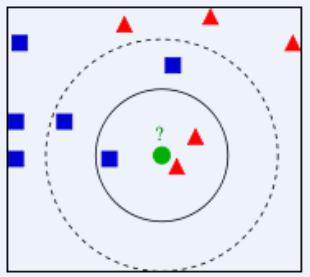
机器学习：K最近邻算法

K最近邻（kNN，k-NearestNeighbor）也称近邻算法，主要用于数据挖掘分类技术中最简单常见的一种算法。

原理

K最近邻算法，就是K个最近的邻居的意思，每个样本数据都可以用它最接近的K个邻居来进行代表。



通过上图中可以清晰的看出图中的数据分为两类，一类为红色一类为蓝色。现在需要我们确定出来绿色点的为哪一类颜色，可以根据绿色点临近的K个点进行来确认，当我们确认K数字为3是可以看出距离绿色点最近的分布为蓝色、红色、红色，那么就可以进行标记绿色点为红色。

关于K的取值是一个非常重要的，当我们K的数值选取较小的时候可能会导致数据分类出现巨大的错误，在scikit-learn中，K最近邻算法的K值是通过n_neighbors参数来进行调节设置，默认值为5。

K最近邻算法也可以用于回归，原理和其用于分类的是相同的。当我们使用K最近邻回归计算某个数据点的预测值时，模型会选择该数据最近的若干个训练数据集中的点，并且进行计算他们的y值取平均值，并且把该平均值作为新数据的预测值。

用法

分类任务

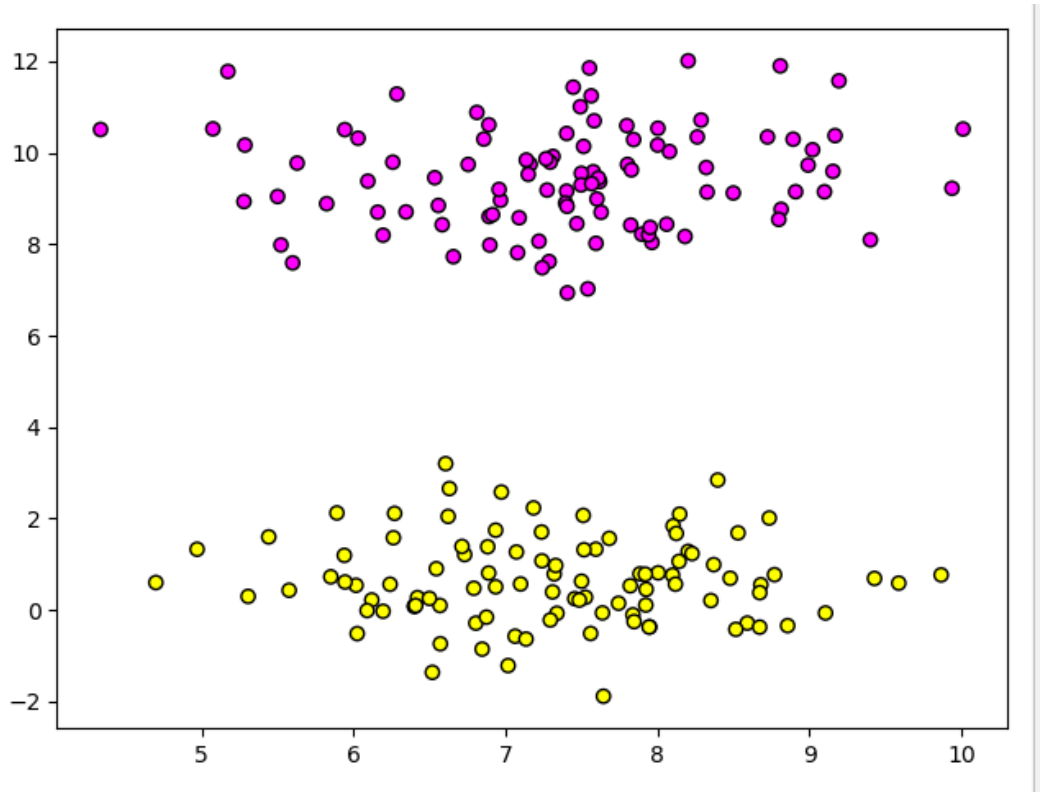
可以通过sicikit-learn进行手动生成数据集来使用K最近邻算法进行分类。

```
from sklearn.datasets import make_blobs
# KNN 分类器
import matplotlib.pyplot as plt
data = make_blobs(n_samples=200, centers=2, random_state=8)
x, y = data
plt.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.spring, edgecolors="k")
plt.show()
```

使用`sklearn`中的`make_blobs`函数来进行生成数量为200的样本数据，将数据进行分成两类的数据集，然后将数据集赋值给 `x y`然后通过`matplotlib`将数据图形进行展示。

`make_blobs` 函数是为聚类产生数据集，主要参数作用如下：

- n_samples 生成的样本数量
- n_features 每个样本的特征数
- centers 生成样本数量的分类数量
- cluster_std 每个类别的平方差
- random_state 随机种子



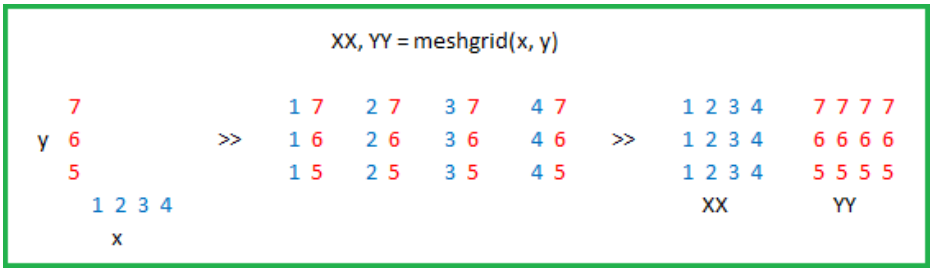
上图可以看出通过`make_blobs`生成的数据分为两类，我们可以将这些数据用于算法进行模型的训练，然后针对新的或未知的数据分类或者回归。

接下来通过K最近邻算法来进行拟合这些数据：

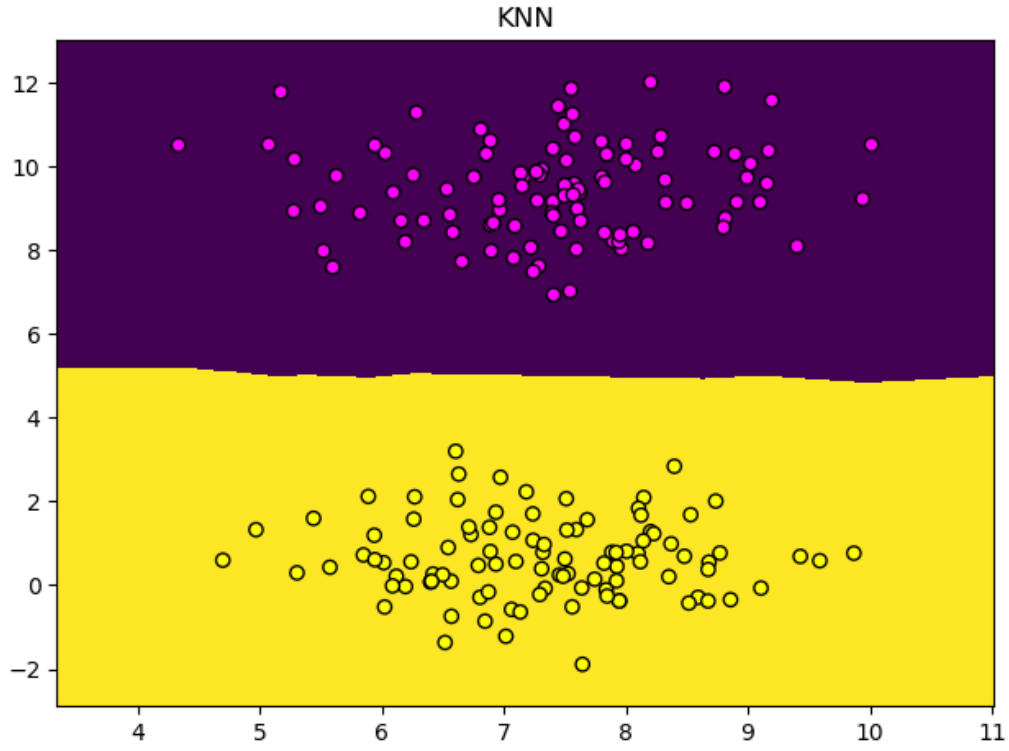
```
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import numpy as np

# 创建数据集
data = make_blobs(n_samples=200, centers=2, random_state=8)
x, y = data
# 创建KNN分类器
clf = KNeighborsClassifier()
clf.fit(x, y)
# 绘制图
x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z)
plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.spring, edgecolors="k")
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("KNN")
plt.show()
```

其中`np.meshgrid`用于生成网格坐标矩阵，具体生成方式如下图：



执行代码，会出现以下结果：

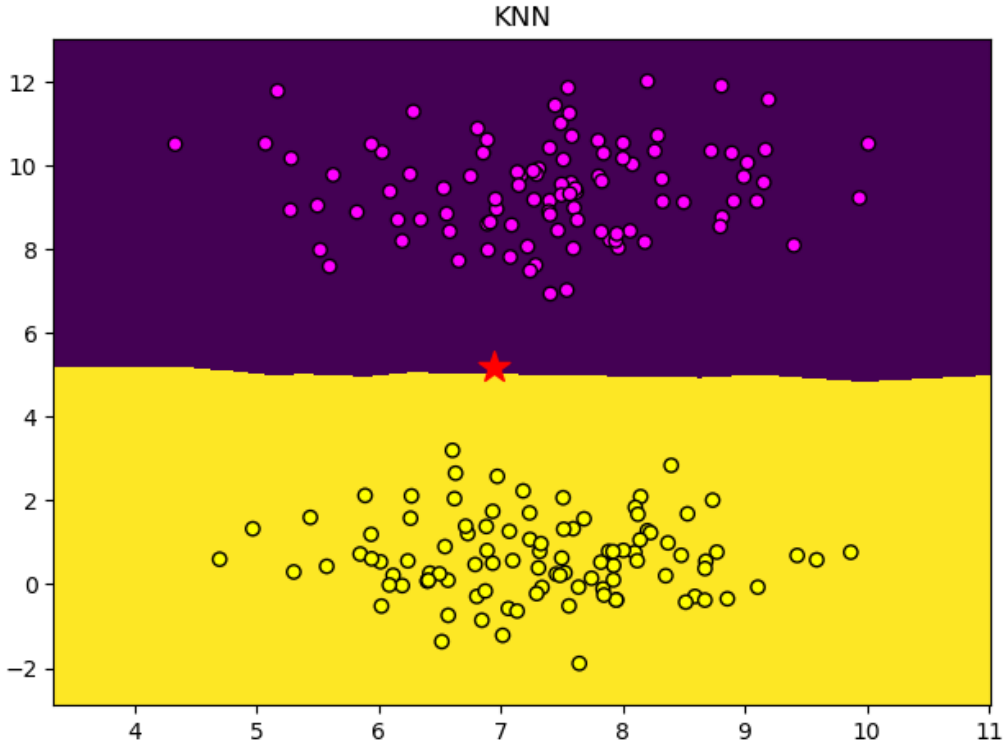


通过上述代码可以看到K最近邻算法基于生成的数据集创建了一个分类模型，将数据分成两部分使用不同的颜色进行区分，如果有新的数据进行输入，模型会自动将新数据进行分类到对应的类别中。

假设有一个新的数据点，特征值为`6.95`和`5.2`，进行试验模型是否可以将新的数据进行正确的分类，首先我们可以在`plt.show()`之前加入以下代码：

```
plt.scatter(6.95, 5.2, marker="*", c='red', s=200)
```

再次运行程序，会出现下图：



图中★为新的数据位置，可以看到K最近邻算法将它放在了上方的区域，与上方的区域归为了一类。

通过代码进行验证：

```
print('='*30)
print('新的数据点为：（6.95，5.2），分类结果为：', clf.predict([[6.95, 5.2]]))
print('='*30)
```

运行结果：

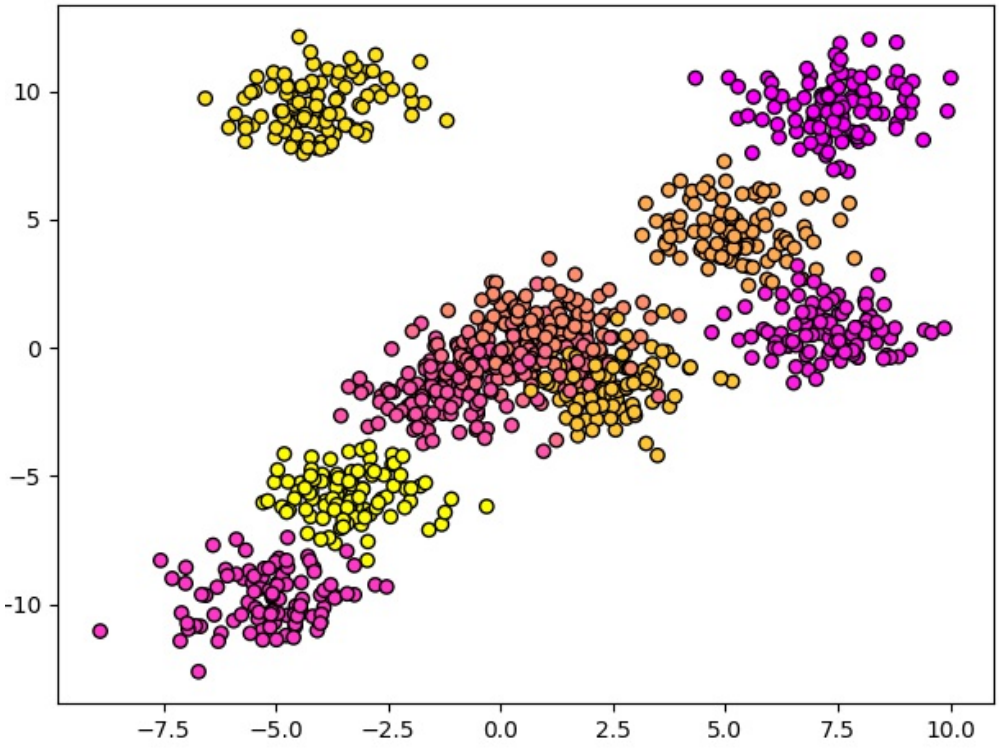
```
==============================
新的数据点为：（6.95，5.2），分类结果为： [0]
==============================
```

多元分类

接下来我们来处理多元分类，为了将难度加大，修改`make_blobs`中的`centers`参数生成1,000个数据样本，数据类型扩展到10类：

```
from sklearn.datasets import make_blobs
# KNN 分类器
import matplotlib.pyplot as plt
data = make_blobs(n_samples=1000, centers=10, random_state=8)
x, y = data
plt.scatter(x[:,0], x[:,1], c=y, cmap=plt.cm.spring, edgecolors="k")
plt.show()
```

运行程序，生成我们生成数据集展示图：



可以看到新的数据被分成了10类，其中有3类数据有一些重合，然后通过K最近邻算法进行拟合数据：

```
from sklearn.datasets import make_blobs
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import numpy as np

# 创建数据集
data = make_blobs(n_samples=1000, centers=10, random_state=8)
x, y = data
# 创建KNN分类器
clf = KNeighborsClassifier()
clf.fit(x, y)
# 绘制图
x_min, x_max = x[:, 0].min() - 1, x[:, 0].max() + 1
y_min, y_max = x[:, 1].min() - 1, x[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))
z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
z = z.reshape(xx.shape)
plt.pcolormesh(xx, yy, z)
plt.scatter(x[:, 0], x[:, 1], c=y, cmap=plt.cm.spring, edgecolors="k")
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("KNN")
plt.show()
```

运行程序会出现K最近邻算法拟合结果：

可以看到K最近邻算法将大部分的数据点分类到正确的分类中，单有一部分小数据分入了错误的分类中，分类错误的数据基本为重合的数据点。

我们可以通过代码计算K最近邻算法的正确率：

```
print('=' * 30)
print("模型正确率：{:2f}".format(clf.score(x,y)))
print('=' * 30)
```

```
==============================
模型正确率：0.917000
==============================
```

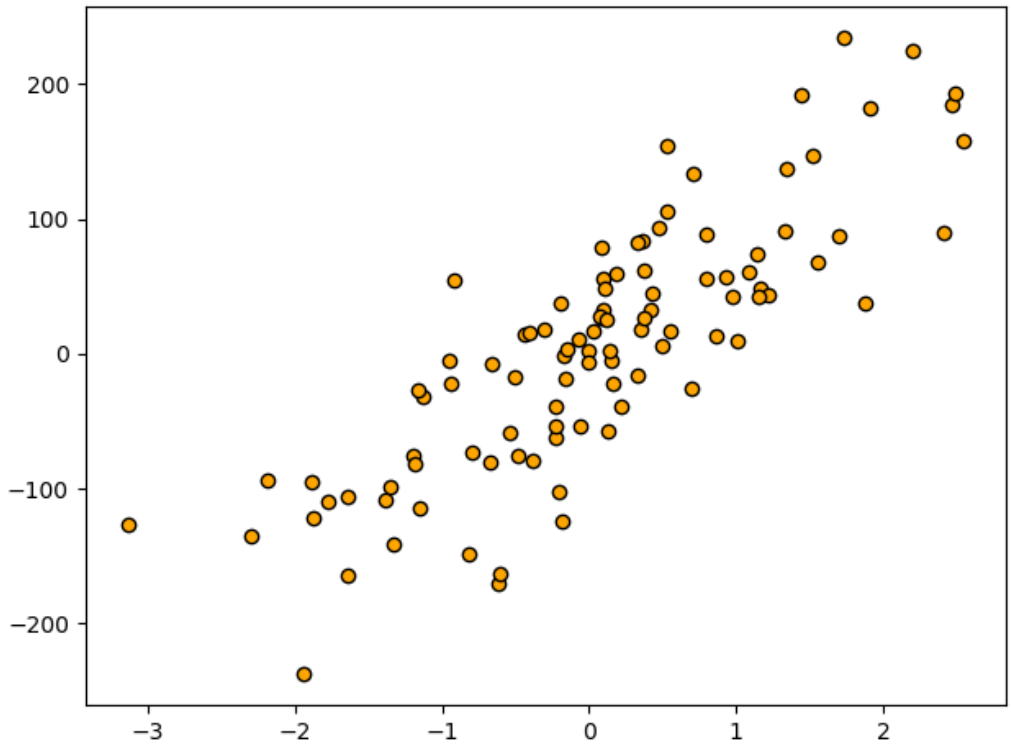
可以看出将数据集样本提升到1,000条，分类提升到10中数据类型，模型的正确率为`91.7%`。

回归分析

通过scikit-learn中的`make_regression`进行生成用于回归分析的数据集。

```
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
x, y = make_regression(n_features=1, n_informative=1, noise=50, random_state=8)
plt.scatter(x, y, c='orange', edgecolors="k")
plt.show()
```

将样本的特征数量为1个，噪音为50，然后运行程序。

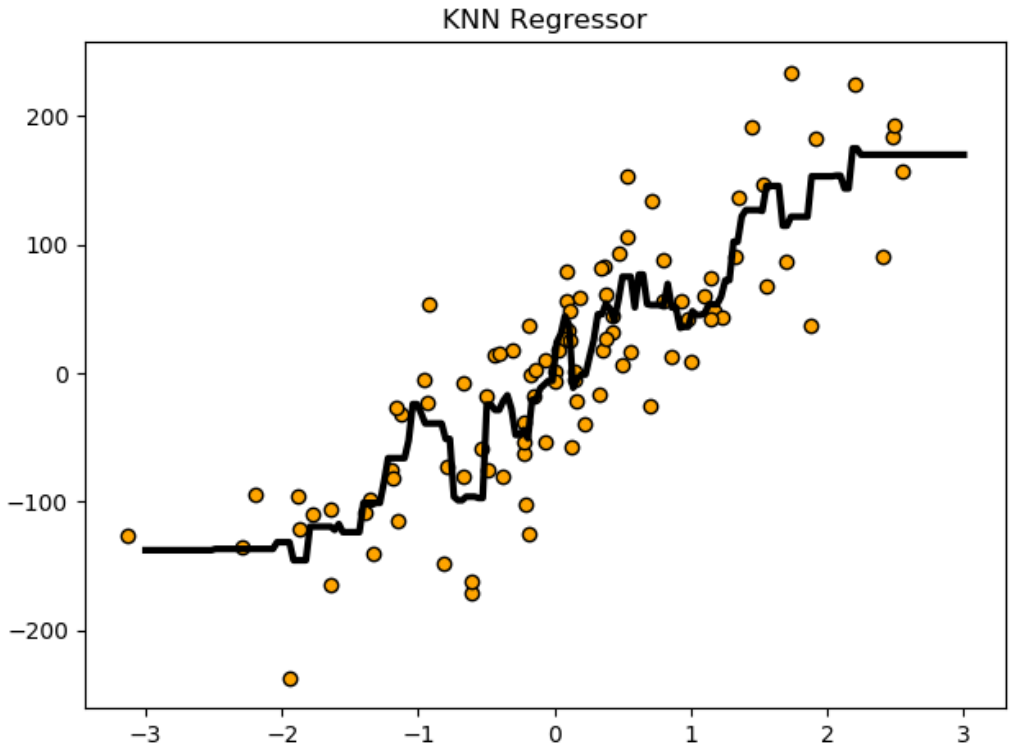


上图可以看到横轴（X轴）为样本的特征数值，大致范围在-3~3之间；纵轴（Y轴）为样本的测定值，大致范围在-250~250之间。

然后通过K最近邻算法来进行回归测试：

```
from sklearn.datasets import make_regression
# KNN 回归模型
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
import numpy as np
x, y = make_regression(n_features=1, n_informative=1, noise=50, random_state=8)
reg = KNeighborsRegressor()
# 使用 KNN 模型进行拟合数据
reg.fit(x, y)
z = np.linspace(-3, 3, 200).reshape(-1, 1)
plt.scatter(x, y, c='orange', edgecolors="k")
plt.plot(z, reg.predict(z), c='k', linewidth=3)
plt.title('KNN Regressor')
plt.show()
```

运行程序，生成拟合结果。



从图中可以看出K最近邻算法拟合生成的数据，直观来看拟合结果并不是很好，有大量的数据点没有被覆盖到。

通过`score`来进行计算模型分数：

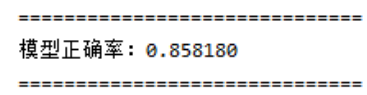
```
print('=' * 30)
print("模型正确率：{:2f}".format(reg.score(x, y)))
print('=' * 30)
```

```
==============================
模型正确率：0.772117
==============================
```

看也看到模型的正确率只有`77.2%`，该结果比较差。为了提高正确率可以进行降低K最近邻算法的`n_neighbors`来提高准确率。

```
from sklearn.datasets import make_regression
# KNN 回归模型
from sklearn.neighbors import KNeighborsRegressor
import matplotlib.pyplot as plt
import numpy as np
x, y = make_regression(n_features=1, n_informative=1, noise=50, random_state=8)
reg = KNeighborsRegressor(n_neighbors=2)
reg.fit(x, y)
z = np.linspace(-3, 3, 200).reshape(-1, 1)
plt.scatter(x, y, c='orange', edgecolors="k")
plt.plot(z, reg.predict(z), c='k', linewidth=3)
plt.title('KNN Regressor')
plt.show()
print('=' * 30)
print("模型正确率：{:2f}".format(reg.score(x, y)))
print('=' * 30)
```

代码通过将K最近邻算法的`n_neighbors`调整到2，运行程序，生成拟合结果以及模型争取率。



可以看到相对第一次来说覆盖的数据点变多，准确率从`77.2%`提升到`85.8%`。

K最近邻算法实战-酒分类

数据集分析

数据集来自scikit-learn中内置的数据集来进行试验，该数据中在`datasets`模块中。

首先加载酒数据集到项目中：

```
from sklearn.datasets import load_wine
win_dataset = load_wine()
print(win_dataset.keys())
print(win_dataset['DESCR'])
print(win_dataset['data'].shape)
```

运行代码可以看到加载的数据。

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
        - Alcohol
        - Malic acid
        - Ash
        - Alcalinity of ash
        - Magnesium
        - Total phenols
        - Flavanoids
        - Nonflavanoid phenols
        - Proanthocyanins
        - Color intensity
        - Hue
        - OD280/OD315 of diluted wines
        - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:
```

可以看到数据集中主要有`'data', 'target', 'target_names', 'DESCR', 'feature_names'`，对应`数据,分类目标,分类名称,描述,特征名称`，以及178个样本数据，每条样本数据中有13个特征，主要分为3类（`class_0,class_1,class_2`），其中`class_0`有59个样本，`class_1`有71个样本，`class_2`中有48个样本。
特征主要包括酒精含量、苹果酸、镁含量、青花素含量、色彩饱和度等等。

生成训练集和测试集

创建一个自动将酒分类的机器算法模型之前，需要能够对模型的可信度进行评判，否则我们无法知道针对新的酒进行的分类是否正确。

所以我们需要将数据集拆分为两部分：一部分称为训练集；另一部分为测试集。

使用scikit-learn中的`train_test_split`函数进行拆分数据集，该函数可以将数据集进行随机排列，默认情况下75%的数据以及对应的标签划分到训练集，25%的数据集划分到测试集中。

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
win_dataset = load_wine()
X_train, X_test, y_train, y_test = train_test_split(win_dataset['data'], win_dataset['target'], random_state=0
print('X_train shape:{}'.format(X_train.shape))
print('X_test shape:{}'.format(X_test.shape))
print('y_train shape:{}'.format(y_train.shape))
print('y_test shape:{}'.format(y_test.shape))
```

```
X_train shape:(133, 13)
X_test shape:(45, 13)
y_train shape:(133,)
y_test shape:(45,)
```

可以看出在样本总数量178个中训练集以及对应的标签有133个约占（74.7%），测试集以及标签共有45个，约占（25.3%），特征数量为13个。

建模

使用scikit-learn的K最近邻算法进行分类，然后基于训练数据集进行训练建模，在训练数据集中进行寻找和新输入的数据最近的数据点，然后将这个数据点的标签分配给新的数据点，以此对新的样本进行分类。

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
```

导入KNN分类模型，然后将`n_neighbors`参数值设置为1，`n_neighbors`为K最近邻算法中的近邻数据。

然后通过`knn`来进行“拟合（`fit`）”的方法来进行建模，建模的依据就是训练集中样本数据`X_train`以及对应的`y_train`：

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
win_dataset = load_wine()
X_train, X_test, y_train, y_test = train_test_split(win_dataset['data'], win_dataset['target'], random_state=0
knn.fit(X_train,y_train)
print(knn)
```

新样本分类预测

测试集样本分类

将训练好的模型用于进行训练测试集样本的准确率，来针对模型进行打分：

```
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
win_dataset = load_wine()
X_train, X_test, y_train, y_test = train_test_split(win_dataset['data'], win_dataset['target'], random_state=0
knn.fit(X_train,y_train)
print(knn)
print('测试集样本预测', knn.score(X_test, y_test))
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
测试集样本预测 0.7555555555555555
```

可以看到模型针对测试集样本的预测正确的概率为75.5%。

新样本预测

假设目前有一瓶新的红酒以及对应的特征进行使用模型预测：

```
X_new = np.array([[13.2, 2.77, 2.51, 18.5, 96.6, 1.04, 2.55, 0.57, 1.47, 6.2, 1.05, 3.33, 820]])
production = knn.predict(X_new)
print('预测红酒的类别为', win_dataset['target_names'][production])
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
预测红酒的类别为 ['class_2']
```