Data prep!

Data → Categorical → Nominal [เปรียบเทียบไม่ได้]
Data → Categorical → Ordinal [เปรียบเทียบได้]
Data → Numerical

→ df.info() : ดูพวก type
→ df.describe() : ดูค่าทาง Statistic
→ ดูว่าข้อมูล valid มั้ย เช่น อายุ ไม่ควรติด ⊖, ไม่เกิน 100 ปี ...

* drop ตัวแปรที่ไม่เกี่ยวข้องกัน เช่น National ID ไม่เกี่ยวข้องกับการตัดสินใจ *ต้องทำ
* ตัวที่มี missing value >50% [Unqualify data]
* ตัวแปรประเภท Categorical → ไม่ควร unique เกินไป
↳ ไม่ควร flat เกินไป

Preparing feature for ML

① Impute missing values [ห้าม impute target ⇒ ต้อง remove ทิ้ง]
- numerical variable ⇒ Mean, Median
- Categorical variable ⇒ Mode (most-freq.)

② Categorical to numeric variable

→ One-hot vector [dummy code] *n-1 แถว drop

```
pd.get_dummies(df, column=['Sex', 'Embarked'], drop_first=True)
```

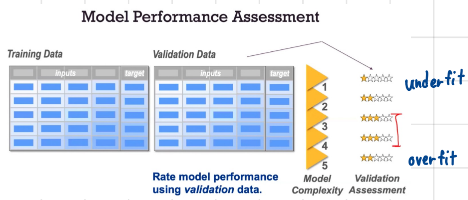
③ เอา Outlier ออก
→ ตัวที่ percentile น้อย-มากที่สุด 1%
→ $\mu \pm 3$ S.D.

④ แก้การเบ้ของข้อมูล ⇒ log, Binning

แยก Test/Train/Validate

- training data ⇒ model construction
- testing data ⇒ model evaluation
- Validating data ⇒ model tuning

*ต้อง stratify ด้วย



Feature Selection based-on Statistics

1.13.2. Univariate feature selection ¶ ⇒ ใช้เทียบ target ↔ ตัวแปรต้นแต่ละตัว

48

Univariate feature selection works by selecting the best features based on univariate statistical tests. It can be seen as a preprocessing step to an estimator. Scikit-learn exposes feature selection routines as objects that implement the `transform` method:

- `SelectKBest` removes all but the k highest scoring features
- `SelectPercentile` removes all but a user-specified highest scoring percentage of features
- using common univariate statistical tests for each feature: false positive rate `SelectFpr`, false discovery rate `SelectFdr`, or family wise error `SelectFwe`.
- `GenericUnivariateSelect` allows to perform univariate feature selection with a configurable strategy. This allows to select the best univariate selection strategy with hyper-parameter search estimator.

For instance, we can perform a χ^2 test to the samples to retrieve only the two best features as follows:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.feature_selection import chi2
>>> X, y = load_iris(return_X_y=True)
>>> X.shape
(150, 4)
>>> X_new = SelectKBest(chi2, k=2).fit_transform(X, y)  # เอา X มา 2 ตัวที่ได้ค่า chi² สูงสุดสำหรับ target y
>>> X_new.shape
(150, 2)
```

(เรียงตาม chi-square)

https://scikit-learn.org/stable/modules/feature_selection.html

These objects take as input a scoring function that returns univariate scores and p-values (or only scores for `SelectKBest` and `SelectPercentile`):

- For regression: `f_regression` (t-test), `mutual_info_regression`, r_regression (correlation) ⇐ สำหรับค่าที่เป็นตัวเลข
- For classification: `chi2`, `f_classif`, `mutual_info_classif`

based-on Model

1.13.4. Feature selection using SelectFromModel

49

`SelectFromModel` is a meta-transformer that can be used alongside any estimator that assigns importance to each feature through a specific attribute (such as `coef_`, `feature_importances_`) or via an `importance_getter` callable after fitting. The features are considered unimportant and removed if the corresponding importance of the feature values are below the provided `threshold` parameter. Apart from specifying the threshold numerically, there are built-in heuristics for finding a threshold using a string argument. Available heuristics are "mean", "median" and float multiples of these like "0.1*mean". In combination with the `threshold` criteria, one can use the `max_features` parameter to set a limit on the number of features to select.

For examples on how it is to be used refer to the sections below.

Examples

- Model-based and sequential feature selection

1.13.4.1. L1-based feature selection

Linear models penalized with the L1 norm have sparse solutions: many of their estimated coefficients are zero. When the goal is to reduce the dimensionality of the data to use with another classifier, they can be used along with `SelectFromModel` to select the non-zero coefficients. In particular, sparse estimators useful for this purpose are the `Lasso` for regression, and of `LogisticRegression` and `LinearSVC` for classification:

```
>>> from sklearn.svm import LinearSVC
>>> from sklearn.datasets import load_iris
>>> from sklearn.feature_selection import SelectFromModel
>>> X, y = load_iris(return_X_y=True)
>>> X.shape
(150, 4)
>>> lsvc = LinearSVC(C=0.01, penalty="l1", dual=False).fit(X, y)
>>> model = SelectFromModel(lsvc, prefit=True)
>>> X_new = model.transform(X)
>>> X_new.shape
(150, 3)
```

ML

scikit-learn
Machine Learning in Python

Getting Started | What's New in 0.22.2 | GitHub

- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification Category
Identifying which category an object belongs to.
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...
Examples

Regression Numerical
Predicting a continuous-valued attribute associated with an object.
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...
Examples

Clustering
Automatic grouping of similar objects into sets.
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...
Examples

Dimensionality reduction
Reducing the number of random variables to consider.
Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...
Examples

Model selection
Comparing, validating and choosing parameters and models.
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...
Examples

Preprocessing
Feature extraction and normalization.
Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...
Examples

- Superisor ทำนาย ⇒ มี target
- Unsuperisor หา insight ⇒ ไม่มี target
- RL

CHULA ENGINEERING COMPUTER

Estimator Interface

Decision Trees

We'll start just by training a single decision tree.

```
In [8]: from sklearn.tree import DecisionTreeClassifier
In [9]: dtree = DecisionTreeClassifier(min_samples_leaf=10, criterion='entropy')
In [10]: dtree.fit(X_train,y_train)   ⇒ เรียนเสร็จแล้ว
Out[10]: DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=10, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

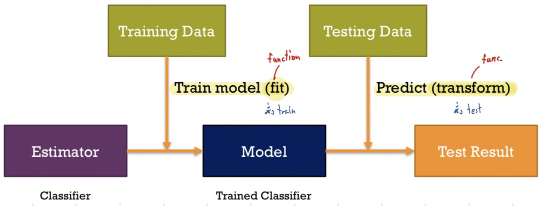
Prediction and Evaluation

Let's evaluate our decision tree.

```
[11]: predictions = dtree.predict(X_test)   ⇒ เอามา Prediction
[12]: from sklearn.metrics import classification_report,confusion_matrix
[13]: print(classification_report(y_test,predictions))
             precision    recall  f1-score   support

     absent       0.85      0.85      0.85        20
    present       0.40      0.40      0.40         5

avg / total       0.76      0.76      0.76        25
```



Training Phase

22

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()    # Get some data
X_train, X_test, y_train, y_test = train_test_split(
        cancer.data, cancer.target,
        stratify=cancer.target, random_state=1337)

tree = DecisionTreeClassifier(random_state=7331)
tree.fit(X_train, y_train)   # Learn a Decision Function
```

Input, target

Ref: C. Travis Johnson, Mines Linux Users Group on April 27, 2017

CHULA ENGINEERING COMPUTER

Testing Phase (Prediction)

23

Prediction, input

```
y_pred = tree.predict(X_test)
```

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
              precision    recall  f1-score   support

     class 0       0.50      1.00      0.67         1
     class 1       0.00      0.00      0.00         1
     class 2       1.00      0.67      0.80         3

   micro avg       0.60      0.60      0.60         5
   macro avg       0.50      0.56      0.49         5
weighted avg       0.70      0.60      0.61         5
```

เฉลย, Predicted

เนี่ยไรไม่รู้แปะไว้ก่อน

CHULA ENGINEERING COMPUTER

https://www.datatechnotes.com/2021/04/selectfrommodel-feature-selection.html

77

#ออกสอบได้

```
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import AdaBoostRegressor
from sklearn.datasets import load_boston
from numpy import array

boston = load_boston()
x = boston.data
y = boston.target

print("Feature data dimension: ", x.shape)

estimator = AdaBoostRegressor(random_state=0, n_estimators=50
selector = SelectFromModel(estimator)
selector = selector.fit(x, y)

status = selector.get_support()
print("Selection status: ", status)

features = array(boston.feature_names)
print("All features:")
print(features)

print("Selected features:")
print(features[status])
selector.transform(x)
```

ใส่ Threshold ด้วยได้

```
status = selector.get_support()
print("Selection status: ", status)

Selection status:  [False False False False False  True False  True False False False False
  True]

features = array(boston.feature_names)
print("All features:")
print(features)

print("Selected features:")
print(features[filter])
selector.transform(x)

All features:
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']

Selected features:
['RM' 'DIS' 'LSTAT']
array([[6.575 , 4.09  , 4.98  ],
       [6.421 , 4.9671, 9.14  ],
       [7.185 , 4.9671, 4.03  ],
       ...,
       [6.976 , 2.1675, 5.64  ],
       [6.794 , 2.3889, 6.48  ],
       [6.03  , 2.505 , 7.88  ]])
```

Decision Tree Classifier ⇒ ใช้กับโจทย์ Classification

```
select_features = 'age' #@param ['age']
target_column = 'SeriousDlqin2yrs' #@param ['Serio

select_features = [select_features]
# Setup Features & Target
features = train_df[select_features]
target = train_df[target_col]

# Setup Parameters
Citerion = 'Entropy' #@param ['Entropy', 'Gini']
max_depth = 1
dtree = tree.DecisionTreeClassifier(
    criterion=Citerion.lower(),
    max_depth=max_depth,
)

# Training you tree
dtree.fit(features, target)
```

select_features: age
target_column: SeriousDlqin2yrs
Citerion: Entropy (Entropy / Gini)

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=1, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

หลังจากได้ Tree มาแล้ว ⇒ dtree.features_importances_

[ดูความสำคัญของตัวแปร]

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *{"gini", "entropy"}, default="gini"*
The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *{"best", "random"}, default="best"*
The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*
The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leav contain less than min_samples_split samples.

min_samples_split : *int or float, default=2*
The minimum number of samples required to split an internal node:

Hyper param.

Important Parameters in Decision Tree

- Splitting measure (criterion) : gini / entropy
- Maximum depth : ~5-10 (depend on number of feature)
- Maximum leaf nodes : depend on number of class (target) and feature
- Minimum sample split : 5 – 20% (depend on number of data)
- Minimum impurity decrease : (default 0)
- Pruning adjustment (ccp_alpha) : 0.001 - 0.01 (depend on size of tree)

* Stop การแตก Leaf node เมื่อถึง Max depth
* Minimum leaf side

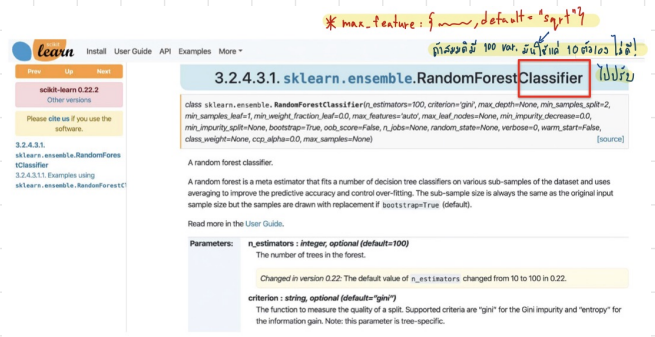
Random Forest [Strong base line!!] ⇒ ใช้ทั้ง Classifier, Regressor

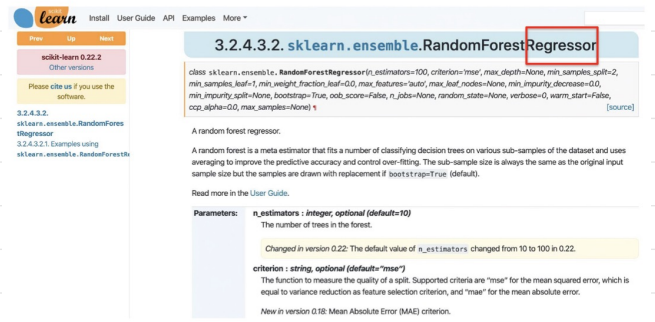
Importance Params: • # Tree
Columns (Variables)
Rows (examples)

* max_feature : ∫ ~~, default = "sqrt"

ค่าเริ่มต้นมี 100 var. มันใช้แค่ 10 ตัวเอง ไม่ดี!

⇒ max_features ปรับให้มันดี
เช่นมี 100 features ก็ควรอยู่ประมาณ 50-80
อย่าใช้ default = "sqrt"

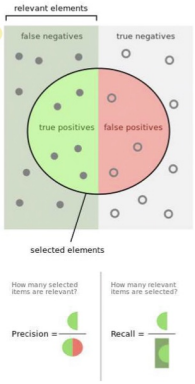




Conclusion Matrix

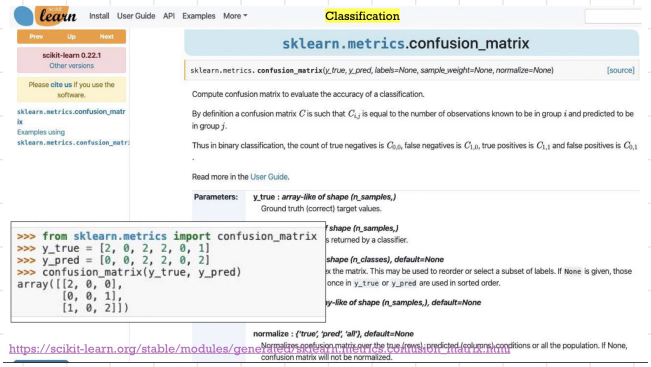
Precision, Recall, F1 // Confusion Matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	True Negative TN = 50	False Positive FP = 10	60
Actual: YES	False Negative FN = 5	True Positive TP = 100	105
	55	110	

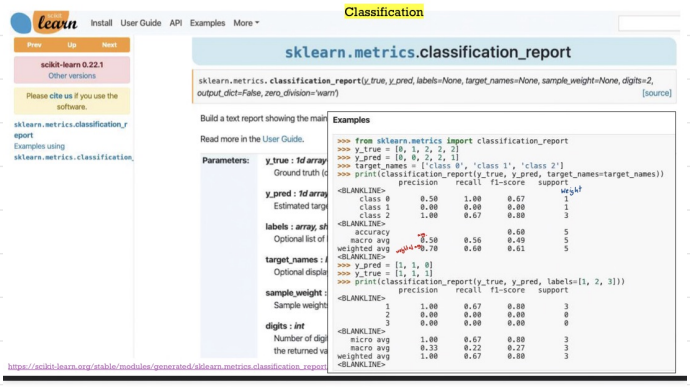


- Precision = correctly predict = TP / (TP + FP)
- Recall = coverage = TP / (TP + FN)
- F1 = (2*pre*rec) / (pre + rec)

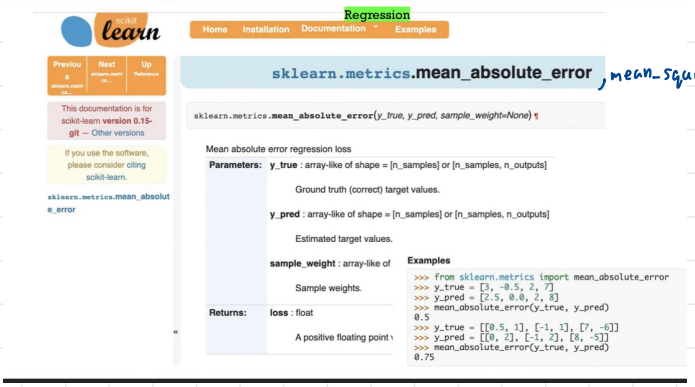
Accuracy = TP+TF



https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html



https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report



, mean_square_error

Linear Regression → ต้องทำ Lasso (L1), Ridge (L2)

Feature Selection

CHULA ENGINEERING COMPUTER https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

learn Install User Guide API Examples More

Prev Up Next

scikit-learn 0.22.1
Other versions

Please cite us if you use the software.

sklearn.linear_model.LinearRegression

Examples using sklearn.linear_model.LinearRegression

sklearn.linear_model.LinearRegression

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

normalize : bool, 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.

normalize : bool, default=False
This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normali
before regression by subtracting the mean and dividing by the l2-norm. If you wish to standardize, plea
use StandardScaler before calling fit on an estimator with normalize=False.

Deprecated since version 1.0: normalize was deprecated in version 1.0 and will be removed in 1.2.

ตัวอย่าง coefficient ต้าง

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), only one target is passed, this is a 1D array of

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.
New 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.
New in version 1.0.

for n_targets > 1 and sufficient
ine means 1 unless in a joblib.parallel_backend context. -1 means using all processors.
ore details.

ใช้ตัวแปรนี้เลยอย่างเดียว?

```
estimator = Lasso()
selector = SelectFromModel(estimator)

estimator = Ridge()
selector = SelectFromModel(estimator, threshold = ?!?)
```

sklearn.linear_model.Ridge L2

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

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 l2-norm. Also known as Ridge Regression or Tikhonov 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,)}
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 Lasso 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. data is expected to be centered).

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. If you wish to standardize, please use StandardScaler before calling fit on an estimator with normalize=False.

sklearn.linear_model.Lasso L1

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

Linear Model trained with L1 prior as regularizer (aka the Lasso).

The optimization objective for Lasso is:

```
(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1
```

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

Read more in the User Guide.

Parameters::

alpha : float, default=1.0
Constant that multiplies the L1 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 Lasso object is not advised. Instead, you should use the LinearRegression object.

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).

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. If you wish to standardize, please use StandardScaler before calling fit on an estimator with normalize=False.

Deprecated since version 1.0: normalize was deprecated in version 1.0 and will be

sklearn.linear_model.ElasticNet L1 & L2

class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True, normalize='deprecated', precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic') [source]

Linear regression with combined L1 and L2 priors as regularizer.

Minimizes the objective function:

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

If you are interested in controlling the L1 and L2 penalty separately, keep in mind that this is equivalent to:

```
a * ||w||_1 + 0.5 * b * ||w||_2^2
```

where:

```
alpha = a + b and l1_ratio = a / (a + b)
```

The parameter l1_ratio corresponds to alpha in the glmnet R package while alpha corresponds to the lambda parameter in glmnet. Specifically, l1_ratio = 1 is the lasso penalty. Currently, l1_ratio <= 0.01 is not reliable, unless you supply your own sequence of alpha.

Read more in the User Guide.

Parameters::

alpha : float, default=1.0
Constant that multiplies the penalty terms. Defaults to 1.0. See the notes for the exact mathematical meaning of this parameter. alpha = 0 is equivalent to an ordinary least square, solved by the LinearRegression object. For numerical reasons, using alpha = 0 with the Lasso object is not advised. Given this, you should use the LinearRegression object.

l1_ratio : float, default=0.5
The ElasticNet mixing parameter, with 0 <= l1_ratio <= 1. For l1_ratio = 0 the penalty is an L2 penalty. For l1_ratio = 1 it is an L1 penalty. For 0 < l1_ratio < 1, the penalty is a combination of L1 and L2.

fit_intercept : bool, default=True
Whether the intercept should be estimated or not. If False, the data is assumed to be already centered.

normalize : bool, default=False