KNN regression-weights

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
data = pd.read_csv(r"iris.csv")
# 删除不需要的列：Unnamed:0,Species
data.drop(["Unnamed: 0","Species"],axis=1,inplace=True)
# 删除重复的记录
data.drop_duplicates(inplace=True)
data.head()
```

Out[2]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [3]:
```
class KNN:
    """
    使用python 实现K近邻算法（回归预测）
    根据前3个特征属性，寻找最近的k个邻居，再根据k个邻居的第4个特征属性，预测当前样
    """
    def __init__(self,k):
        """初始化方法
        Parameters
        ------
        k:int
          邻居的个数
        """
        self.k = k
    def fit(self,X,y):
        """训练方法
        Parameters
        ------
        X:类数组类型（特征矩阵），形状为[样本数量,特征数量]
          待训练的样本特征（属性）
        y:类数组类型（目标标签），形状为[样本数量]
          每个样本的目标值（标签）
        """
        self.X = np.asarray(X)
        self.y = np.asarray(y) # 将X与y转换成 ndarray数组形式，方便统一进行操作
    def predict2(self,X):
        """根据参数传递的X，对样本数据进行预测(考虑权重weights:每个节点（邻居）距
        Parameters:
        ------
        X:类数组类型，形状为[样本数量，特征数量]
          待测试的样本特征（属性）
        Returns
        ------
        result:数组类型
               预测的结果值
```

```
        """
        # 转换成数组类型
        X = np.asarray(X)
        result = [] # 保存预测的结果值
        for x in X:
            # 计算与训练集中每个X的距离
            dis = np.sqrt(np.sum((x-self.X) ** 2, axis=1))
            # 返回数组排序后，每个元素在原数组中（排序之前的数组）的索引
            index = dis.argsort()
            # 取原数组中前k个距离最近的索引
            index = index[:self.k]
            #求所有节点（邻居）距离的倒数之和[注意：最后加上一个很小的值，是为避免
            s = np.sum(1/(dis[index] + 0.001))
            # 使用每个节点的倒数，再除以倒数之和，得到权重
            weight = (1/(dis[index] + 0.001)) / s
            # 使用邻居节点的标签值，乘以对应的权重，然后相加，得到最终的预测结果
            result.append(np.sum(self.y[index] * weight))
        return np.array(result)
```

In [4]:
```
t = data.sample(len(data), random_state=0)
train_X = t.iloc[:120,:-1]
train_y = t.iloc[:120,-1]
test_X = t.iloc[120:,:-1]
test_y = t.iloc[120:,-1]
knn = KNN(k=3) # 选择3个邻居
knn.fit(train_X,train_y) # 训练
result = knn.predict2(test_X)
display(result)
np.mean(np.sum((result-test_y) ** 2))
display(test_y.values)
```

```
array([0.2       , 2.06034623, 0.2       , 1.92517496, 1.27413603,
       1.19618445, 1.23000185, 2.04006363, 1.12909093, 1.93134757,
       2.02753182, 1.85536796, 1.81368336, 0.2       , 1.1368278 ,
       2.23590685, 1.59999864, 0.28797364, 1.47134903, 1.25587923,
       1.6919192 , 1.39627809, 0.27823376, 0.24368835, 0.2       ,
       2.0714158 , 1.25481575, 2.14183025, 0.22616013])
array([0.2, 1.6, 0.2, 2.3, 1.3, 1.2, 1.3, 1.8, 1. , 2.3, 2.3, 1.5, 1.7,
       0.2, 1. , 2.1, 2.3, 0.2, 1.3, 1.3, 1.8, 1.3, 0.2, 0.4, 0.1, 1.8,
       1. , 2.2, 0.2])
```

In [5]:
```
result = knn.predict2(test_X)
display(np.mean(np.sum((result-test_y) ** 2)))
```

```
1.55205592089719
```

In [6]:
```
# 可视化
import matplotlib as mpl
import matplotlib.pyplot as plt
mpl.rcParams["font.family"] = "SimHei"
mpl.rcParams["axes.unicode_minus"] = False
```

In [7]:
```
plt.figure(figsize=(10,10))
# 绘制预测值
plt.plot(result,"ro-",label="预测值")
# 绘制真实值
plt.plot(test_y.values,"go--",label="真实值")
plt.title("KNN连续预测展示")
plt.xlabel("节点序号")
plt.ylabel("花瓣宽度")
```

```
plt.legend()
plt.show()
```