**Standardization** of datasets is a **common requirement for many machine learning estimators** implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with **zero mean and unit variance**.

In practice we often ignore the shape of the distribution and just transform the data to center it by removing the mean value of each feature, then scale it by dividing non-constant features by their standard deviation.

For instance, many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the l1 and l2 regularizers of linear models) assume that all features are centered around zero and have variance in the same order. If a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

The function [**scale**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html#sklearn.preprocessing.scale) provides a quick and easy way to perform this operation on a single array-like dataset:

>>>

**>>> from** **sklearn** **import** preprocessing

**>>> import** **numpy** **as** **np**

**>>>** X\_train = np.array([[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]])

**>>>** X\_scaled = preprocessing.scale(X\_train)

**>>>** X\_scaled

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

Scaled data has zero mean and unit variance:

>>>

**>>>** X\_scaled.mean(axis=0)

array([0., 0., 0.])

**>>>** X\_scaled.std(axis=0)

array([1., 1., 1.])

The preprocessing module further provides a utility class **[StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler" \o "sklearn.preprocessing.StandardScaler)** that implements the Transformer API to compute the mean and standard deviation on a training set so as to be able to later reapply the same transformation on the testing set. This class is hence suitable for use in the early steps of a **[sklearn.pipeline.Pipeline](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html" \l "sklearn.pipeline.Pipeline" \o "sklearn.pipeline.Pipeline)**:

>>>

**>>>** scaler = preprocessing.StandardScaler().fit(X\_train)

**>>>** scaler

StandardScaler()

**>>>** scaler.mean\_

array([1. ..., 0. ..., 0.33...])

**>>>** scaler.scale\_

array([0.81..., 0.81..., 1.24...])

**>>>** scaler.transform(X\_train)

array([[ 0. ..., -1.22..., 1.33...],

[ 1.22..., 0. ..., -0.26...],

[-1.22..., 1.22..., -1.06...]])

The scaler instance can then be used on new data to transform it the same way it did on the training set:

>>>

**>>>** X\_test = [[-1., 1., 0.]]

**>>>** scaler.transform(X\_test)

array([[-2.44..., 1.22..., -0.26...]])

It is possible to disable either centering or scaling by either passing with\_mean=False or with\_std=False to the constructor of **[StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler" \o "sklearn.preprocessing.StandardScaler)**.

### **6.3.1.1. Scaling features to a range**

An alternative standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using **[MinMaxScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html" \l "sklearn.preprocessing.MinMaxScaler" \o "sklearn.preprocessing.MinMaxScaler)** or **[MaxAbsScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html" \l "sklearn.preprocessing.MaxAbsScaler" \o "sklearn.preprocessing.MaxAbsScaler)**, respectively.

The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.

Here is an example to scale a toy data matrix to the [0, 1] range:

>>>

**>>>** X\_train = np.array([[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]])

**...**

**>>>** min\_max\_scaler = preprocessing.MinMaxScaler()

**>>>** X\_train\_minmax = min\_max\_scaler.fit\_transform(X\_train)

**>>>** X\_train\_minmax

array([[0.5 , 0. , 1. ],

[1. , 0.5 , 0.33333333],

[0. , 1. , 0. ]])

The same instance of the transformer can then be applied to some new test data unseen during the fit call: the same scaling and shifting operations will be applied to be consistent with the transformation performed on the train data:

>>>

**>>>** X\_test = np.array([[-3., -1., 4.]])

**>>>** X\_test\_minmax = min\_max\_scaler.transform(X\_test)

**>>>** X\_test\_minmax

array([[-1.5 , 0. , 1.66666667]])

It is possible to introspect the scaler attributes to find about the exact nature of the transformation learned on the training data:

>>>

**>>>** min\_max\_scaler.scale\_

array([0.5 , 0.5 , 0.33...])

**>>>** min\_max\_scaler.min\_

array([0. , 0.5 , 0.33...])

If **[MinMaxScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html" \l "sklearn.preprocessing.MinMaxScaler" \o "sklearn.preprocessing.MinMaxScaler)** is given an explicit feature\_range=(min, max) the full formula is:

X\_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X\_scaled = X\_std \* (max - min) + min

[**MaxAbsScaler**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler) works in a very similar fashion, but scales in a way that the training data lies within the range [-1, 1] by dividing through the largest maximum value in each feature. It is meant for data that is already centered at zero or sparse data.

Here is how to use the toy data from the previous example with this scaler:

>>>

**>>>** X\_train = np.array([[ 1., -1., 2.],

**...**  [ 2., 0., 0.],

**...**  [ 0., 1., -1.]])

**...**

**>>>** max\_abs\_scaler = preprocessing.MaxAbsScaler()

**>>>** X\_train\_maxabs = max\_abs\_scaler.fit\_transform(X\_train)

**>>>** X\_train\_maxabs

array([[ 0.5, -1. , 1. ],

[ 1. , 0. , 0. ],

[ 0. , 1. , -0.5]])

**>>>** X\_test = np.array([[ -3., -1., 4.]])

**>>>** X\_test\_maxabs = max\_abs\_scaler.transform(X\_test)

**>>>** X\_test\_maxabs

array([[-1.5, -1. , 2. ]])

**>>>** max\_abs\_scaler.scale\_

array([2., 1., 2.])

As with [**scale**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html#sklearn.preprocessing.scale), the module further provides convenience functions **[minmax\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.html" \l "sklearn.preprocessing.minmax_scale" \o "sklearn.preprocessing.minmax_scale)** and **[maxabs\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.maxabs_scale.html" \l "sklearn.preprocessing.maxabs_scale" \o "sklearn.preprocessing.maxabs_scale)**if you don’t want to create an object.

### **6.3.1.2. Scaling sparse data**

Centering sparse data would destroy the sparseness structure in the data, and thus rarely is a sensible thing to do. However, it can make sense to scale sparse inputs, especially if features are on different scales.

[**MaxAbsScaler**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler) and **[maxabs\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.maxabs_scale.html" \l "sklearn.preprocessing.maxabs_scale" \o "sklearn.preprocessing.maxabs_scale)** were specifically designed for scaling sparse data, and are the recommended way to go about this. However, [**scale**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html#sklearn.preprocessing.scale) and **[StandardScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html" \l "sklearn.preprocessing.StandardScaler" \o "sklearn.preprocessing.StandardScaler)** can accept scipy.sparsematrices as input, as long as with\_mean=False is explicitly passed to the constructor. Otherwise a ValueError will be raised as silently centering would break the sparsity and would often crash the execution by allocating excessive amounts of memory unintentionally. **[RobustScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html" \l "sklearn.preprocessing.RobustScaler" \o "sklearn.preprocessing.RobustScaler)** cannot be fitted to sparse inputs, but you can use the transform method on sparse inputs.

Note that the scalers accept both Compressed Sparse Rows and Compressed Sparse Columns format (see scipy.sparse.csr\_matrix and scipy.sparse.csc\_matrix). Any other sparse input will be **converted to the Compressed Sparse Rows representation**. To avoid unnecessary memory copies, it is recommended to choose the CSR or CSC representation upstream.

Finally, if the centered data is expected to be small enough, explicitly converting the input to an array using the toarray method of sparse matrices is another option.

### **6.3.1.3. Scaling data with outliers**

If your data contains many outliers, scaling using the mean and variance of the data is likely to not work very well. In these cases, you can use **[robust\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.robust_scale.html" \l "sklearn.preprocessing.robust_scale" \o "sklearn.preprocessing.robust_scale)** and **[RobustScaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html" \l "sklearn.preprocessing.RobustScaler" \o "sklearn.preprocessing.RobustScaler)** as drop-in replacements instead. They use more robust estimates for the center and range of your data.

**References:**

Further discussion on the importance of centering and scaling data is available on this FAQ: [Should I normalize/standardize/rescale the data?](http://www.faqs.org/faqs/ai-faq/neural-nets/part2/section-16.html)

**Scaling vs Whitening**

It is sometimes not enough to center and scale the features independently, since a downstream model can further make some assumption on the linear independence of the features.

To address this issue you can use **[sklearn.decomposition.PCA](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html" \l "sklearn.decomposition.PCA" \o "sklearn.decomposition.PCA)** with whiten=True to further remove the linear correlation across features.

**Scaling a 1D array**

All above functions (i.e. [**scale**](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.scale.html#sklearn.preprocessing.scale), **[minmax\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.minmax_scale.html" \l "sklearn.preprocessing.minmax_scale" \o "sklearn.preprocessing.minmax_scale)**, **[maxabs\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.maxabs_scale.html" \l "sklearn.preprocessing.maxabs_scale" \o "sklearn.preprocessing.maxabs_scale)**, and **[robust\_scale](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.robust_scale.html" \l "sklearn.preprocessing.robust_scale" \o "sklearn.preprocessing.robust_scale)**) accept 1D array which can be useful in some specific case.