COSI 104a Introduction to machine learning

Chapter 5 –Regression

Instructor: Dr. Hongfu Liu
Email: hongfuliu@brandeis.edu

Main source: https://www-users.cs.umn.edu/~kumar001/dmbook/index.php

Regression Models

Answer “What is the relationship between the variables?”

- Dependent (response) variable – what to be predicted.
- One or more numerical or categorical independent (explanatory) variables.

Given a dataset $\{\langle \vec{x}_n, y_n \rangle\}_{n=1,\dots,N}$, where $\vec{x}_n \in \mathcal{R}^d$ and $y_n \in \mathcal{R}$

Build a function $y = f(\vec{x}, \vec{w})$ to predict the output y of a given sample $\vec{x}$, where $\vec{w}$ is the parameter of the function $f(\cdot)$

Simple Linear Regression

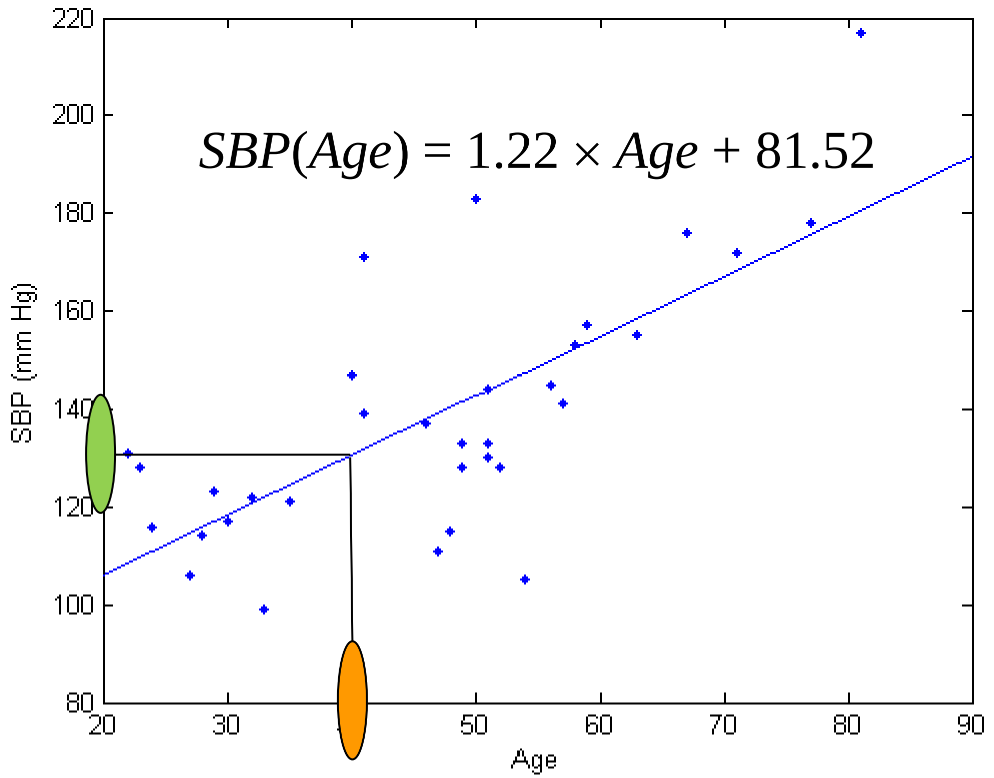
Systolic Blood Pressure (SBP)

Age	SBP (mm Hg)	Age	SBP (mm Hg)	Age	SBP (mm Hg)
22	131	41	139	52	128
23	128	41	171	54	105
24	116	46	137	56	145
27	106	47	111	57	141
28	114	48	115	58	153
29	123	49	133	59	157
30	117	49	128	63	155
32	122	50	183	67	176
33	99	51	130	71	172
35	121	51	133	77	178
40	147	51	144	81	217

adapted from Colton T. Statistics in Medicine. Boston: Little Brown, 1974

Simple Linear Regression

$f(x) = ax + b$, where a measures the association between y and x



Prediction

Simple Linear Regression Model

A linear combination of the input variables

$$f(\vec{x}, \vec{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d$$

Let $\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$ $\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix}$

$$f(\vec{x}, \vec{w}) = \vec{w}^T \vec{x}$$

Estimate Parameters

Given $\{\langle \vec{x}_n, y_n \rangle\}_{n=1,\ldots,N}$ and $f(\vec{x}, \vec{w}) = \vec{w}^T \vec{x}$

Define the objective $Err = \sum_{n=1}^{N} \left(y_n - f(\vec{x}_n, \vec{w})\right)^2$

Arrange the data into the following

$$\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \qquad X = \begin{bmatrix} \vec{x}_1^T \\ \vec{x}_2^T \\ \vdots \\ \vec{x}_N^T \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N1} & x_{N2} & \cdots & x_{Nd} \end{bmatrix}$$

Obtain $Err = (\vec{y} - X\vec{w})^T (\vec{y} - X\vec{w})$

Solution (if X is full rank): $\vec{w} = (X^T X)^{-1} X^T \vec{y}$ How?

Minimize $Err = (\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$

$$\frac{\partial Err}{\partial \vec{w}} = -2X^T(\vec{y} - X\vec{w})$$

$$\frac{\partial^2 Err}{\partial \vec{w}^2} = 2X^T X$$

Assuming X is full rank

$$\frac{\partial Err}{\partial \vec{w}} = 0 \quad \Rightarrow \quad \vec{w} = (X^T X)^{-1} X^T \vec{y}$$

Discussion

$\vec{w} = (X^T X)^{-1} X^T \vec{y}$ What if $(X^T X)$ is singular?

How to deal with categorical variables?

Create dummy variables, one for each categorical value, and delete one dummy variable.

Coefficient of Determination R^2

R^2 is the proportion of the variation in the dependent variable that is predictable from the independent variable(s).

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

$$SS_{residual} = Err = \sum_{n=1}^{N} \left(y_n - f(\vec{x}_n, \vec{w})\right)^2$$

$$SS_{total} = \sum_{n=1}^{N} (y_n - \bar{y})^2 \quad \text{where } \bar{y} = \frac{1}{N} \textstyle\sum_{n=1}^{N} y_n$$

$\bar{y}$ $\downarrow$ equivalent to

$$f(\vec{x}, \vec{w}) = \bar{y} + 0 \cdot x_1 + \cdots + 0 \cdot x_d$$

Discussion

$$R^2 = 1 - \frac{SS_{residual}}{SS_{total}}$$

R^2: The larger or smaller, the better?

Discussion

- Very often, the input variables are of very different scale.
- Normalize all input variables so they are “comparable” in scale.
  - One popular way: apply a linear transformation to each variable

- Comparable scales lead to more stable numerical performance
- Comparable scales make it easier to interpret coefficients

Linear Basis Function Model

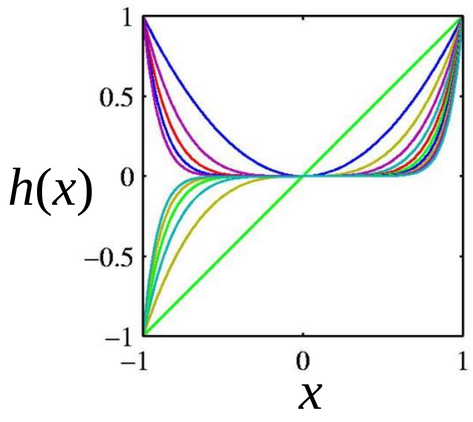
$$f(\vec{x}, \vec{w}) = w_0 + w_1 x_1 + \cdots + w_d x_d \qquad \text{input} \xleftrightarrow{\substack{\text{linear}\\ \text{relationship}}} \text{output}$$

$$f(\vec{x}, \vec{w}) = w_0 + w_1 h_1(\vec{x}) + \cdots + w_m h_m(\vec{x}) \qquad \text{input} \xleftrightarrow{\substack{\text{nonlinear}\\ \text{relationship}}} \text{output}$$

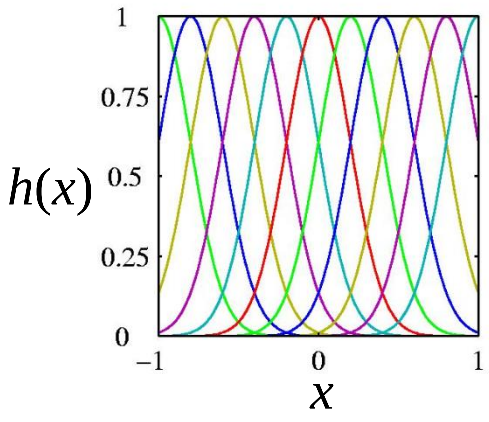
$\{h_i(\vec{x})\}$ is a set of non-linear basis functions (or *kernel* functions)

We can rewrite $f(\vec{x}, \vec{w}) = \vec{w}^T \begin{bmatrix} 1 \\ h_1(\vec{x}) \\ \vdots \\ h_m(\vec{x}) \end{bmatrix}$

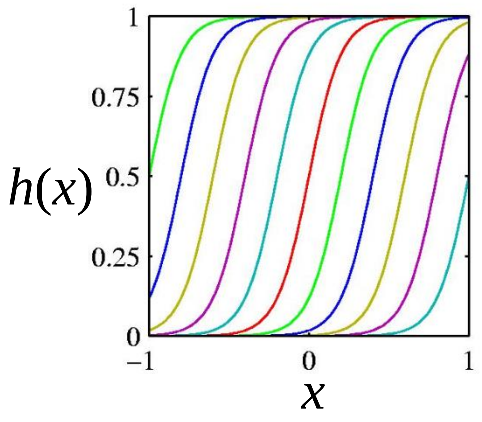
Examples of Basis Functions



polynomial



Gaussian



sigmoidal

Estimate Parameters

Let $\vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$ $\quad X = \begin{bmatrix} 1 & h_1(\vec{x}_1) & h_2(\vec{x}_1) & \cdots & h_m(\vec{x}_1) \\ 1 & h_1(\vec{x}_2) & h_2(\vec{x}_2) & \cdots & h_m(\vec{x}_2) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & h_1(\vec{x}_N) & h_2(\vec{x}_N) & \cdots & h_m(\vec{x}_N) \end{bmatrix}$

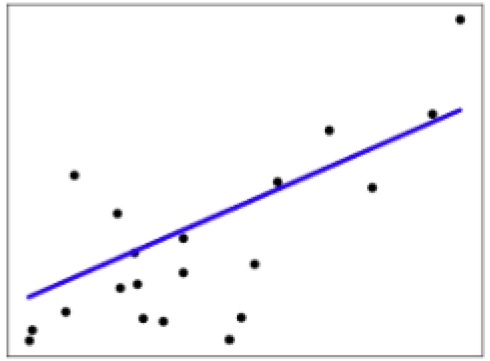
Define the objective function: $Err = (\vec{y} - X\vec{w})^T(\vec{y} - X\vec{w})$

Assume X is full rank $\quad \vec{w} = (X^T X)^{-1} X^T \vec{y}$

Linear Regression

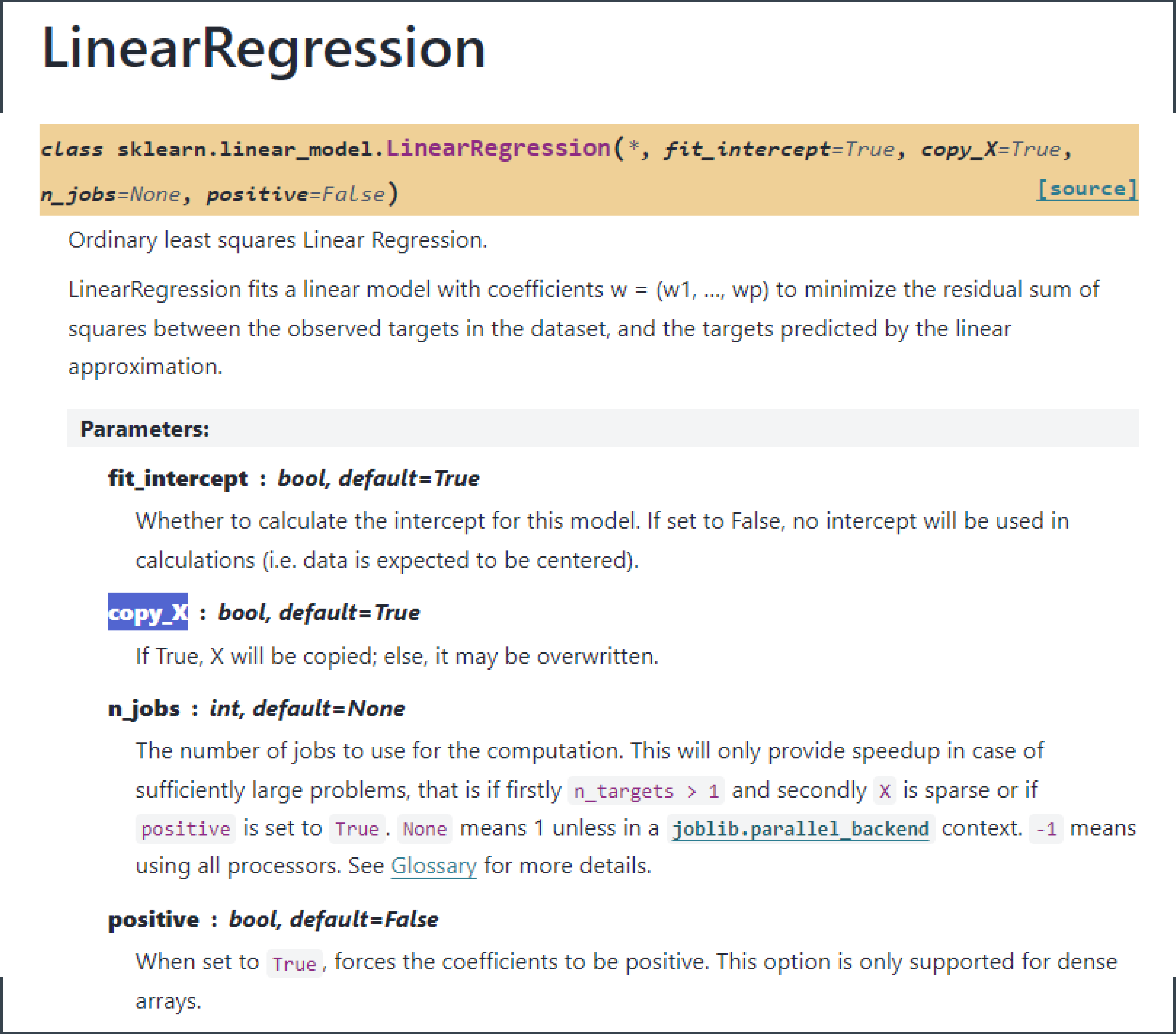
`LinearRegression` fits a linear model with coefficients $w = (w_1, \ldots, w_p)$ to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_{w} || X w - y||_2^2$$



`LinearRegression` will take in its `fit` method arrays `X`, `y` and will store the coefficients w of the linear model in its `coef_` member:

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```



LinearRegression

```
class sklearn.linear_model.LinearRegression(*, fit_intercept=True, copy_X=True,
n_jobs=None, positive=False)
```
[source]

Ordinary least squares Linear Regression.

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.

Parameters:

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.

n_jobs : ***int, default=None***

The number of jobs to use for the computation. This will only provide speedup in case of sufficiently large problems, that is if firstly `n_targets > 1` and secondly `X` is sparse or if `positive` is set to `True`. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

positive : ***bool, default=False***

When set to `True`, forces the coefficients to be positive. This option is only supported for dense arrays.

Attributes:

coef_ : *array of shape (n_features,) or (n_targets, n_features)*

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.

rank_ : *int*

Rank of matrix `X`. Only available when `X` is dense.

singular_ : *array of shape (min(X, y),)*

Singular values of `X`. Only available when `X` is dense.

intercept_ : *float or array of shape (n_targets,)*

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

n_features_in_ : *int*

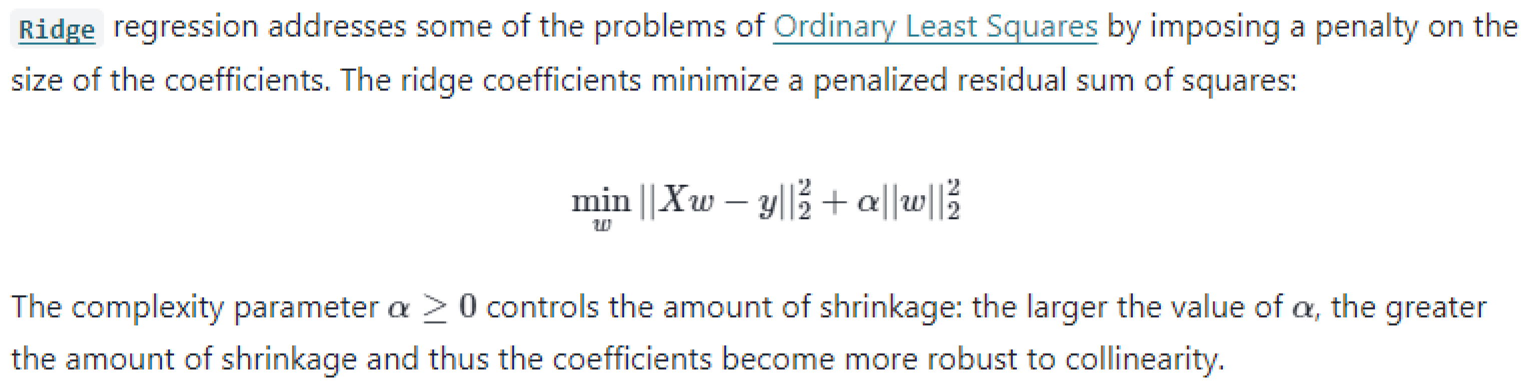
Number of features seen during fit.

Added in version 0.24.

feature_names_in_ : *ndarray of shape (`n_features_in_`,)*

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

Ridge regression



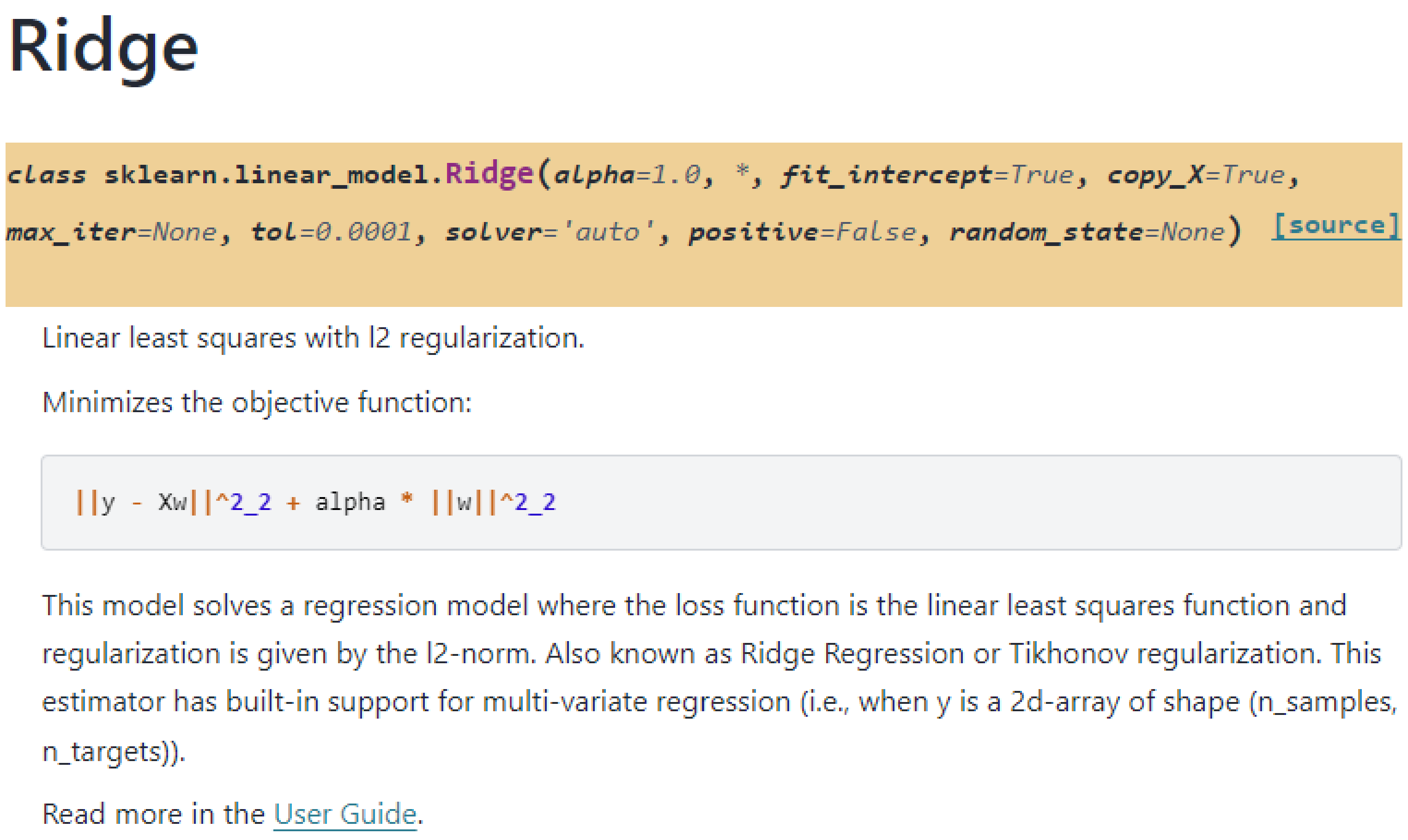
`Ridge` regression addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of the coefficients. The ridge coefficients minimize a penalized residual sum of squares:

$$\min_{w} || X w - y||_2^2 + \alpha ||w||_2^2$$

The complexity parameter $\alpha \geq 0$ controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

As with other linear models, `Ridge` will take in its `fit` method arrays `X`, `y` and will store the coefficients w of the linear model in its `coef_` member:

```
>>> from sklearn import linear_model
>>> reg = linear_model.Ridge(alpha=.5)
>>> reg.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
Ridge(alpha=0.5)
>>> reg.coef_
array([0.34545455, 0.34545455])
>>> reg.intercept_
0.13636...
```



Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True, max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None) [source]
```

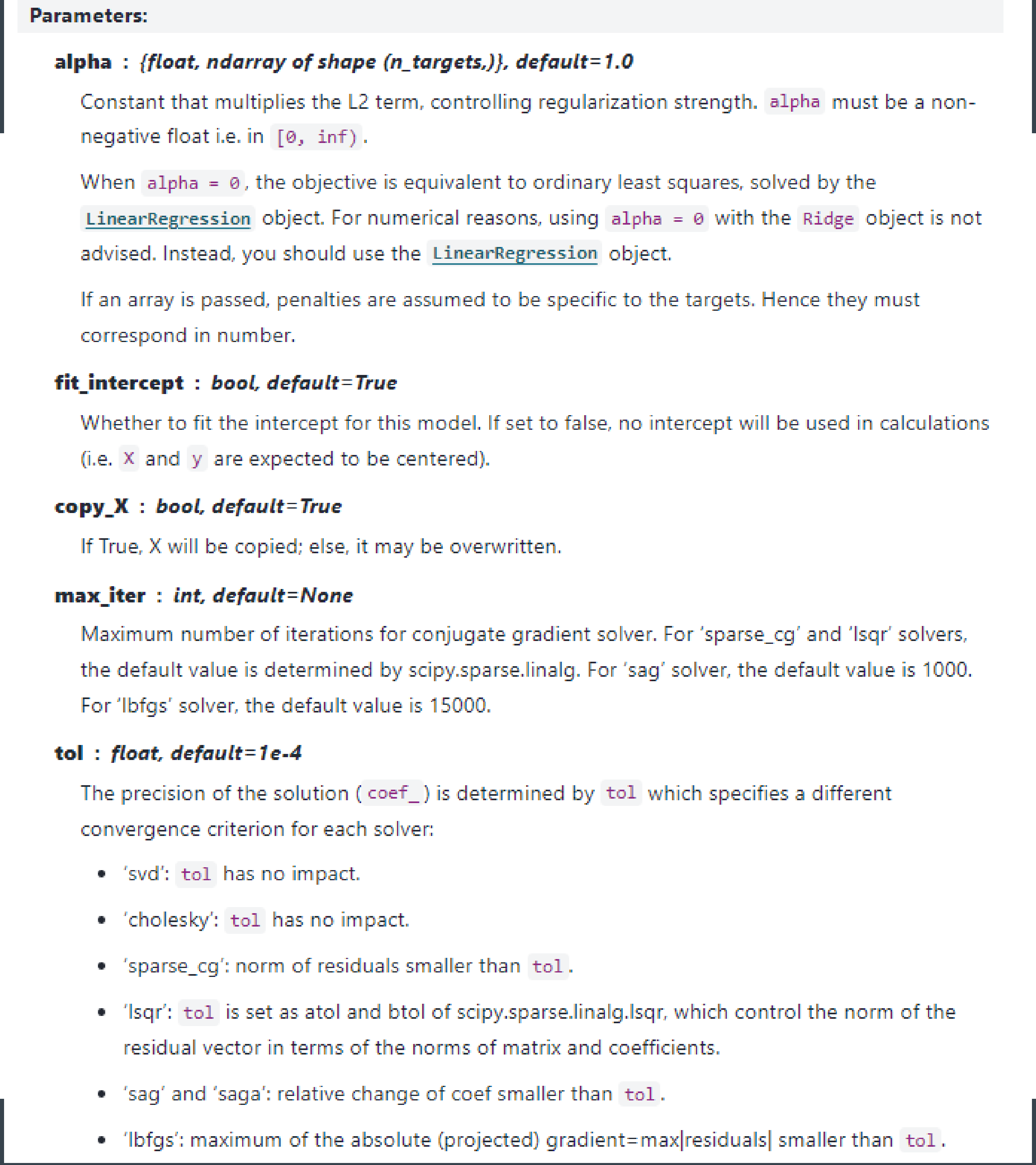
Linear least squares with l2 regularization.

Minimizes the objective function:

```
||y - Xw||^2_2 + alpha * ||w||^2_2
```

This model solves a regression model where the loss function is the linear least squares function and regularization is given by the l2-norm. Also known as Ridge Regression or Tikhonov regularization. This estimator has built-in support for multi-variate regression (i.e., when y is a 2d-array of shape (n_samples, n_targets)).

Read more in the User Guide.



Parameters:

alpha : ***{float, ndarray of shape (n_targets,)}, default=1.0***

Constant that multiplies the L2 term, controlling regularization strength. `alpha` must be a non-negative float i.e. in `[0, inf)`.

When `alpha = 0`, the objective is equivalent to ordinary least squares, solved by the `LinearRegression` object. For numerical reasons, using `alpha = 0` with the `Ridge` object is not advised. Instead, you should use the `LinearRegression` object.

If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.

fit_intercept : ***bool, default=True***

Whether to fit the intercept for this model. If set to false, no intercept will be used in calculations (i.e. `X` and `y` are expected to be centered).

copy_X : ***bool, default=True***

If True, X will be copied; else, it may be overwritten.

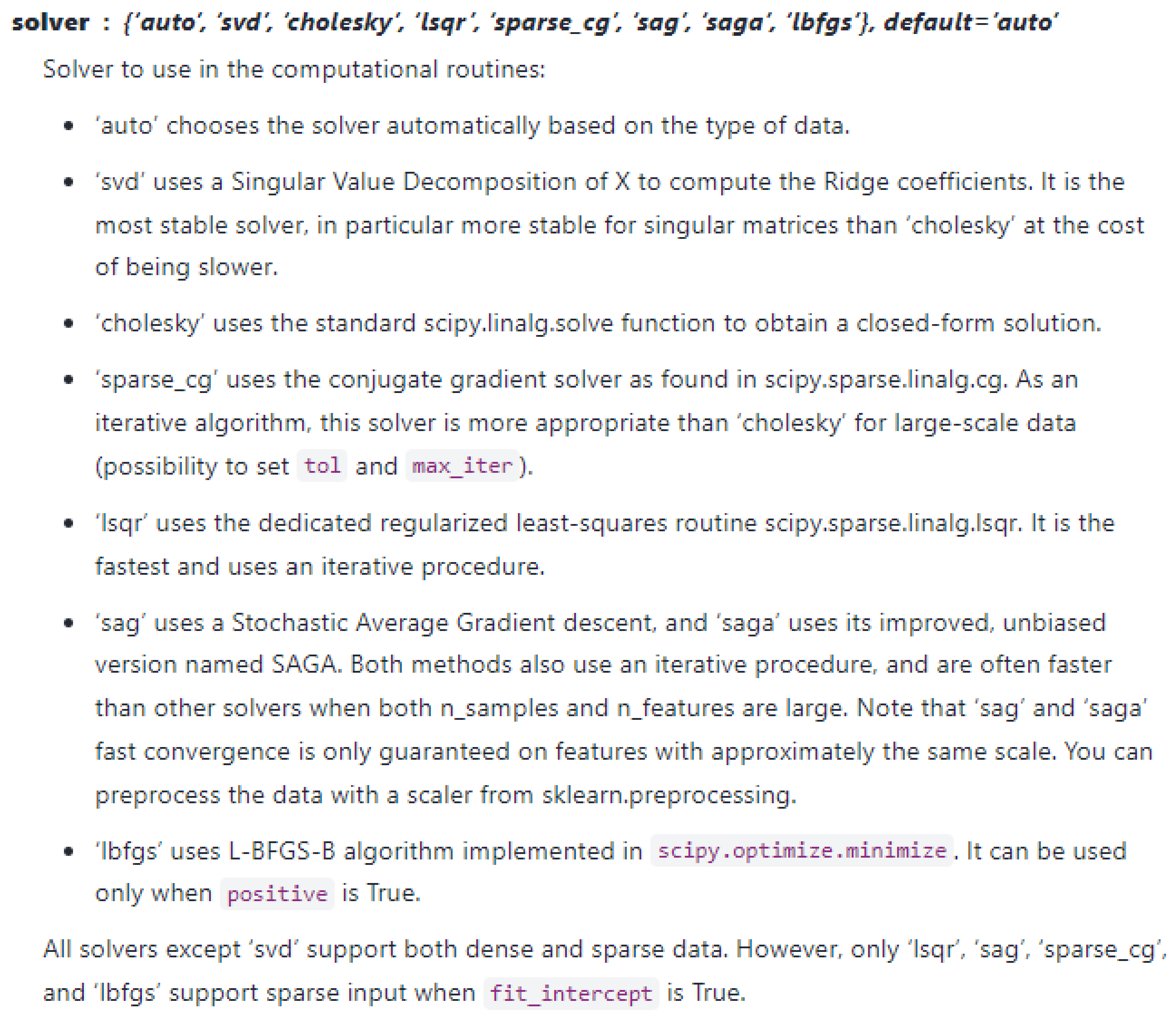
max_iter : ***int, default=None***

Maximum number of iterations for conjugate gradient solver. For 'sparse_cg' and 'lsqr' solvers, the default value is determined by scipy.sparse.linalg. For 'sag' solver, the default value is 1000. For 'lbfgs' solver, the default value is 15000.

tol : ***float, default=1e-4***

The precision of the solution (`coef_`) is determined by `tol` which specifies a different convergence criterion for each solver:

- 'svd': `tol` has no impact.
- 'cholesky': `tol` has no impact.
- 'sparse_cg': norm of residuals smaller than `tol`.
- 'lsqr': `tol` is set as atol and btol of scipy.sparse.linalg.lsqr, which control the norm of the residual vector in terms of the norms of matrix and coefficients.
- 'sag' and 'saga': relative change of coef smaller than `tol`.
- 'lbfgs': maximum of the absolute (projected) gradient=max|residuals| smaller than `tol`.

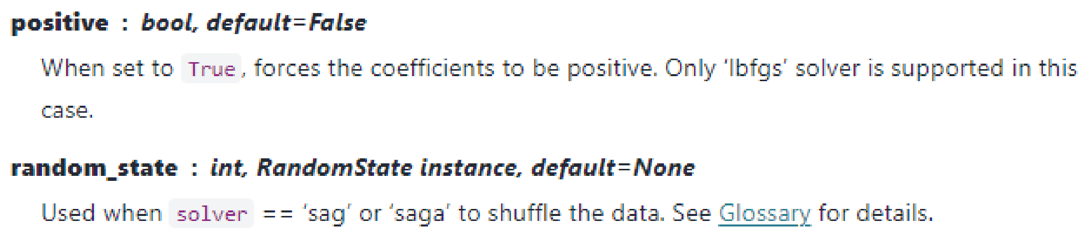


solver : *{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga', 'lbfgs'}, default='auto'*

Solver to use in the computational routines:

- 'auto' chooses the solver automatically based on the type of data.
- 'svd' uses a Singular Value Decomposition of X to compute the Ridge coefficients. It is the most stable solver, in particular more stable for singular matrices than 'cholesky' at the cost of being slower.
- 'cholesky' uses the standard scipy.linalg.solve function to obtain a closed-form solution.
- 'sparse_cg' uses the conjugate gradient solver as found in scipy.sparse.linalg.cg. As an iterative algorithm, this solver is more appropriate than 'cholesky' for large-scale data (possibility to set `tol` and `max_iter`).
- 'lsqr' uses the dedicated regularized least-squares routine scipy.sparse.linalg.lsqr. It is the fastest and uses an iterative procedure.
- 'sag' uses a Stochastic Average Gradient descent, and 'saga' uses its improved, unbiased version named SAGA. Both methods also use an iterative procedure, and are often faster than other solvers when both n_samples and n_features are large. Note that 'sag' and 'saga' fast convergence is only guaranteed on features with approximately the same scale. You can preprocess the data with a scaler from sklearn.preprocessing.
- 'lbfgs' uses L-BFGS-B algorithm implemented in `scipy.optimize.minimize`. It can be used only when `positive` is True.

All solvers except 'svd' support both dense and sparse data. However, only 'lsqr', 'sag', 'sparse_cg', and 'lbfgs' support sparse input when `fit_intercept` is True.

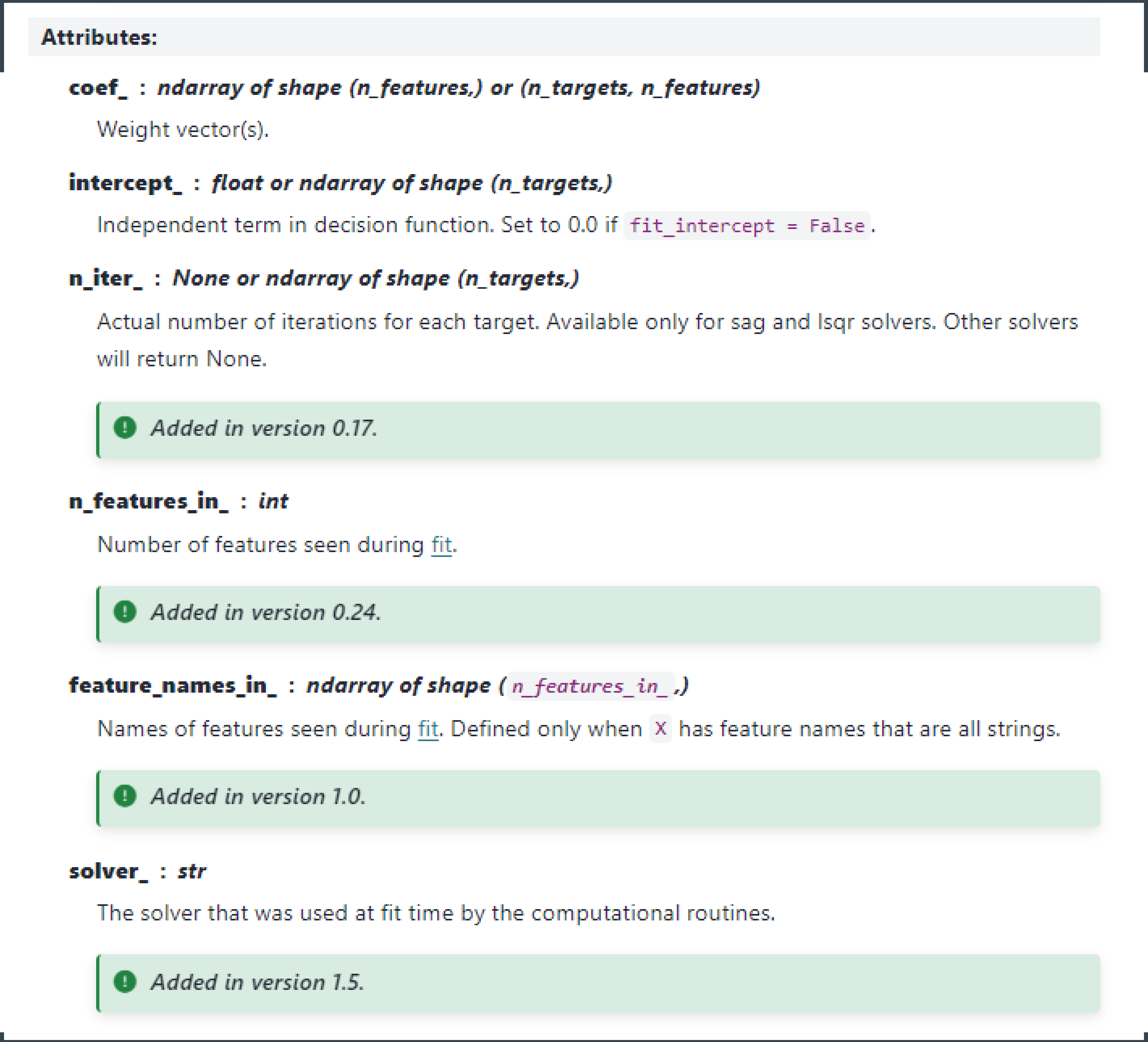


positive : *bool, default=False*

When set to `True`, forces the coefficients to be positive. Only 'lbfgs' solver is supported in this case.

random_state : *int, RandomState instance, default=None*

Used when `solver` == 'sag' or 'saga' to shuffle the data. See Glossary for details.



Attributes:

coef_ : ***ndarray of shape (n_features,) or (n_targets, n_features)***

Weight vector(s).

intercept_ : ***float or ndarray of shape (n_targets,)***

Independent term in decision function. Set to 0.0 if `fit_intercept = False`.

n_iter_ : ***None or ndarray of shape (n_targets,)***

Actual number of iterations for each target. Available only for sag and lsqr solvers. Other solvers will return None.

> *Added in version 0.17.*

n_features_in_ : ***int***

Number of features seen during fit.

> *Added in version 0.24.*

feature_names_in_ : ***ndarray of shape (`n_features_in_`,)***

Names of features seen during fit. Defined only when `X` has feature names that are all strings.

> *Added in version 1.0.*

solver_ : ***str***

The solver that was used at fit time by the computational routines.

> *Added in version 1.5.*

Q
&
A