data')`

Performs a demonstration of suftware.

Parameters

example: (str) A string specifying which demo to run. Must be 'real_data', 'simulated_data', or 'custom_data'.

CHAPTER 4

Contact

For technical assistance or to report bugs, please contact Ammar Tareen.

For more general correspondence, please contact Justin Kinney.

Other links:

- Kinney Lab
- Simons Center for Quantitative Biology
- Cold Spring Harbor Laboratory

CHAPTER 5

References

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index