1) Linear Regression

class sklearn.linear_model.LinearRegression(*, fit_intercept=True, normalize=False, copy_X=True, n_jobs=None, positive=False)

```
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [3, 3]])
# y = 3 * x_0 + 2 * x_1 + 4
y = np.dot(X, np.array([3, 2])) + 4
model=LinearRegression().fit(X,y)
model.score(X,y)
```

```
1.0
```

```
model.coef_,model.intercept_
```

```
(array([3., 2.]), 4.0)
```

```
model.predict(np.array([[3,5]]))
```

```
array([23.])
```

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*
- **Normalize : bool, default=False** - *If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.*
- **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 for n_targets > 1 and sufficient large problems.*
- **Positive : bool, default=False** – *When set to True, forces the coefficients to be positive. This option is only available for dense arrays.*

Attributes:

- **coef_ : array of shape (n_features,) or (n_targets, n_features)** - *Estimated coefficients for the linear regression problem.*
- **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.*

2) Ridge Regression (L2)

class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, normalize=False, copy_X=True, max_iter=None, tol=0.001, solver='auto', random_state=None)

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

```
from sklearn.linear_model import Ridge
import numpy as np
n_samples, n_features = 10, 5
rng = np.random.RandomState(0)
y = rng.randn(n_samples)
X = rng.randn(n_samples, n_features)
model = Ridge(alpha=1.0)
model.fit(X,y)
```

```
Ridge()
```

```
model.fit(X,y).score(X,y)
```

```
0.6836781050289736
```

Parameters:

- **alpha : {float, ndarray of shape (n_targets,)}, default=1.0** - *Regularization strength; must be a positive float. Regularization improves the conditioning of the problem and reduces the variance of the estimates.*
- **fit_intercept : bool, default=True** - *Whether to fit the intercept for this model. If set to false, no intercept will be used in calculations.*
- **normalize : bool, default=False** - *This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.*
- **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.*
- **tol : float, default=1e-3** - *Precision of the solution.*
- **solver : {'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'}, default='auto'** - *Solver to use in the computational routines.*
- **random_state : int, RandomState instance, default=None** – *Used to shuffle the data.*

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.*
- **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.*

3) Lasso Regression (L1)

class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')

Technically the Lasso model is optimizing the same objective function as the Elastic Net with l1_ratio=1.0 (no L2 penalty).

```
from sklearn import linear_model
model = linear_model.Lasso(alpha=0.1)
model.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
print(model.coef_)
```

```
[0.85 0.  ]
```

```
print(model.intercept_)
```

```
0.15000000000000002
```

Parameters:

- **alpha : float, default=1.0** - *Constant that multiplies the L1 term.*
- **fit_intercept : bool, default=True** - *Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations.*
- **normalize : bool, default=False** - *This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.*
- **precompute : bool or array-like of shape (n_features, n_features), default=False** - *Whether to use a precomputed Gram matrix to speed up calculations. The Gram matrix can also be passed as argument.*
- **copy_X : bool, default=True** - *If True, X will be copied; else, it may be overwritten.*
- **max_iter : int, default=1000** - *The maximum number of iterations.*
- **tol : float, default=1e-4** - *The tolerance for the optimization: if the updates are smaller than tol, the optimization code checks the dual gap for optimality and continues until it is smaller than tol.*
- **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.*
- **positive : bool, default=False** - *When set to True, forces the coefficients to be positive.*
- **random_state : int, RandomState instance, default=None** - *The seed of the pseudo random number generator that selects a random feature to update.*
- **selection : {'cyclic', 'random'}, default='cyclic'** - *If set to 'random', a random coefficient is updated every iteration rather than looping over features sequentially by default.*

Attributes:

- **coef_ : ndarray of shape (n_features,) or (n_targets, n_features)** - *Parameter vector (w in the cost function formula).*
- **dual_gap_ : float or ndarray of shape (n_targets,)** - *Given param alpha, the dual gaps at the end of the optimization, same shape as each observation of y.*
- **sparse_coef_ : sparse matrix of shape (n_features, 1) or (n_targets, n_features)** - *Sparse representation of the fitted coef_.*
- **intercept_ : float or ndarray of shape (n_targets,)** - *Independent term in decision function.*
- **n_iter_ : int or list of int** - *Number of iterations run by the coordinate descent solver to reach the specified tolerance.*

4) Logistic Regression

class sklearn.linear_model.LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
X, y = load_iris(return_X_y=True)
model = LogisticRegression(max_iter=200,random_state=0).fit(X, y)
model.predict(X[:2, :])
```

```
array([0, 0])
```

```
model.predict_proba(X[:2, :])
```

```
array([[9.81578647e-01, 1.84213388e-02, 1.44845926e-08],
       [9.71330819e-01, 2.86691504e-02, 3.01593572e-08]])
```

```
model.score(X, y)
```

```
0.9733333333333334
```

Parameters:

- **penalty : {'l1', 'l2', 'elasticnet', 'none'}, default='l2'** - *Used to specify the norm used in the penalization.*
- **dual : bool, default=False** - *Dual or primal formulation.*
- **tol : float, default=1e-4** - *Tolerance for stopping criteria.*
- **C : float, default=1.0** - *Inverse of regularization strength; must be a positive float.*
- **fit_intercept : bool, default=True** - *Specifies if a constant (a.k.a. bias or intercept) should be added to the decision function.*
- **intercept_scaling : float, default=1** - *Useful only when the solver 'liblinear' is used and self.fit_intercept is set to True.*
- **class_weight : dict or 'balanced', default=None** - *Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.*
- **random_state : int, RandomState instance, default=None** - *Used when solver == 'sag', 'saga' or 'liblinear' to shuffle the data.*
- **solver : {'newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'}, default='lbfgs'** - *Algorithm to use in the optimization problem.*
- **max_iter : int, default=100** - *Maximum number of iterations taken for the solvers to converge.*
- **multi_class : {'auto', 'ovr', 'multinomial'}, default='auto'** - *If the option chosen is 'ovr', then a binary problem is fit for each label. For 'multinomial' the loss minimised is the multinomial loss fit across the entire probability distribution, even when the data is binary. 'multinomial' is unavailable when solver='liblinear'. 'auto' selects 'ovr' if the data is binary, or if solver='liblinear', and otherwise selects 'multinomial'.*
- **verbose : int, default=0** - *For the liblinear and lbfgs solvers set verbose 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, default=None** - *Number of CPU cores used when parallelizing over classes if multi_class='ovr'".*
- **l1_ratio : float, default=None** - *The Elastic-Net mixing parameter.*

Attributes:

- **classes_ : ndarray of shape (n_classes,)** - *A list of class labels known to the classifier.*
- **coef_ : ndarray of shape (1, n_features) or (n_classes, n_features)** - *Coefficient of the features in the decision function.*
- **intercept_ : ndarray of shape (1,) or (n_classes,)** - *Intercept (a.k.a. bias) added to the decision function.*
- **n_iter_ : ndarray of shape (n_classes,) or (1,)** - *Actual number of iterations for all classes. If binary or multinomial, it returns only 1 element. For liblinear solver, only the maximum number of iteration across all classes is given.*