statsmodels.nonparametric.smoothers_lowess.lowess

`statsmodels.nonparametric.smoothers_lowess.`**`lowess`**(*endog*, *exog*, *frac=0.6666666666666666*, *it=3*, *delta=0.0*, *is_sorted=False*, *missing='drop'*, *return_sorted=True*) [source]

LOWESS (Locally Weighted Scatterplot Smoothing)

A lowess function that outs smoothed estimates of endog at the given exog values from points (exog, endog)

Parameters:

endog: 1-D numpy array

The y-values of the observed points

exog: 1-D numpy array

The x-values of the observed points

frac: float

Between 0 and 1. The fraction of the data used when estimating each y-value.

it: int

The number of residual-based reweightings to perform.

delta: float

Distance within which to use linear-interpolation instead of weighted regression.

is_sorted : bool

If False (default), then the data will be sorted by exog before calculating lowess. If True, then it is assumed that the data is already sorted by exog.

missing : str

Available options are 'none', 'drop', and 'raise'. If 'none', no nan checking is done. If 'drop', any observations with nans are dropped. If 'raise', an error is raised. Default is 'drop'.

return_sorted : bool

If True (default), then the returned array is sorted by exog and has missing (nan or infinite) observations removed. If False, then the returned array is in the same length and the same sequence of observations as the input array.

Returns:

out: ndarray, float

The returned array is two-dimensional if return_sorted is True, and one dimensional if return_sorted is False. If return_sorted is True, then a numpy array with two columns. The first column contains the sorted x (exog) values and the second column the associated estimated y (endog) values. If return_sorted is False, then only the fitted values are returned, and the observations will be in the same order as the input arrays.

Notes

This lowess function implements the algorithm given in the reference below using local linear estimates.

Suppose the input data has N points. The algorithm works by estimating the *smooth* y_i by taking the frac*N closest points to (x_i,y_i) based on their x values and estimating y_i using a weighted linear regression. The weight for (x_j,y_j) is tricube function applied to abs(x_i-x_j).

If it > 1, then further weighted local linear regressions are performed, where the weights are the same as above times the _lowess_bisquare function of the residuals. Each iteration takes approximately the same amount of time as the original fit, so these iterations are expensive. They are most useful when the noise has extremely heavy tails, such as Cauchy noise. Noise with less heavy-tails, such as t-distributions with df>2, are less problematic. The weights downgrade the influence of points with large residuals. In the extreme case, points whose residuals are larger than 6 times the median absolute residual are given weight 0.

delta can be used to save computations. For each *x_i*, regressions are skipped for points closer than *delta*. The next regression is fit for the farthest point within delta of *x_i* and all points in between are estimated by linearly interpolating between the two regression fits.

Judicious choice of delta can cut computation time considerably for large data (N > 5000). A good choice is `delta = 0.01 * range(exog)`.

Some experimentation is likely required to find a good choice of *frac* and *iter* for a particular dataset.

References

Cleveland, W.S. (1979) “Robust Locally Weighted Regression and Smoothing Scatterplots”. Journal of the American Statistical Association 74 (368): 829-836.

Examples

The below allows a comparison between how different the fits from lowess for different values of frac can be.

```
>>> import numpy as np
>>> import statsmodels.api as sm
>>> lowess = sm.nonparametric.lowess
>>> x = np.random.uniform(low = -2*np.pi, high = 2*np.pi, size=500)
>>> y = np.sin(x) + np.random.normal(size=len(x))
>>> z = lowess(y, x)
>>> w = lowess(y, x, frac=1./3)
```

This gives a similar comparison for when it is 0 vs not.

```
>>> import numpy as np
>>> import scipy.stats as stats
>>> import statsmodels.api as sm
>>> lowess = sm.nonparametric.lowess
>>> x = np.random.uniform(low = -2*np.pi, high = 2*np.pi, size=500)
>>> y = np.sin(x) + stats.cauchy.rvs(size=len(x))
>>> z = lowess(y, x, frac= 1./3, it=0)
>>> w = lowess(y, x, frac=1./3)
```