10.模型选择

# 1 实验目的

1.了解交叉验证的实现原理，学会用sklearn.metrics中的常用方法计算法算法模型的各个指标；

2.了解网格搜素的实现原理，学会用GridSearchCV搜索算法模型的最佳参数。

# 2 实验要求

本次实验后，要求同学们能：

1.利用交叉验证，对模型进行评估；

2.利用网格搜索找出最佳参数，得到更可靠的模型。

# 3 实验内容

## 3.1 交叉验证

### 3.1.1 原理

通过前面的学习我们知道，同一个数据集，如果既用它训练模型，又用它进行预测，最后模型的准确度是偏高的，这是因为它过度拟合训练数据的同时，也拟合了训练数据的噪音。这样的模型就算看上去评分很好，泛化能力却不强，在新的数据上表现欠佳，甚至是不可用的。所以需要使用“干净”的验证集（训练时没有用到过的数据）来对模型的泛化能力进行测试。

前面章节中都是用train\_test\_split()方法将实验数据划分为一个训练集和一个测试集，在训练集上对模型进行训练，在测试集上对模型进行评估，这种评估模型的方法既简单又高效。

而另一种不错的评估模型的方式就是使用Scikit-learn的k折交叉验证功能。k折交叉验证会将实验数据划分成k个子集，每个子集称为一“折”或“折叠”。每次会用第i折以外的k-1折数的数据对模型进行训练，用第i折的数据让模型进行预测（i=1,2，……，k），重复k次。最后k折交叉验证会得到一个包含k个评估分数的数组。

k折交叉验证的好处在于，它不仅可以看到模型的评分，还能了解这个评分的精确度（由得到的k个评分可以知道评分的浮动范围）。而train\_test\_split()方法划分数据集进行验证只能得到一个评分，由于某些算法具有随机性，很可能正好算出了一个最高的评分或是最低的评分。但交叉验证法也不是在所有情况下都适用，交叉验证需要多次训练模型，会导致训练和验证的开销过大。

### 3.1.2 实验

（1）导入依赖加载数据集

本实验用KNN算法在红酒数据集上进行演示。

import numpy as np

import sklearn

from sklearn.neighbors import KNeighborsClassifier

from sklearn.datasets import load\_wine

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import train\_test\_split

# 加载数据集

wine\_data = load\_wine()

data = wine\_data.data

target = wine\_data.target

（2）交叉验证

可以用sklearn.model\_selection类中的KFold划分器的split()方法对数据集进行k折划分，再使用其返回的划分出的训练集、测试集索引进行交叉验证。

也可以直接用sklearn.model\_selection中的cross\_val\_score()方法更方便地实现交叉验证。cross\_val\_score()方法的重要参数及其含义如下：

**estimator：指定被验证的学习器。**这个“学习器”不一定是sklearn中提供的学习器，只需要这个构造器能提供sklearn中学习器需要提供的fit()、transform()、predict()等方法即可，如下一章中建立的流水线也可以作为学习器进行交叉验证。

**X：传入实验数据。**

**y：传入实验数据的标签。**

**scoring：**用字符串类型传入一个sklearn.metrics类中提供的评估标准。比如回归问题可以传入R2分数“r2\_score”（Scikit-learn中各个回归器score()方法的默认评价指标）、均方误差“mean\_squared\_error”等等；分类问题可以传入分类正确度“accuracy\_score”、精度“precision\_score”、召回率“recall\_score”等等。也可以传入一个自定义的评估分数计算方法。

**cv：设置k折交叉验证中要将数据集划分成的折数k。**

**fit\_params：可以用这个参数给被验证的学习器的fit()方法传递参数。比如许多分类器的fit()方法除了数据和标签还可以接受一个样本权值向量。**

cross\_val\_score()方法会返回一个**scores，**是由每次交叉验证运行时估算器得分所组成的数组，包含了k个得分。

用两种评估方式分别对算法模型的性能分别进行评估，对比两者的结果。

wine\_data = load\_wine()

data = wine\_data.data

target = wine\_data.target

# 法1：交叉验证进行测试

knn\_clf = KNeighborsClassifier()

scores = cross\_val\_score(estimator=knn\_clf, X=data, y=target, scoring="accuracy", cv=4)

print("CV Scores: ", scores)

print("CV Scores Mean: ", scores.mean())

wine\_data = load\_wine()

data = wine\_data.data

target = wine\_data.target

# 法2：划分测试集训练集进行测试

knn\_clf = KNeighborsClassifier()

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.25,

random\_state=1,

)

knn\_clf.fit(X\_train, Y\_train)

print("\nNormal Score: ", knn\_clf.score(X\_test, Y\_test))

### 3.1.3 实验结果

实验结果如下图：

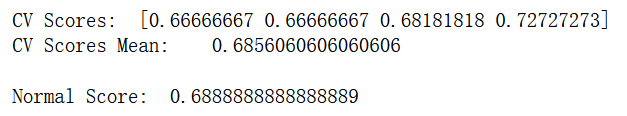


图1 两种评估方式的评估结果

可以看到用交叉验证的评估结果，与之前划分训练集测试集进行测试的方法得到的结果近似，而且利用交叉验证不仅能得到一个模型性能的评估值，还能得到这个评估值的准确度。

例如，上图中用默认参数的KNN算法对红酒数据集进行分类，准确度大致在[0.6667, 0.7273]这个区间内。

像之前在第3章中提到的一样，KNN算法涉及距离计算对数据的尺度敏感，需要先将数据标准化或归一化，否则准确率偏低，本实验没有进行标准化，正确率仅有68%左右。

在k折交叉验证中，依次将每一折作为测试集，其他折作为训练集，应训练并预测k次。每次都要用归一化或标准化转换器在训练集上拟合在全部数据上转换，会十分繁琐，用管道将转换与训练组合成流水线就可以解决这个问题，关于管道的介绍将在下一章节中说明。

## 3.2 网格搜索

### 3.2.1 原理

为了找到一个更好的模型，常常需要尝试大量不同的参数组合，人工进行调参枯燥乏味且耗时耗力。用Scikit-learn的GridSearchCV来进行网格搜索，能更轻松更准确地为机器学习算法模型找到合适的参数。

网格搜索的思想十分简单，就是先给每个参数选取几个可能的取值，然后根据给出的各个参数可能值的列表，形成一个不同参数可能值进行组合的网格，对网格上每一种参数组合都进行尝试，用它训练模型并验证，最后返回验证评分最高的一种参数组合。

当参数较多，且参数可能取值的范围较大时，参数组合过多，可能导致用普通的网格搜索开销过大，这时候可以用随机网格搜索RandomizedSearchCV更好地完成参数搜索任务。RandomizedSearchCV类与GridSearchCV类大致相同，只不过RandomizedSearchCV只指定迭代次数**n\_iter**，会在参数取值范围内任意选取**n\_iter种**参数组合，不一定得到最优解，但是可以在**n\_iter次迭代后及时停止，控制了搜索时间。**

### 3.2.2 实验

（1）导入依赖并处理数据

第6章支持向量机回归算法实验中，我们用参数C=10, degree=1, gamma="scale", kernel="sigmoid"，创建了一个支持向量机回归算法学习器对糖尿病数据集进行回归实验，其实这个参数组合就是由网格搜索得来，下面用糖尿病数据集进行演示。

import numpy as np

from sklearn.svm import SVR

from sklearn.datasets import load\_diabetes

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import GridSearchCV

# 导入数据集

diabetes\_data = load\_diabetes()

data = diabetes\_data.data

target = diabetes\_data.target

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(

data,

target,

test\_size=0.2,

random\_state=1

)

# 标准化

STD = StandardScaler()

STD.fit(X\_train)

X\_train = STD.transform(X\_train)

X\_test = STD.transform(X\_test)

（2）网格搜索找寻最佳参数

用构造函数sklearn.model\_selection模块的GridSearchCV类可以创建网格搜索算法实例，其构造器重要参数及含义如下：

**estimator：**指定待网格搜索最佳参数的学习器。这个“学习器”不一定是sklearn中提供的学习器，只需要这个构造器能提供sklearn中学习器需要提供的fit()、transform()、predict()等方法即可，如下一章中建立的流水线也可以作为学习器进行网格搜索。

**param\_grid：**输入一个以参数名为键，可能的参数取值列表为值的字典，或这样的字典列表。网格搜索器会根据这里输入的参数及可能取值，对所有参数组合进行网格搜索。

**scoring：**用字符串类型传入一个sklearn.metrics类中提供的评估标准。

**n\_jobs：**指定网格搜索时要并行运行的CPU内核数，默认为None，表示不并行搜索。取-1时表示用所有CPU内核进行并行搜素。

**cv：**确定交叉验证的切分策略，默认值为None，代表5折交叉验证。也可以设置为其他整数。

**refit：如果设置为True，则完成网格搜索后，直接那搜索到的参数组合创建模型并训练，可以直接在**GridSearchCV实例上调用predict()方法进行预测。

根据字典{"kernel":["linear", "rbf", "sigmoid"], "degree":[1, 3, 5, 7], "gamma":["scale", "auto"], "C":[0.1, 1, 10, 100]}，让网格搜索实例自动搜索最佳参数，参数“kernel”一共有3中选择，“degree”一共有4种选择，“gamma”一共有两种选择，“C”一共有4中选择（参数的具体含义请回顾第6章），总共有3×4×2×4 = 96种不同的参数组合方式。由于我们设置的是4折交叉验证，所以又会对每个模型训练4次，所以总共需要训练4×96 = 384个模型。

由上面的分析可以发现，如果要验证的参数及参数范围过大网格搜索会变得十分复杂，可以先组粒度搜索再细粒度搜索，或用随机网格搜索RandomizedSearchCV，从而减少参数搜索的开销。

我们构建一个对SVC算法模型根据{"kernel":["linear", "rbf", "sigmoid"], "degree":[1, 3, 5, 7], "gamma":["scale", "auto"], "C":[0.1, 1, 10, 100]}进行4折交叉验证的网格搜索算法实例，以“r2”指标为评价算法的指标。

# 网格搜索

param\_grid = {"kernel":["linear", "rbf", "sigmoid"], "degree":[1, 3, 5, 7],

"gamma":["scale", "auto"], "C":[0.1, 1, 10, 100]}

model = SVR()

grid\_search = GridSearchCV(model, param\_grid, cv=4, scoring="r2")

grid\_search.fit(data, target)

print(grid\_search.best\_params\_)

网格搜索的结果如下：



图2 网格搜索得到的参数

（3）用网格搜索出的参数创建模型

用上面搜索出的{'C': 10, 'degree': 1, 'gamma': 'scale', 'kernel': 'sigmoid'}创建支持向量机回归算法模型，并创建用一个默认参数的支持向量机模型作为对照组

# 用网格搜索出的参数训练模型

model = SVR(C=10, degree=1, gamma="scale", kernel="sigmoid")

model.fit(X\_train, Y\_train)

print("parameters = {'C': 10, 'degree': 1, 'gamma': 'scale', 'kernel': 'sigmoid'}:")

print("Score :", model.score(X\_test, Y\_test))

# 默认参数的对照组

model = SVR()

model.fit(X\_train, Y\_train)

print("Default Parameters SVR: ", model.score(X\_test, Y\_test))

两个模型的评分如下：

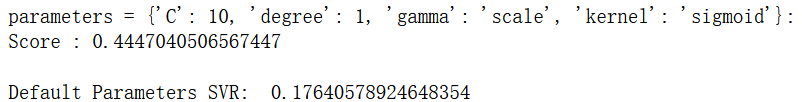


图3 模型评分

可以看到，使用了网格搜索得到的参数后，模型的R2评分大幅度上升。