



**scikit-learn** 是一个开源的基于 python 语言的机器学习工具包。它通过 NumPy、SciPy 和 Matplotlib 等 python 数值计算的库实现高效的算法应用，并且涵盖了几乎所有主流机器学习算法。

sklearn 包含分析采集到的数据、根据数据特征选择适合的算法、在工具包中调用算法、调整算法的参数、获取需要的信息等机器学习算法应用全流程。

## Scikit-learn 示例

```
> from sklearn import neighbors, datasets, preprocessing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
iris = datasets.load_iris()
X, y = iris.data[:, :2], iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=33)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy_score(y_test, y_pred)
```

Out[1] : 0.631578947368421 # 输出 accuracy 指标得分

## 1. 加载数据

Scikit-learn 处理的数据是存储为 NumPy 数组或 SciPy 稀疏矩阵的数字，还支持 Pandas 数据框等可转换为数字数组的其它数据类型。

```
> import numpy as np
> X = np.random.random((10,5))
> y = np.array(['M', 'M', 'F', 'F', 'M', 'F', 'M', 'M', 'F', 'F'])
> X[X < 0.7] = 0
```

## 2. 训练 / 测试集切分

```
> from sklearn.model_selection import train_test_split
> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

## 3. 数据预处理

### 标准化

```
> from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(X_train) # 拟合
standardized_X = scaler.transform(X_train) # 训练集变换
standardized_X_test = scaler.transform(X_test) # 测试集变换
```

### 归一化

```
> from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(X_train) # 拟合
normalized_X = scaler.transform(X_train) # 训练集变换
normalized_X_test = scaler.transform(X_test) # 测试集变换
```

### 二值化

```
> from sklearn.preprocessing import Binarizer
binarizer = Binarizer(threshold=0.0).fit(X) # 拟合
binary_X = binarizer.transform(X) # 变换
```

### 编码分类特征

```
> from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
y = enc.fit_transform(y)
```

### 缺失值处理

```
> from sklearn.impute import SimpleImputer
imp = Imputer(missing_values=0, strategy='mean') # 均值填充器
imp.fit_transform(X_train) # 对数据进行缺失值均值填充变换
```

### 生成多项式特征

```
> from sklearn.preprocessing import PolynomialFeatures
poly = PolynomialFeatures(5)
poly.fit_transform(X)
```

## 4. 创建模型

### 有监督学习评估器

#### 线性回归

```
> from sklearn.linear_model import LinearRegression
lr = LinearRegression(normalize=True)
```

#### 支持向量机 (SVM)

```
> from sklearn.svm import SVC
svc = SVC(kernel='linear')
```

#### 朴素贝叶斯

```
> from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

#### KNN

```
> from sklearn import neighbors
knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

### 无监督学习评估器

#### 主成分分析 (PCA)

```
> from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
```

#### K-Means 聚类

```
> from sklearn.cluster import KMeans
k_means = KMeans(n_clusters=3, random_state=0)
```

Show Me AI

## 5. 模型拟合

### 有监督学习

```
> lr.fit(X, y) # 拟合数据与模型
> knn.fit(X_train, y_train)
> svc.fit(X_train, y_train)
```

### 无监督学习

```
> k_means.fit(X_train) # 拟合数据与模型
> pca_model = pca.fit_transform(X_train) # 拟合并转换数据
```

## 6. 预测

### 有监督评估器

```
> y_pred = svc.predict(np.random.random((2,5))) # 预测标签
> y_pred = lr.predict(X_test) # 预测标签
> y_pred = knn.predict_proba(X_test) # 评估标签概率
```

### 无监督评估器

```
> y_pred = k_means.predict(X_test) # 预测聚类算法里的标签
```



## 7. 评估模型性能

### 分类评价指标

#### 准确率

```
> svc.fit(X_train, y_train)
> svc.score(X_test, y_test) # 评估器评分法
> from sklearn.metrics import accuracy_score # 指标评分函数
> y_pred = svc.predict(X_test)
> accuracy_score(y_test, y_pred) # 评估 accuracy
```

#### 分类预估评价函数

```
> from sklearn.metrics import classification_report # 精确度、召回率、F1 分数及支持率
print(classification_report(y_test, y_pred))
```

#### 混淆矩阵

```
> from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

### 回归评价指标

#### 平均绝对误差

```
> from sklearn.metrics import mean_absolute_error

house_price = datasets.load_boston()
X, y = house_price.data, house_price.target
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor().fit(X_train, y_train)

y_pred = dt.predict(X_test)

mean_absolute_error(y_test, y_pred)
```

#### 均方误差

```
> from sklearn.metrics import mean_squared_error
mean_squared_error(y_test, y_pred)
```

#### R^2 评分

```
> from sklearn.metrics import r2_score
r2_score(y_test, y_pred)
```

### 聚类评价指标

#### 调整兰德系数

```
> from sklearn.metrics import adjusted_rand_score
adjusted_rand_score(y_true, y_pred)
```

#### 同质性

```
> from sklearn.metrics import homogeneity_score
homogeneity_score(y_true, y_pred)
```

#### V-measure

```
> from sklearn.metrics import v_measure_score
metrics.v_measure_score(y_true, y_pred)
```

### 交叉验证

```
> from sklearn.model_selection import cross_val_score
print(cross_val_score(knn, X_train, y_train, cv=4))
print(cross_val_score(lr, X, y, cv=2))
```

## 8. 模型调整

### 网格搜索超参优化

```
> from sklearn.model_selection import GridSearchCV
params = {"n_neighbors": np.arange(1,3),
          "metric": ["euclidean", "cityblock"]}
grid = GridSearchCV(estimator=knn, param_grid=params)
grid.fit(X_train, y_train)
print(grid.best_score_)
print(grid.best_estimator_.n_neighbors)
```

### 随机搜索超参优化

```
> from sklearn.model_selection import RandomizedSearchCV
params = {"n_neighbors": range(1,5), "weights": ["uniform", "distance"]}
rsearch = RandomizedSearchCV(estimator=knn,
                             param_distributions=params,
                             cv=4,
                             n_iter=8,
                             random_state=5)

rsearch.fit(X_train, y_train)
print(rsearch.best_score_)
```