Regression models

1) Simple Linear Regression:

Linear regression assumes a linear or straight line relationship between the input variables (X) and the single output variable (y).

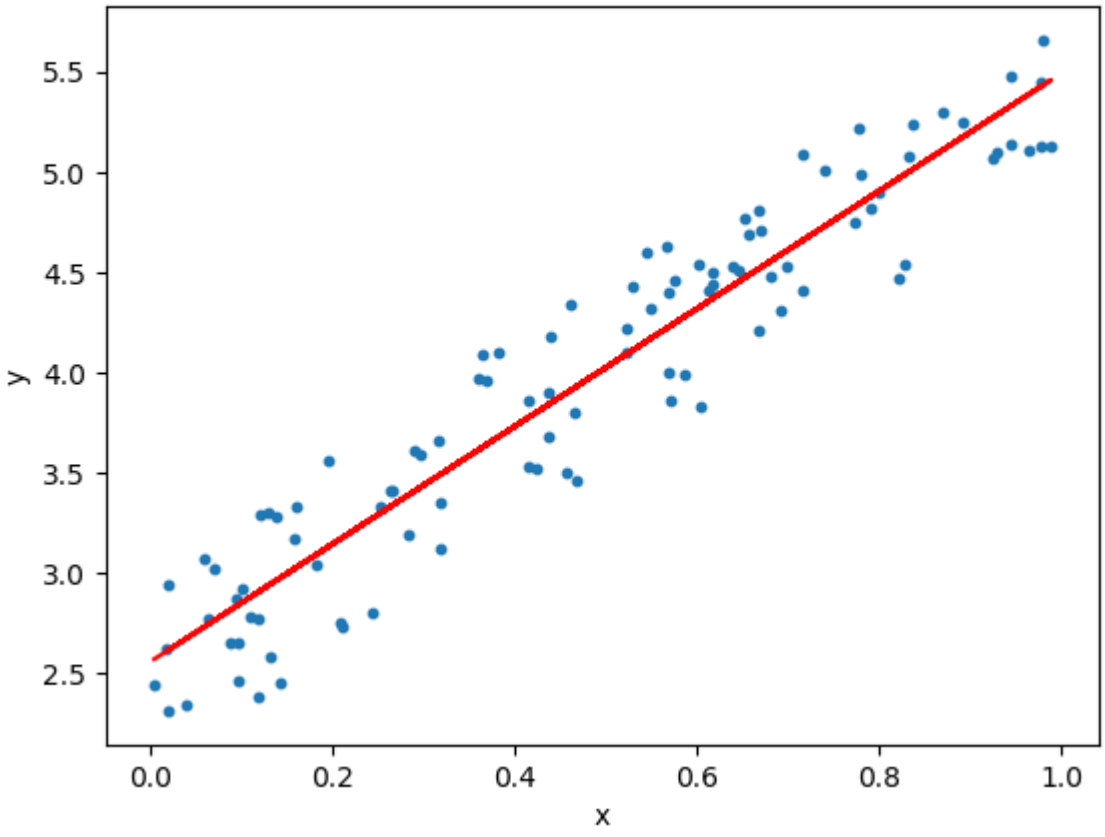
More specifically, that output (y) can be calculated from a linear combination of the input variables (X). When there is a single input variable, the method is referred to as a simple linear regression.

In simple linear regression we can use statistics on the training data to estimate the coefficients required by the model to make predictions on new data.

The line for a simple linear regression model can be written as:

y = b0 + b1 * x

b0, b1 – coefficients we must estimate from the training data.



Syntax:

From sklearn.linear_model import LinearRegression

X=input variable

Y=target variable

model = LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=None).fit(X, y)

model.score(X, y)

Parameters:

- **fit_intercept** : boolean, optional, default True

Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered).

- **normalize** : boolean, optional, 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. If you wish to standardize, please use sklearn.preprocessing.StandardScaler before calling fit on an estimator with normalize=False.

- **copy_X** : boolean, optional, default True

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

- **n_jobs** : int or None, optional (default=None)

The number of jobs to use for the computation. This will only provide speedup for n_targets > 1 and sufficient large problems. None means 1 unless in a joblib.parallel_backend context. -1 means using all processors.

Attributes:

- **coef_** : array, 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.

- **intercept_** : array

  Independent term in the linear model.

Methods:

fit(X, y[, sample_weight]) - Fit linear model.

get_params([deep] - Get parameters for this estimator.

predict(X) - Predict using the linear model

score(X, y[, sample_weight]) - Returns the coefficient of determination R^2 of the prediction.

Example:

```
import numpy as np
from sklearn.linear_model import LinearRegression
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X, y)
reg.score(X, y) - 1.0
reg.coef_ - array([1., 2.])
reg.intercept_ - 3.0000...
reg.predict(np.array([[3, 5]])) - array([16.])
```

2) Multiple Linear Regression:

Multiple Linear Regression is a simple and common way to analyze linear regression. The model is often used for predictive analysis since it defines the relationship between two or more variables.

Multiple linear regression used to be a lot easier, only taking a couple of variables into consideration. Nowadays, we need to take a large variety of variables into consideration, and we have to decide which variables to use opposed to which variables to throw out.

y = b0 + b1 * x1+b2*x2+……..bn*xn

Regression performance Metrics

1) Mean Absolute Error:

The Mean Absolute Error (MAE) is the average of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were.

The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting)

Syntax:

from sklearn.metrics import mean_absolute_error

y_true = Ground truth (correct) target values.

y_pred = Estimated target values.

mean_absolute_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')

Parameters:

sample_weight : array-like of shape = (n_samples), optional - Sample weights.

multioutput : string in ['raw_values', 'uniform_average'] - or array-like of shape (n_outputs) Defines aggregating of multiple output values. Array-like value defines weights used to average errors.

'raw_values' : Returns a full set of errors in case of multioutput input.

'uniform_average' : Errors of all outputs are averaged with uniform weight.

Example:

from sklearn.metrics import mean_absolute_error

y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

mean_absolute_error(y_true, y_pred) - 0.5

y_true = [[0.5, 1], [-1, 1], [7, -6]]

y_pred = [[0, 2], [-1, 2], [8, -5]]

mean_absolute_error(y_true, y_pred) - 0.75

mean_absolute_error(y_true, y_pred, multioutput='raw_values') - array([0.5, 1.])

mean_absolute_error(y_true, y_pred, multioutput=[0.3, 0.7]) - 0.85

2) Mean Squared error:

The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error.

Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation. This is called the Root Mean Squared Error (or RMSE).

Syntax:

from sklearn.metrics import mean_squared_error

y_true = Ground truth (correct) target values.

y_pred = Estimated target values.

mean_squared_error(y_true, y_pred, sample_weight=None, multioutput='uniform_average')

Example:

from sklearn.metrics import mean_squared_error

y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

mean_squared_error(y_true, y_pred) - 0.375

y_true = [[0.5, 1],[-1, 1],[7, -6]]

y_pred = [[0, 2],[-1, 2],[8, -5]]

mean_squared_error(y_true, y_pred) - 0.708

mean_squared_error(y_true, y_pred, multioutput='raw_values') - array([0.41666667, 1.])

mean_squared_error(y_true, y_pred, multioutput=[0.3, 0.7]) - 0.825...

3) R Squared Metric:

The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature, this measure is called the **coefficient of determination.**

This is a value between 0 - no-fit and 1 - perfect fit.

Syntax:

from sklearn.metrics import r2_score

y_true = Ground truth (correct) target values.

y_pred = Estimated target values.

r2_score(y_true, y_pred, sample_weight=None, multioutput='uniform_average')

Example:

from sklearn.metrics import r2_score

y_true = [3, -0.5, 2, 7]

y_pred = [2.5, 0.0, 2, 8]

r2_score(y_true, y_pred) - 0.948...

y_true = [[0.5, 1], [-1, 1], [7, -6]]

y_pred = [[0, 2], [-1, 2], [8, -5]]

r2_score(y_true, y_pred,multioutput='variance_weighted') - 0.938...

y_true = [1, 2, 3]

y_pred = [1, 2, 3]

r2_score(y_true, y_pred) - 1.0

y_true = [1, 2, 3]

y_pred = [2, 2, 2]

r2_score(y_true, y_pred) - 0.0

y_true = [1, 2, 3]

y_pred = [3, 2, 1]

r2_score(y_true, y_pred) - -3.0