scikit-learn K近邻法类库使用小结

刘建平Pinard 机器学习算法那些事 2019-01-04

作者：刘建平Pinard
链接：https://www.cnblogs.com/pinard/p/6065607.html
编辑：石头

本文对scikit-learn中KNN相关的类库使用做了一个总结，主要关注于类库调参时的一个经验总结，且该文非常详细地介绍了类的参数含义，这是小编见过最详细的KNN类库参数介绍 ， K近邻算法原理请结合文章《K近邻算法(KNN)原理小结》来理解 。

目录

1. scikit-learn中KNN相关的类库概述

在scikit-learn 中，与近邻法这一大类相关的类库都在sklearn.neighbors包之中。KNN分类树的类是KNeighborsClassifier，KNN回归树的类是KNeighborsRegressor。除此之外，还有KNN的扩展，即限定半径最近邻分类树的类RadiusNeighborsClassifier和限定半径最近邻回归树的类RadiusNeighborsRegressor， 以及最近质心分类算法NearestCentroid。

在这些算法中，KNN分类和回归的类参数完全一样。限定半径最近邻法分类和回归的类的主要参数也和KNN基本一样。

比较特别是的最近质心分类算法，由于它是直接选择最近质心来分类，所以仅有两个参数，距离度量和特征选择距离阈值，比较简单，因此后面就不再专门讲述最近质心分类算法的参数。

另外几个在sklearn.neighbors包中但不是做分类回归预测的类也值得关注。kneighbors_graph类返回用KNN时和每个样本最近的K个训练集样本的位置。radius_neighbors_graph返回用限定半径最近邻法时和每个样本在限定半径内的训练集样本的位置。NearestNeighbors是个大杂烩，它即可以返回用KNN时和每个样本最近的K个训练集样本的位置，也可以返回用限定半径最近邻法时和每个样本最近的训练集样本的位置，常常用在聚类模型中。

2. K近邻法和限定半径最近邻法类库参数小结

本节对K近邻法和限定半径最近邻法类库参数做一个总结。包括KNN分类树的类KNeighborsClassifier，KNN回归树的类KNeighborsRegressor， 限定半径最近邻分类树的类RadiusNeighborsClassifier和限

定半径最近邻回归树的类RadiusNeighborsRegressor。这些类的重要参数基本相同，因此我们放到一起讲：

参数	KNighborsClassfie	KNighborsRegressor	RadiusNeighborsCl	RadiusNeighborsRe
KNN中的K值 n_neighors	K值的选择与样本分布有关，一般选择一个较小的K值，可以通过交叉验证来选择一个比较优的K值，默认值是5。		不适用于限定半径最近邻法	
限定半径最近邻法中的半径 radius	不适用于KNN		半径的选择与样本分布有关，可以通过交叉验证来选择一个较小的半径，尽量保证每类训练样本其他类别样本的距离较远，默认值是1.0。	
近邻权 weights	选择默认的"uniform"，意味着所有最近邻样本权重都一样，在做预测时一视同仁。如果是"distance"，则权重和距离成反比例，即距离预测目标更近的近邻具有更高的权重，这样在预测类别或者做回归时，更近的近邻所占的影响因子会更加大。当然，我们也可以自定义权重，即自定义一个函数，输入是距离值，输出是权重值。这样我们可以自己控制不同的距离所对应的权重。			
KNN和限定半径最近邻法使用的算法 algorithm	算法一共有三种，第一种是蛮力实现，第二种是KD树实现，第三种是球树实现。对于这个参数，一共有4种可选输入，‘brute’对应第一种蛮力实现，‘kd_tree’对应第二种KD树实现，‘ball_tree’对应第三种的球树实现，‘auto’则会在上面三种算法中做权衡，选择一个拟合最好的最优算法。需要注意的是，如果输入样本特征是稀疏的时候，无论我们选择哪种算法，最后scikit-learn都会去用蛮力实现‘brute’			
停止建子树的叶子节点阈值 leaf_size	这个值控制了使用KD树或者球树时， 停止建子树的叶子节点数量的阈值。这个值越小，则生成的KD树或者球树就越大，层数越深，建树时间越长，反之，则生成的KD树或者球树会小，层数较浅，建树时间较短。默认是30. 这个值一般依赖于样本的数量，随着样本数量的增加，这个值必须要增加，否则不光建树预测的时间长，还容易过拟合。可以通过交叉验证来选择一个适中的值。 如果使用的算法是蛮力实现，则这个参数可以忽略。			

距离度量 metric	K近邻法和限定半径最近邻法类可以使用的距离度量较多，一般来说默认的欧式距离（即p=2的闵可夫斯基距离）就可以满足我们的需求。可以使用的距离度量参数有： a)欧氏距离"euclidean"：$\sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$ b) 曼哈顿距离"manhattan"：$\sum_{i=1}^{n}\lvert x_i-y_i\rvert$ c)切比雪夫距离"chebyshev"：$\max\lvert x_i-y_i\rvert(i=1,2,...n)$ d)闵可夫斯基距离"minkowski"（默认参数）$\sqrt[p]{\sum_{i=1}^{n}(x_i-y_i)^p}$ e)带权重闵可夫斯基距离"wminkowski"：$\sqrt[p]{\sum_{i=1}^{n}(w*\lvert x_i-y_i\rvert)^p}$ f)标准化欧氏距离"seuclidean"：即对于各特征维度做了归一化以后的欧氏距离。此时各特征维度的均值为0，方差为1。 g) 马氏距离"mahalanobis"：$\sqrt{(x-y)^T S^{-1}(x-y)}$,其中,$S^{-1}$为样本协方差矩阵的逆矩阵。 当样本分布独立时，S为单位矩阵，此时马氏距离等同于欧式距离。

距离度量参数	p是使用距离度量参数 metric 附属参数，只用于闵可夫斯基距离和带权重闵可夫斯基距离中p值的选择，p=1为曼哈顿距离， p=2为欧式距离。默认为2		
距离度量其他附属参数 metric_params	一般都用不上，主要是用于带权重闵可夫斯基距离的权重，以及其他一些比较复杂的距离度量的参数。		
并行处理任务数 n_jobs	主要用于多核CPU时的并行处理，加快建立KNN树和预测搜索的速度。一般用默认的-1就可以了，即所有的CPU核都参与计算。	不适用于限定半径最近邻法	
异常点类别选择 outlier_label	不适用于KNN	主要用于预测时，如果目标点半径内没有任何训练集的样本点时，应该标记的类别，不建议选择默认值 none,因为这样遇到异常点会报错。一般设置为训练集里最多样本的类别。	不适用于限定半径最近邻回归

3. 使用KNeighborsClassifier做分类的实例

完整代码见github:

https://github.com/ljpzzz/machinelearning/blob/master/classic-machine-learning/knn_classifier.ipynb

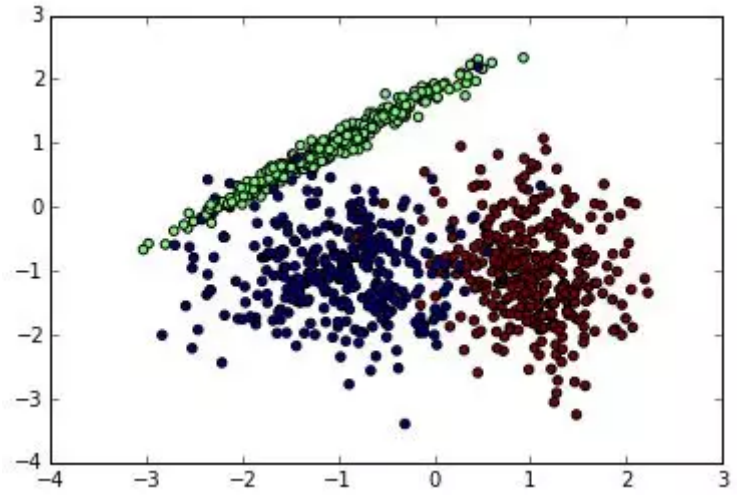
3.1 生成随机数据

首先，我们生成我们分类的数据，代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets.samples_generator import make_classification
# X为样本特征，Y为样本类别输出， 共1000个样本，每个样本2个特征，输出有3个类别，没有冗余特征，每个类别一个簇
X, Y = make_classification(n_samples=1000, n_features=2, n_redundant=0,
                           n_clusters_per_class=1, n_classes=3)
```

```
plt.scatter(X[:, 0], X[:, 1], marker='o', c=Y)
plt.show()
```

先看看我们生成的数据图如下。由于是随机生成，如果你也跑这段代码，生成的随机数据分布会不一样。下面是我某次跑出的原始数据图。



接着我们用KNN来拟合模型，我们选择K=15，权重为距离远近。代码如下：

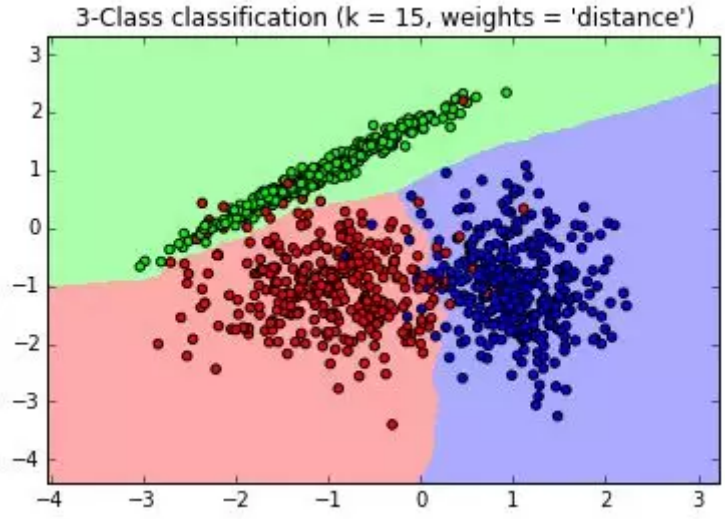
```
from matplotlib.colors import ListedColormap
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

#确认训练集的边界
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
#生成随机数据来做测试集然后预测
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
                     np.arange(y_min, y_max, 0.02))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# 画出测试集数据
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# 也画出所有的训练集数据
plt.scatter(X[:, 0], X[:, 1], c=Y, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("3-Class classification (k = 15, weights = 'distance')" )
```

生成的图如下，可以看到大多数数据拟合不错，仅有少量的异常点不在范围内。



推荐阅读

K近邻算法(KNN)原理小结