3.最邻近算法

# 1 实验目的

1.理解sklearn.neighbors模块中KNN算法的实现原理；

2.了解sklearn.neighbors模块中KNN回归、分类算法的使用方法。

3.了解如何使用sklearn.neighbors模块中各个常用算法完成机器学习任务。

# 2 实验要求

本次实验后，要求同学们能：

1.掌握KNN分类算法的实现原理；

2.学会使用Scikit-learn实现常用的最邻近算法；

3.了解sklearn.neighbors模块中各常用算法的不同特点，理解k值对KNN算法的影响。

# 3 实验内容

## 3.1 最邻近算法简介

最常见的是k邻近算法（KNN），所谓“k邻近”就是用k个最近的邻居，即训练集中用k个最邻近的样本点来决定新样本的属性。

主要包含三个基本要素——距离度量（例如欧氏距离）、k值的选择、分类决策规则（例如少数服从多数投票法）。这三个基本要素以及训练数据集确定后，对于任意新的输入，它所属类别都会被唯一第确定。

scikit-learn库中sklearn. neighbors模块提供了不同的k邻近算法模型，下面介绍最基础的KNN分类、回归算法。

sklearn.neighbors中的其他算法都原理相似，是在这两个基础算法上进行优化或者扩展。如用kd树作为存储结构提高k邻近搜索效率的KDTree；

## 3.2 k邻近分类算法

### 3.2.1原理

k邻近算法原理十分简单。为了判断未知样本的类别，以所有已知类别的样本作为参照，来计算未知样本与所有已知样本的距离，然后从中选取与未知样本距离最近的 k 个已知样本，并根据少数服从多数的投票法则（majority-voting），将未知样本与 k 个最邻近样本中所属类别占比较多的归为一类。

KNN算法需要计算新数据到各个训练数据的距离，距离的度量主要有以下几种：

曼哈顿距离：

欧几里得距离：

明科夫斯基距离。

不同的距离度量可能算出不同的邻近点，需要结合具体使用场景来选择合适的度量方式，一般选用欧氏距离。

### 3.2.2 实验

（1）导入依赖并加载数据

用Scikit-learn自带的鸢尾花数据集作为实验数据，数据集一共含有150个样本（每一类各50个样本），每一行对应一朵花，列代表每朵花的四个测量数据，分别是：花瓣的长度，宽度，花萼的长度、宽度。最后一项表示鸢尾花的类别，共三类，分别是山鸢尾（0表示）、色鸢尾（1表示）、维吉尼亚鸢尾（2表示）

先将数据集划分为训练集和测试集，由于KNN算法涉及到距离计算，对特征尺度敏感，所以要对数据进行标准化处理。

注意：标准化、归一化等转换器**只用训练数据来拟合，用拟合出的转换器转换全部数据**。如果在整个数据集上拟合会将测试集的信息引入到训练集中。

数据预处理代码如下：

from sklearn.neighbors import KNeighborsClassifier

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

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

# 加载数据集

iris\_data = load\_iris()

data = iris\_data.data

target = iris\_data.target

# 划分数据集

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

data,

target,

test\_size=0.2,

random\_state=1

)

X = X\_test[-3:] # 取最后3个测试数据单独检验

# 标准化

STD = StandardScaler()

STD.fit(X\_train)

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

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

（2）创建算法模型实例

用sklearn.neighbors模块的KneighborsClassifier类可以创建KNN算法模型实例，其构造器重要参数及其含义如下：

**n\_neighbors：**指定KNN算法的k值，即选取距离新样本点最近的k个样本点进行投票来判断类别，默认为5，会很大程度影响算法的结果。

**weights：**设置k个邻近点投票时的权重。默认为“**uniform**”即统一权重，k个邻近点投票的影响力相同；取“**distance**”时，将邻近点权重设置为距离的倒数，从而更近的邻近点比更远的临近点投票时有更大的影响力；还可以设为用户**自定义**的函数，要求函数接受距离数组，并返回一个同结构的权值数组。

**p：**用来指定明科夫斯基距离的参数。p=1时是曼哈顿距离。p=2时，使用欧几里得距离。p为其他任意值时，使用明科夫斯基距离。

创建一个k值为6，用p=3的明科夫斯基公式计算距离的k邻近分类算法实例。

model1 = KNeighborsClassifier(n\_neighbors=6, p=3)

（3）训练并测试

该算法实例提供以下方法：

kneighbors(X=None, n\_neighbors=None, return\_distance=True)：对X中的每个待测试样本xi，分别找出训练集中离xi最近的n\_neighbors个样本点的索引序号。return\_distance为True时，同时返回这n\_neighbors个最邻近点到xi的距离。

predict\_proba(X)：返回X中个样本属于每个类别的概率组成的矩阵。

此外，fit、predict、get\_params、get\_params、score等方法的用法不再赘述。

下面对KNN分类算法模型进行训练并验证。

# 训练

model1.fit(X\_train, Y\_train)

# 评分

print("\n训练集评分", model1.score(X\_train, Y\_train))

print("测试集评分", model1.score(X\_test, Y\_test))

print("最后三条测试数据", X)

X = STD.transform(X)

print("最后三条测试数据最近的6个点的距离及其序号索引")

print(model1.kneighbors(X, n\_neighbors=6, return\_distance=True))

print("最后三条测试数据属于每一类的概率")

print(model1.predict\_proba(X))

print("最后三条测试数据的分类结果为", model1.predict(X))

print("最后三条测试数据的实际类别位", Y\_test[-3:])

（4）查看训练后的参数

可以用p**arameter\_（参数名加下划线）的形式直接访问训练得到的以下参数：**

**classes\_：查看分类器已知的分类标签。**

**effective\_metric\_：查看用到的距离计算公式（返回公式名称）。**

**effective\_metric\_params\_：查看距离计算法公式的参数。**

**n\_features\_in\_：查看特征个数。**

**feature\_names\_in\_：仅当X的特征有字符串类型的特征名时，可查看X的各个特征名。**

**n\_samples\_fit\_：查看用于拟合的数据个数。**

**代码如下：**

# 查看参数

print("classes\_", model1.classes\_)

print("effective\_metric\_", model1.effective\_metric\_)

print("effective\_metric\_params\_", model1.effective\_metric\_params\_)

print("n\_features\_in\_", model1.n\_features\_in\_)

# print("feature\_names\_in\_", model1.feature\_names\_in\_)

print("n\_samples\_fit\_", model1.n\_samples\_fit\_)

### 3.2.3 实验结果

（1）算法模型评分



图1 KNN分类模型评分

（2）取三个测试数据单独检验

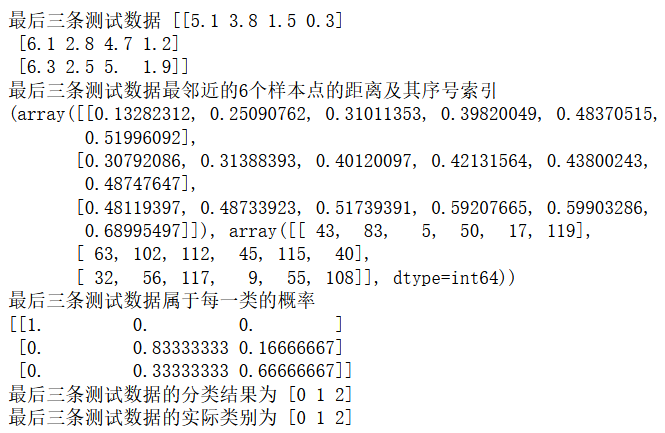


图2 单独测验

（3）查看训练出的模型的各个参数

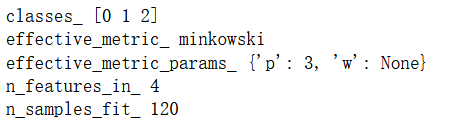


图3 查看参数

## 3.3 K邻近回归算法

### 3.3.1原理

KNN算法也可以用于回归问题，过程与KNN分类算法类似，第一步也是找出离待测样本xi最近的k个样本点；找到之后，分类算法是根据这k个最邻近样本点投票确定新样本的分类，而回归算法是计算这k个最邻近的样本点标签的平均值作为对新样本点的预测。

### 3.3.2 实验

（1）导入依赖并加载数据

加载鸢尾花数据集，用鸢尾花的花萼长度、花萼宽度、花瓣长度，来预测花瓣宽度。对训练数据、测试数据进行标准化处理。

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsRegressor

from sklearn.datasets import load\_iris

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

from sklearn.preprocessing import StandardScaler

# 加载数据集

iris\_data = load\_iris()

# 用鸢尾花的花萼长度、花萼宽度、花瓣长度，来预测花瓣宽度

data = iris\_data.data[:, :-1]

target = iris\_data.data[:, np.newaxis, -1]

# 划分数据集

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）KNN回归实验

用sklearn.neighbors模块的KneighborsRegressor类可以创建KNN回归预测算法实例，其构造器与KNN分类算法构造器接受相似的参数，且创建的模型实例有相似的方法与成员变量，不再重复说明。

下面根据鸢尾花的花萼长度、花萼宽度、花瓣长度，使用KNN回归算法预测花瓣宽度，并在测试集上进行验证，实验步骤与KNN分类算法一致，代码如下：

# 创建实例

model2 = KNeighborsRegressor(n\_neighbors=6)

# 训练

model2.fit(X\_train, Y\_train)

# 预测测试集

result = model2.predict(X\_test)

# 评分

print("\n训练集评分", model2.score(X\_train, Y\_train))

print("测试集评分", model2.score(X\_test, Y\_test))

（3）实验结果可视化

用matplotlib可视化回归预测的结果，横轴是数据编号，纵轴是花瓣的宽度。

# 可视化

plt.plot(result,"ro-",label="predict value")

plt.plot(Y\_test,"bo--",label="real value")

plt.title("KNNRegression")

plt.xlabel("index")

plt.ylabel("value")

plt.legend()

plt.show()

### 3.3.3 实验结果

（1）模型评分



图4 KNN回归算法评分

（2）绘图展示出模型对测试集的预测结果

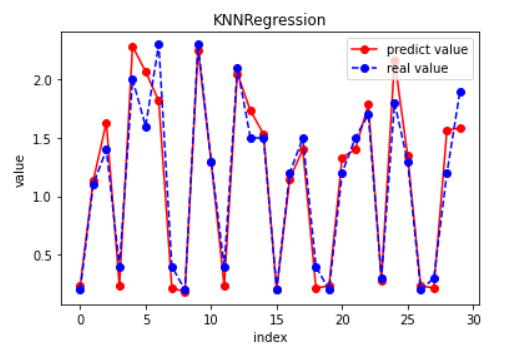


图5 KNN回归算法预测结果

## 3.4 k值的选择

### 3.4.1 原理

KNN算法中，k值的选取会对算法的结果产生很大的影响。

k值选取较小，会导致只用较少的邻域中的训练样本来进行预测，如果恰巧待测数据周围都是噪声，那么预测容易出错；如果k值选取过大，距离待测数据较远的实例也会在决策中起到作用（尽管它跟待测数据距离并不相似），影响最终的判断。

所以可以看出，k值越小，模型就越复杂；k值越大，模型就越简单。过于复杂和过于简单都不是一件好事。在实际应用中，k一般不会取得很大，可以通过网格搜索来求出最佳的k值（网格搜索将在第10章中介绍）。

### 3.4.2 实验

（1）数据准备

用Scikit-learn.datasets中的make\_moons()方法得到带有噪音的两个交错半月（双月型）数据，每个半月代表一类，可以用这个数据集来展示二分类算法实验。make\_moons()方法的重要参数及含义如下：

**n\_samples：**以元组的形式指定两个类别各自的样本数，元组的两个元素分别代表两个半月的样本数。

**shuffle：设置是否打乱样本，默认会打乱.**

**noise：**用于指定噪声大小，默认噪声为0。

**random\_state：**用于固定噪声的随机性，一旦random\_state固定，方法用相同的参数产生的数据集相同。

创建一个双月型数据集用于检测k值对KNN算法的影响，用matplotlib绘制出双月数据集图像。

import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

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

# 导入双月型数据集

from sklearn.datasets import make\_moons

moons = make\_moons(n\_samples=(60, 50), noise=0.4, random\_state=5)

data = moons[0]

target = moons[1]

# 绘制图像

plt.scatter(data[:, 0], data[:, 1], c=target, s=30, cmap="cool")

plt.xlabel('X')

plt.ylabel('Y')

plt.title("moons noise=0.4")

plt.show()

# 划分数据集

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

data,

target,

test\_size=0.3,

random\_state=1

)

（2）创建k值不同的KNN分类算法模型进行实验

创建k分别为1、10、20、30、40、50、70的KNN算法模型，用它们在双月数据集的训练集上进行训练，打印出评分。

k\_list = [1, 10, 20, 30]

KNN\_models = [KNeighborsClassifier(n\_neighbors=k) for k in k\_list]

for model in KNN\_models:

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

print("k=%d" % model.n\_neighbors)

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

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

### 3.4.3 实验结果

（1）实验数据可视化

创建的双月型数据集图像如下：

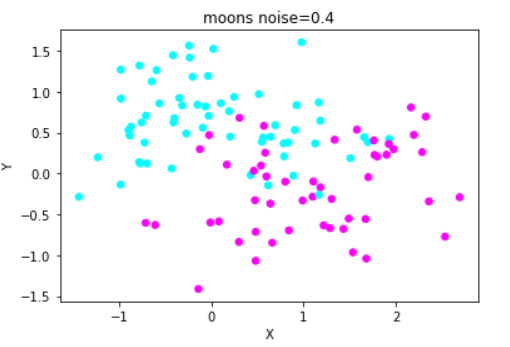


图6 双月型数据集图像

（2）不同k值的KNN算法评分

不同k值的KNN算法评分如下：

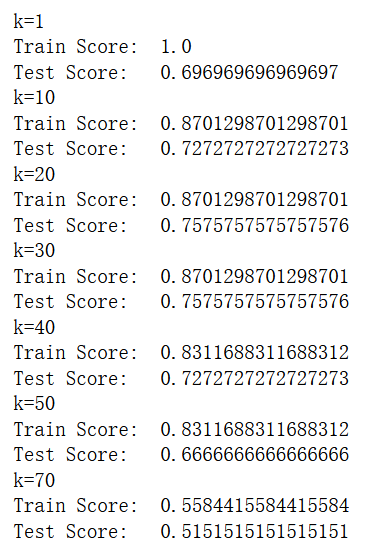


图7 k值对KNN算法的影响

由上图可以看出，k=1时算法模型在训练集上准确度达到100%，但是测试集上表现不好，说明模型过于复杂，产生了过拟合。k从1增加到30的过程中，模型不断简化，在训练集上准确率下降，但是泛化能力提高，测试集上准确率提高。k取更大的值，比如大于40时，模型就变得过于简单，忽略了大量有用信息，训练集、测试集上准确率都在下降。