**Sk-learn交叉验证**

# 划分训练集与测试集，只能测试是否过拟合，不能防止过拟合。

train\_test\_split

# 划分训练集、验证集与测试集，一定程度上防止过拟合（调整合适的参数）。

首先在训练集上训练模型，得到模型后紧接着在验证集上验证，如果验证的模型可以继续优化，则继续训练模型，否则在测试集上进行最终评估。

在训练集上进行训练，然后在验证集上进行评估，并且当训练似乎成功时，可以在测试集上进行最终评估。

# 划分训练集、验证集与测试集的缺点

* 大大减少了用于训练模型的样本数量；
* 模型的预测结果取决于划分出来的训练集与验证集的随机组合；

为了解决上述问题，提出交叉验证的思想

# K折交叉验证

不在保留验证集，只把数据集划分为训练集与测试集。具体做法如下：

将数据集划分为k个子集；

使用k-1个子集用于训练模型，将得到的模型用剩下的数据验证，得到模型指标；

最后取k个指标的平均值

缺点：计算量大；

优点：不浪费数据，适合样本点少的情况

# 数据的独立同分布假定（实际上很难成立）

* 所有的样本数据都来自同一个生成过程；
* 生成过程对历史数据没有记忆能力；

# Sk-learn交叉验证函数

## train\_test\_split()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.4, random\_state=0)

## cross\_val\_score()

>>> clf = svm.SVC(kernel='linear', C=1)

>>> scores = cross\_val\_score(clf, iris.data, iris.target, cv=5,scoring='f1\_macro'))

>>> print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() \* 2))

默认情况下，scores是模型的score方法，

对于分类：sklearn.metrics.accuracy\_score

对于回归：sklearn.metrics.r2\_score

可通过scoring参数指定。

若cv是整数，则默认使用KFold or StratifiedKFold交叉验证

其他的交叉验证策略，即使用交叉验证迭代器。

>>> from sklearn.model\_selection import ShuffleSplit

>>> n\_samples = iris.data.shape[0]

>>> cv = ShuffleSplit(n\_splits=5, test\_size=0.3, random\_state=0)

>>> cross\_val\_score(clf, iris.data, iris.target, cv=cv)

**n\_splits=5代表5折交叉验证**

## cross\_validate

* 可以使用多个评价指标；
* 返回一个字典；

## cross\_val\_predict

返回在测试集中的预测值。因为在交叉验证中，每一个值都只在测试集中出现一次。

cross\_val\_predict的结果可能与使用cross\_val\_score获得的结果不同，因为元素以不同的方式分组。 函数cross\_val\_score采用交叉验证折叠的平均值，而cross\_val\_predict只是从几个不同的模型返回标记（或概率）。 因此，cross\_val\_predict不是对泛化错误的适当度量。

from sklearn import datasets

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

from sklearn import linear\_model

import matplotlib.pyplot as plt

lr = linear\_model.LinearRegression()

boston = datasets.load\_boston()

X = boston.data

y = boston.target

# cross\_val\_predict returns an array of the same size as `y` where each entry

# is a prediction obtained by cross validation:

predicted = cross\_val\_predict(lr, X, y, cv=10)

fig, ax = plt.subplots()

ax.scatter(y, predicted, edgecolors=(0, 0, 0))

ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)

ax.set\_xlabel('Measured')

ax.set\_ylabel('Predicted')

plt.show()

# 交叉验证迭代器

## KFold

## RepeatedKFold

## LeaveOneOut

## LeavePOut

## ShuffleSplit（随机性）

## StratifiedKFold（分层性）

## RepeatedStratifiedKFold

## StratifiedShuffleSplit

# 超参数选择

通过交叉验证搜索超参数空间。

estimator.get\_params()，得到学习器的全部参数的名称和当前值

## GridSearchCV

param\_grid = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},

{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},]

estimator.best\_params\_

GridSearchCV(estimator, param\_grid, scoring=None, fit\_params=None, n\_jobs=1, iid=True, refit=True, cv=None, verbose=0, pre\_dispatch=‘2\*n\_jobs’, error\_score=’raise’, return\_train\_score=’warn’)

* estimator：所使用的分类器，比如：estimator=RandomForestClassifier(min\_samples\_split=100, min\_samples\_leaf=20, max\_depth=8, max\_features='sqrt', random\_state=10)，并且传入除需要确定最佳的参数之外的其他参数。每个分类器都需要一个scoring参数或者score方法。
* param\_grid：值为字典或列表，即需要最优化的参数的取值，param\_grid =param\_test1，param\_test1 = {'n\_estimators':range(10,71,10)}
* scoring：准确评价标准，默认为None（使用estimator的误差估计函数），这时需要使用score函数；或者如scoring='roc\_auc'，根据所选模型不同，评价准则不同。
* cv：交叉验证参数，默认为None
* refit：默认为True，程序将会以交叉验证训练集得到的最佳参数，重新对所有可用的训练集与测试集进行，作为最终用于性能评估的最佳模型参数。即在搜索参数结束后，用最佳参数结果再次fit一遍全部数据集。
* iid:默认True,为True时，默认为各个样本fold概率分布一致，误差估计为所有样本之和，而非各个fold的平均。
* verbose：日志冗长度，int：冗长度，0：不输出训练过程，1：偶尔输出，>1：对每个子模型都输出。
* n\_jobs: 并行数，int：个数,-1：跟CPU核数一致, 1:默认值。
* pre\_dispatch：指定总共分发的并行任务数。当n\_jobs大于1时，数据将在每个运行点进行复制，这可能导致OOM，而设置pre\_dispatch参数，则可以预先划分总共的job数量，使数据最多被复制pre\_dispatch次，进行预测的常用方法和属性
* grid.fit()：运行网格搜索
* grid\_scores\_：给出不同参数情况下的评价结果
* best\_params\_：描述了已取得最佳结果的参数的组合
* best\_score\_：成员提供优化过程期间观察到的最好的评分

## RandomizedSearchCV

n\_iter：指定迭代几次。

•可以选择预算，与参数数量和可能的值无关。

•添加不影响性能的参数不会降低效率。

This example uses the scipy.stats module, which contains many useful distributions for sampling parameters,such as expon, gamma, uniform or randint.

{'C':scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),'kernel': ['rbf'], 'class\_weight':['balanced', None]}

# 参数搜索建议

## 改变scoring

默认情况下，scores是模型的score方法，

对于分类：sklearn.metrics.accuracy\_score

对于回归：sklearn.metrics.r2\_score

## 指定多个scorings

best\_params\_, refit parameter

## 并行化

# 模型的性能评估的三种方法

## 模型的Score方法

## 交叉验证的Scoring参数

model\_selection.cross\_val\_score and model\_selection.GridSearchCV

## 性能函数

metrics 模块，Scoring参数与函数一一对应

|  |  |
| --- | --- |
| Scoring | Function |
| **Classification** | |
| ‘accuracy’ | metrics.accuracy\_score |
| ‘balanced\_accuracy’ | metrics.balanced\_accuracy\_score |
| ‘average\_precision’ | metrics.average\_precision\_score |
| ‘brier\_score\_loss’ | metrics.brier\_score\_loss |
| ‘f1’ | metrics.f1\_score |
| ‘f1\_micro’ | metrics.f1\_score |
| ‘f1\_macro’ | metrics.f1\_score |
| ‘f1\_weighted’ | metrics.f1\_score |
| ‘f1\_samples’ | metrics.f1\_score |
| ‘neg\_log\_loss’ | metrics.log\_loss |
| ‘precision’etc. | metrics.precision\_score |
| ‘recall’etc. | metrics.recall\_score |
| ‘roc\_auc’ | metrics.roc\_auc\_score |
| **Clustering** | |
| ‘adjusted\_mutual\_info\_score’ | metrics.adjusted\_mutual\_info\_score |
| ‘adjusted\_rand\_score’ | metrics.adjusted\_rand\_score |
| ‘completeness\_score’ | metrics.completeness\_score |
| ‘fowlkes\_mallows\_score’ | metrics.fowlkes\_mallows\_score |
| ‘homogeneity\_score’ | metrics.homogeneity\_score |
| ‘mutual\_info\_score’ | metrics.mutual\_info\_score |
| ‘normalized\_mutual\_info\_score’ | metrics.normalized\_mutual\_info\_score |
| ‘v\_measure\_score’ | metrics.v\_measure\_score |
| **Regression** | |
| ‘explained\_variance’ | **metrics.explained\_variance\_score** |
| ‘neg\_mean\_absolute\_error’ | **metrics.mean\_absolute\_error** |
| ‘neg\_mean\_squared\_error’ | **metrics.mean\_squared\_error** |
| ‘neg\_mean\_squared\_log\_error’ | metrics.mean\_squared\_log\_error |
| ‘neg\_median\_absolute\_error’ | metrics.median\_absolute\_error |
| ‘r2’ | **metrics.r2\_score** |

回归问题加粗的函数都可用于多个输出问题：multioutput参数

# 两条曲线

## 验证曲线

横坐标是单个超参数，纵坐标是训练集或者验证集上的得分

**validation\_curve函数**

>>> import numpy as np

>>> from sklearn.model\_selection import validation\_curve

>>> from sklearn.datasets import load\_iris

>>> from sklearn.linear\_model import Ridge

>>> np.random.seed(0)

>>> iris = load\_iris()

>>> X, y = iris.data, iris.target

>>> indices = np.arange(y.shape[0])

>>> np.random.shuffle(indices)

>>> X, y = X[indices], y[indices]

>>> train\_scores, valid\_scores = validation\_curve(Ridge(), X, y, "alpha", np.logspace(-7, 3, 3), cv=5)

array([[0.93..., 0.94..., 0.92..., 0.91..., 0.92...],

[0.93..., 0.94..., 0.92..., 0.91..., 0.92...],

[0.51..., 0.52..., 0.49..., 0.47..., 0.49...]])

>>> valid\_scores

array([[0.90..., 0.84..., 0.94..., 0.96..., 0.93...],

[0.90..., 0.84..., 0.94..., 0.96..., 0.93...],

[0.46..., 0.25..., 0.50..., 0.49..., 0.52...]])

## 学习曲线

横坐标是训练样本个数，纵坐标是训练集或者验证集上的得分

**learning\_curve函数**

>>> from sklearn.model\_selection import learning\_curve

>>> from sklearn.svm import SVC

>>> train\_sizes, train\_scores, valid\_scores = learning\_curve(

SVC(kernel='linear'), X, y, train\_sizes=[50, 80, 110], cv=5)

>>> train\_sizes

array([ 50, 80, 110])

>>> train\_scores

array([[0.98..., 0.98 , 0.98..., 0.98..., 0.98...],

[0.98..., 1. , 0.98..., 0.98..., 0.98...],

[0.98..., 1. , 0.98..., 0.98..., 0.99...]])

>>> valid\_scores

array([[1. , 0.93..., 1. , 1. , 0.96...],

[1. , 0.96..., 1. , 1. , 0.96...],

[1. , 0.96..., 1. , 1. , 0.96...]])