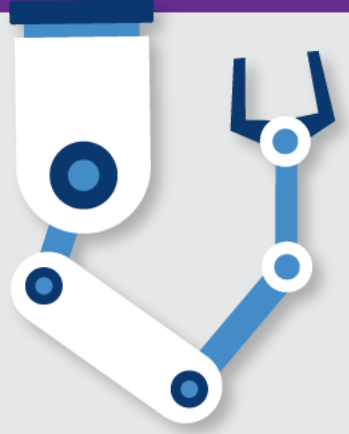
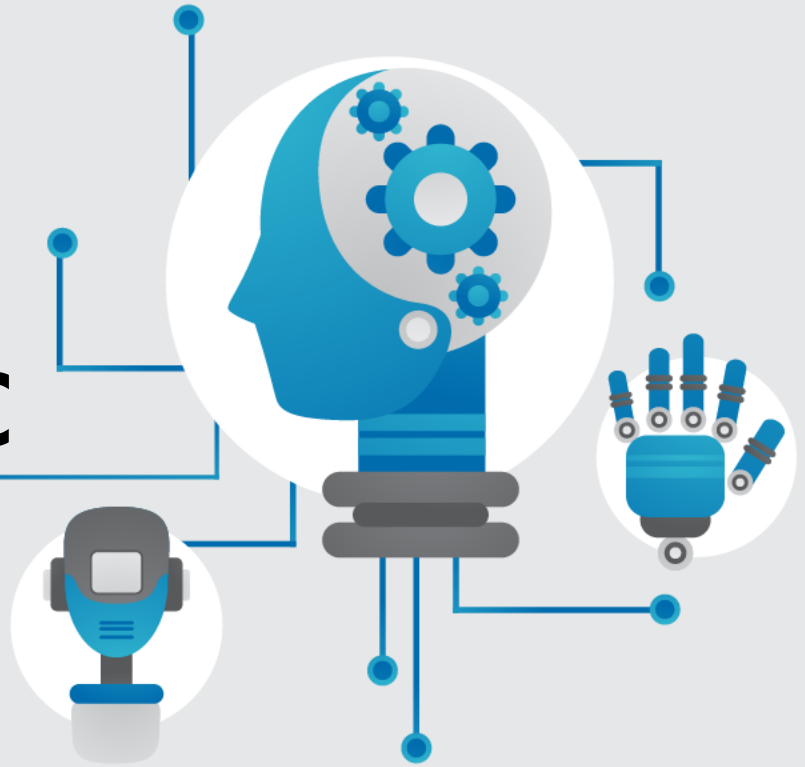


Scikit-learn Logistic Regression



Scikit - learn

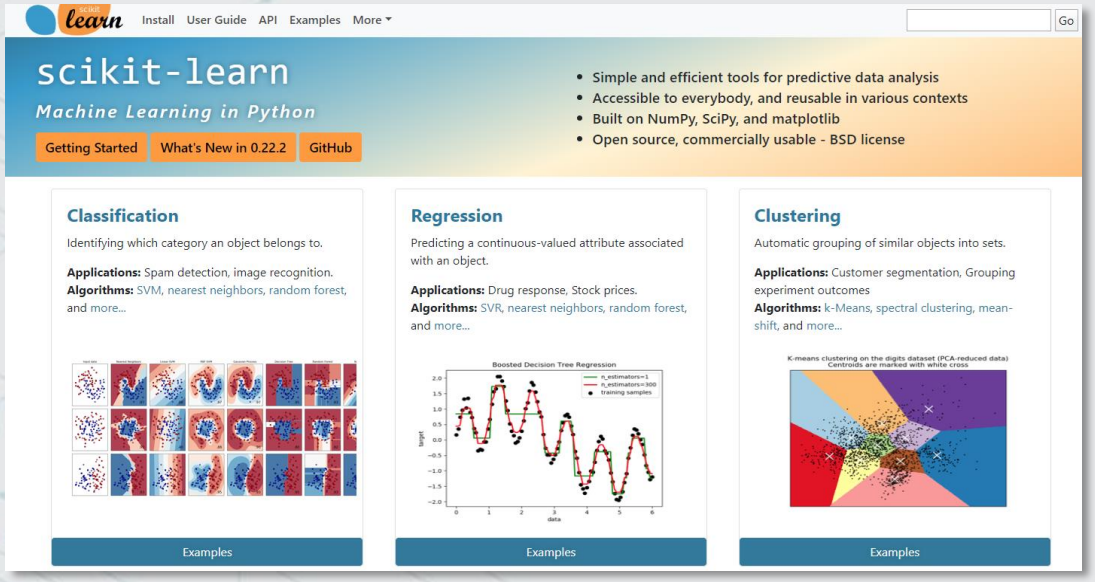
- Scikit-learn是python的免費機器學習套件，具有各種分類、迴歸和群集演算法，其中包含**羅吉斯迴歸**（Logistic Regression）、**隨機森林**（Random Forest）、**支援向量機**（Support Vector Machine）等，並以BSD授權條款授權發行，可以在商業和研究領域中免費使用。

Scikit - learn 下載與安裝

- 官方網站：https://scikit-learn.org/
- Python 安裝套件

  C:\> pip install scikit-learn
- Python 程式匯入套件

  import sklearn



Logistic Regression 介紹

機器學習實務

scikit learn Install User Guide API Examples More Go

Prev Up Next

scikit-learn 0.22.2
Other versions

Please **cite us** if you use the software.

sklearn.linear_model.LogisticRegression
Examples using
sklearn.linear_model.LogisticRegr

sklearn.linear_model.LogisticRegression

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*)
[source]

Logistic Regression (aka logit, MaxEnt) classifier.

In the multiclass case, the training algorithm uses the one-vs-rest (OvR) scheme if the 'multi_class' option is set to 'ovr', and uses the cross-entropy loss if the 'multi_class' option is set to 'multinomial'. (Currently the 'multinomial' option is supported only by the 'lbfgs', 'sag', 'saga' and 'newton-cg' solvers.)

This class implements regularized logistic regression using the 'liblinear' library, 'newton-cg', 'sag', 'saga' and 'lbfgs' solvers. **Note that regularization is applied by default**. It can handle both dense and sparse input. Use C-ordered arrays or CSR matrices containing 64-bit floats for optimal performance; any other input format will be converted (and copied).

Toggle Menu

Logistic Regression 參數說明

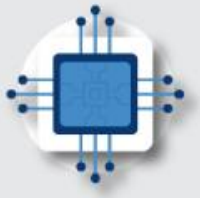
Logistic Regression 類別常用參數

- Penalty
- C
- class_weight
- solver
- multi_class
- n_jobs
- verbose

參數 penality

- penalty : {‘l1’, ‘l2’, ‘elasticnet’, ‘none’}, default= ‘l2’

  - ✓正規化（Regularization）時損失函數（loss）的懲罰規範。
  - ✓ ‘newton-cg’、 ‘sag’和‘lbfgs’ solvers 只支援‘l2’penalties。
  - ✓ ‘elasticnet’懲罰規範只被‘saga’ solver支援。
  - ✓ 選擇‘none’，則不進行正規化。



參數 C

- C : float, optional（default=1.0）

✓C是正規化（Regularization）參數，
正規化的強度與C成反比。

✓C必須是正值

✓C愈大，即對分錯樣本的懲罰程度愈大，
因此在訓練樣本中準確率愈高，但是
泛化能力降低，也就是對測試數據的
分類準確率降低。模型容易過度學習。

參數 class_weight

- class_weight : dict or ‘balanced’ , default=None

  - ✓類別的權重，以字典格式表示{class_label：weight}
  - ✓ 如果未設定，則所有類別的權重都設定為1
  - ✓ ‘balanced’模式使用y的值（label數量）
    來自動調整與輸入數據中的類別頻率成反比的權重：
    n_samples/(n_classes*np.bincount(y))

參數 Solver（1/2）

- solver : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=‘lbfgs’

  - ✓用於優化問題的演算法
  - ✓對於小型數據集，‘liblinear’是一個好的選擇。
  - ✓對於大型數據集，選擇‘sag’和‘saga’則會比較快。
  - ✓對於多類別問題，只有‘newton-cg’、‘sag’、‘saga’和‘lbfgs’能處理多項式損失函數；‘liblinear’僅限於ovr（one_vs_rest）

參數 Solver（2/2）

- solver : {‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}, default=‘lbfgs’

  - ✓‘newton-cg’、 ‘lbfgs’、‘sag’和‘saga’ 能處理L2或no penalty。
  - ✓‘liblinear’和‘saga’也可以處理L1 penalty
  - ✓‘saga’也支援elasticnet penalty
  - ✓‘liblinear’不支援none penalty

參數 multi_class

- multi_class : {‘auto’ , ‘ovr’, ‘multinomial’}, default=‘auto’

  - ✓如果選擇的選項是‘ovr’，需將標籤（label）調整為二元分類問題。
  - ✓當solver ='liblinear'時，'multinomial'不可用。
  - ✓如果數據是二元分類數據或者solver=‘liblinear’，則設定‘auto’時，會自動選擇‘ovr’；否則會自動選擇'multinomial'。

參數 n_jobs

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

  平行運算時所使用的處理器數量，預設為1
  如果設定-1，則使用全部的處理器

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

n_job=1

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 out of   1 | elapsed:    0.2s finished
```

n_job=-1

參數 verbose

- **verbose : int, optional（default=0）**
  用於開啟/關閉迭代中間輸出的日誌

```
from sklearn.linear_model import LogisticRegression
logisticRegression = LogisticRegression(verbose=0)
logisticRegression = logisticRegression.fit(X_train, y_train)
accuracy = logisticRegression.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7777777777777778
```

verbose=0

```
from sklearn.linear_model import LogisticRegression
logisticRegression = LogisticRegression(verbose=1)
logisticRegression = logisticRegression.fit(X_train, y_train)
accuracy = logisticRegression.score(X_test, y_test)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.7777777777777778
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s finished
```

verbose=1

Logistic Regression 函式說明

常用函式

- fit
- predict
- score

訓練（fit）

- 指令　fit(self, x, y, sample_weight=None)
- 參數
  - x : 訓練向量
  - y : 相對於x的目標向量
- 回傳：訓練後的logistic regression 模型物件
- 說明：根據給定的訓練數據訓練模型。
- 範例程式
  - from sklearn.linear_model import LogisticRegression
  - logisticRegression = LogisticRegression()
  - logisticRegression.fit(x_train, y_train)

預測（predict）

- 指令　predict(self, x)
- 參數
  - x : 輸入樣本
- 回傳：每個樣本的預測類別標籤
- 說明：預測x中樣本的類別標籤
- 範例程式

```
from sklearn.linear_model import LogisticRegression
logisticRegression = LogisticRegression()
logisticRegression.fit(x_train, y_train)
predictions = logisticRegression.predict(x_test)
```

評分（score）

- 指令　score(self, x, y, sample_weight=None)
- 參數
  - x : 測試樣本
  - y : 測試樣本的正確答案
- 回傳：測試樣本的平均準確度
- 說明：返回給定測試數據和標籤上的平均準確度。
- 範例程式
  - from sklearn.linear_model import LogisticRegression
  - logisticRegression = LogisticRegression()
  - logisticRegression.fit(x_train, y_train))
  - accuracy = logisticRegression.score(x_test, y_test)

程式範例（IRIS）

程式碼

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import datasets

# 載入資料
iris = datasets.load_iris()
X = iris.data[:, :2] # 只取前兩種特徵
Y = iris.target

# 建立 Logistic Regression Classifier
logreg = LogisticRegression(C=1e5)

# 進行訓練
logreg.fit(X, Y)

# 座標軸
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .02  # 單位間隔
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
```

程式範例（IRIS）

程式碼

```
# 進行預測
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# 繪製預測結果
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(4, 3))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

plt.scatter(X[:, 0], X[:, 1], c=Y, edgecolors='k',
cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```

程式範例（IRIS）

- 輸出結果