機器學習神器Scikit-Learn保姆級入門教程

數據不吹牛 2022-01-16 22:03

以下文章來源於尤而小屋 ， 作者尤而小屋

尤而小屋

尤而小屋，一個溫馨且有愛的小屋🏠 小屋主人，一手代碼謀求生存，一手掌勺享受生活...

數據不吹牛

有趣+乾貨的數據分析寶藏

65篇原創內容

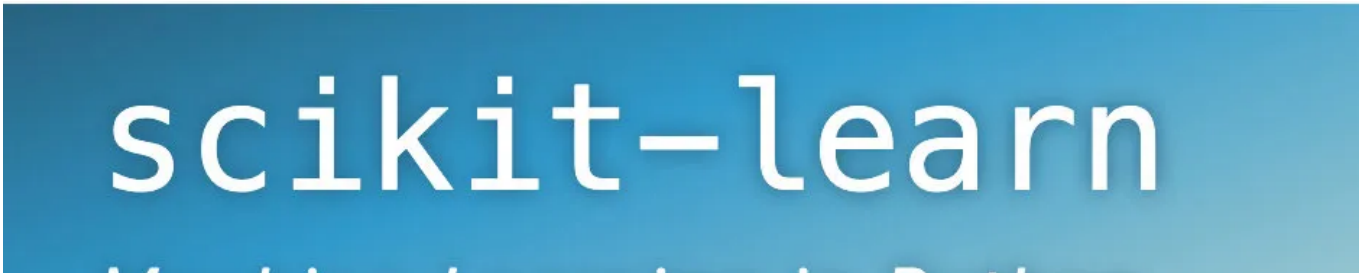
公眾號

Scikit-learn是一個非常知名的Python機器學習庫，它廣泛地用於統計分析和機器學習建模等數據科學領域。

- 建模無敵：用戶通過scikit-learn能夠實現各種監督和非監督學習的模型
- 功能多樣：同時使用sklearn還能夠進行數據的預處理、特徵工程、數據集切分、模型評估等工作
- 數據豐富：內置豐富的數據集，比如：泰坦尼克、鳶尾花等，數據不再愁啦

本篇文章通過簡明快要的方式來介紹scikit-learn的使用，更多詳細內容請參考官網：

1. 內置數據集使用
2. 數據集切分
3. 數據歸一化和標準化
4. 類型編碼
5. 建模6步曲



Scikit-learn使用神圖

下面這張圖是官網提供的，從樣本量的大小開始，分為回歸、分類、聚類、數據降維共4個方面總結了scikit-learn的使用：

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

安裝

關於安裝scikit-learn，建議通過使用anaconda來進行安裝，不用擔心各種配置和環境問題。當然也可以直接pip來安裝：

```
pip install scikit-learn
```

數據集生成

sklearn內置了一些優秀的數據集，比如：Iris數據、房價數據、泰坦尼克數據等。

```
import pandas as pd
import numpy as np

import sklearn
from sklearn import datasets  # 导入数据集
```

分類數據-iris數據

```
# iris数据
iris = datasets.load_iris()
type(iris)

sklearn.utils.Bunch
```

iris數據到底是什麼樣子？每個內置的數據都存在很多的信息

可以將上面的數據生成我們想看到的DataFrame，還可以添加因變量：

回歸數據-波士頓房價

我們重點關注的屬性：

- data

- target、target_names
- feature_names
- filename

同樣可以生成DataFrame：

三種方式生成數據

方式1

```
#调用模块
from sklearn.datasets import load_iris
data = load_iris()

#导入数据和标签
data_X = data.data
data_y = data.target
```

方式2

```
from sklearn import datasets
loaded_data = datasets.load_iris()  # 导入数据集的属性

#导入样本数据
```

```
data_X = loaded_data.data
# 导入标签
data_y = loaded_data.target
```

方式3

```
# 直接返回
data_X, data_y = load_iris(return_X_y=True)
```

數據集使用匯總

```
from sklearn import datasets  # 导入库

boston = datasets.load_boston()  # 导入波士顿房价数据
print(boston.keys())  # 查看键(属性)     ['data','target','feature_names','DE
print(boston.data.shape,boston.target.shape)  # 查看数据的形状
print(boston.feature_names)  # 查看有哪些特征
print(boston.DESCR)  # described 数据集描述信息
print(boston.filename)  # 文件路径
```

數據切分

```
# 导入模块
from sklearn.model_selection import train_test_split
# 划分为训练集和测试集数据
X_train, X_test, y_train, y_test = train_test_split(
  data_X,
  data_y,
  test_size=0.2,
  random_state=111
)

# 150*0.8=120
len(X_train)
```

數據標準化和歸一化

```
from sklearn.preprocessing import StandardScaler  # 标准化
from sklearn.preprocessing import MinMaxScaler  # 归一化

# 标准化
ss = StandardScaler()
X_scaled = ss.fit_transform(X_train)  # 传入待标准化的数据

# 归一化
mm = MinMaxScaler()
X_scaled = mm.fit_transform(X_train)
```

類型編碼

來自官網案例：https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

對數字編碼

對字符串編碼

建模案例

導入模塊

```
from sklearn.neighbors import KNeighborsClassifier, NeighborhoodComponentsAn
from sklearn.datasets import load_iris  # 导入数据
from sklearn.model_selection import train_test_split  # 切分数据
from sklearn.model_selection import GridSearchCV  # 网格搜索
from sklearn.pipeline import Pipeline  # 流水线管道操作

from sklearn.metrics import accuracy_score  # 得分验证
```

模型實例化

```
# 模型实例化
knn = KNeighborsClassifier(n_neighbors=5)
```

訓練模型

```
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier()
```

測試集預測

```
y_pred = knn.predict(X_test)
y_pred  # 基于模型的预测值
```

```
array([0, 0, 2, 2, 1, 0, 0, 2, 2, 1, 2, 0, 1, 2, 2, 0, 2, 1, 0, 2, 1, 2,
       1, 1, 2, 0, 0, 2, 0, 2])
```

得分驗證

模型得分驗證的兩種方式：

```
knn.score(X_test,y_test)
```

```
0.9333333333333333
```

```
accuracy_score(y_pred,y_test)
```

```
0.9333333333333333
```

網格搜索

如何搜索參數

```
from sklearn.model_selection import GridSearchCV

# 搜索的参数
knn_paras = {"n_neighbors":[1,3,5,7]}
# 默认的模型
knn_grid = KNeighborsClassifier()

# 网格搜索的实例化对象
grid_search = GridSearchCV(
 knn_grid,
 knn_paras,
 cv=10  # 10折交叉验证
```

```
)
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'n_neighbors': [1, 3, 5, 7]})
```

```
# 通过搜索找到的最好参数值
grid_search.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=7)
```

```
grid_search.best_params_
```

Out[42]:

```
{'n_neighbors': 7}
```

```
grid_search.best_score_
```

```
0.975
```

基於搜索結果建模

```
knn1 = KNeighborsClassifier(n_neighbors=7)

knn1.fit(X_train, y_train)
```

```
KNeighborsClassifier(n_neighbors=7)
```

通過下面的結果可以看到：網格搜索之後的建模效果是優於未使用網格搜索的模型：

```
y_pred_1 = knn1.predict(X_test)

knn1.score(X_test,y_test)
```

```
1.0
```

```
accuracy_score(y_pred_1,y_test)
```

```
1.0
```