

# 第二章 k-近邻算法

## 一、sklearn简介

### (一) sklearn简要介绍

scikit-learn, 又写作sklearn, 是一个开源的基于python语言的机器学习工具包. 它通过NumPy, SciPy和Matplotlib等python数值计算的库封装了机器学习中常用的算法, 包括监督学习、非监督学习等, 并且涵盖了几乎所有主流机器学习算法.

SKlearn官网: <https://scikit-learn.org>

sklearn中文文档: <https://sklearn.apachecn.org/#/>

在工程应用中, 用python手写代码来从头实现一个算法的可能性非常低, 这样不仅耗时耗力, 还不一定能够写出构架清晰, 稳定性强的模型. 更多情况下, 是分析采集到的数据, 根据数据特征选择适合的算法, 在工具包中调用算法, 调整算法的参数, 获取需要的信息, 从而实现算法效率和效果之间的平衡. 而sklearn, 正是这样一个可以帮助我们高效实现算法应用的工具包.

### (二) sklearn中主要模块

sklearn中常用的模块有**分类、回归、聚类、降维、模型选择、预处理**等

- **分类**: 识别某个对象属于哪个类别, 常用的算法有: **SVM** (支持向量机)、**nearest neighbors** (最近邻)、**random forest** (随机森林), 常见的应用有: 垃圾邮件识别、图像识别.
- **回归**: 预测与对象相关联的连续值属性, 常用的算法有: **SVR** (支持向量回归)、**ridge regression** (岭回归)、**Lasso**, 常见的应用有: 药物反应, 预测股价.
- **聚类**: 将相似对象自动分组, 常用的算法有: **k-Means** (k-均值)、**spectral clustering** (谱聚类)、**mean-shift** (均值漂移), 常见的应用有: 客户细分, 分组实验结果.
- **降维**: 减少要考虑的随机变量的数量, 常用的算法有: **PCA** (主成分分析)、**feature selection** (特征选择)、**non-negative matrix factorization** (非负矩阵分解), 常见的应用有: 可视化, 提高效率.
- **模型选择**: 比较, 验证, 选择参数和模型, 常用的模块有: **grid search** (网格搜索)、**cross validation** (交叉验证)、**metrics** (度量). 它的目标是通过参数调整提高精度.
- **预处理**: 特征提取和归一化, 常用的模块有: **preprocessing** (预处理), **feature extraction** (特征抽取), 常见的应用有: 把输入数据 (如文本) 转换为机器学习算法可用的数据.

### (三) sklearn使用流程(简化版)

总结起来最主要的是以下三个:

- 首先引入训练的方法, 比如k-近邻, 决策树, 等等
- 然后用训练数据进行训练
- 最后用测试数据进行预测

用一个简单的伪代码表示k-近邻算法的大致流程,如下:

```
import numpy as np
from sklearn.neighbors import KNeighborsClassifier as knn#k-近邻算法

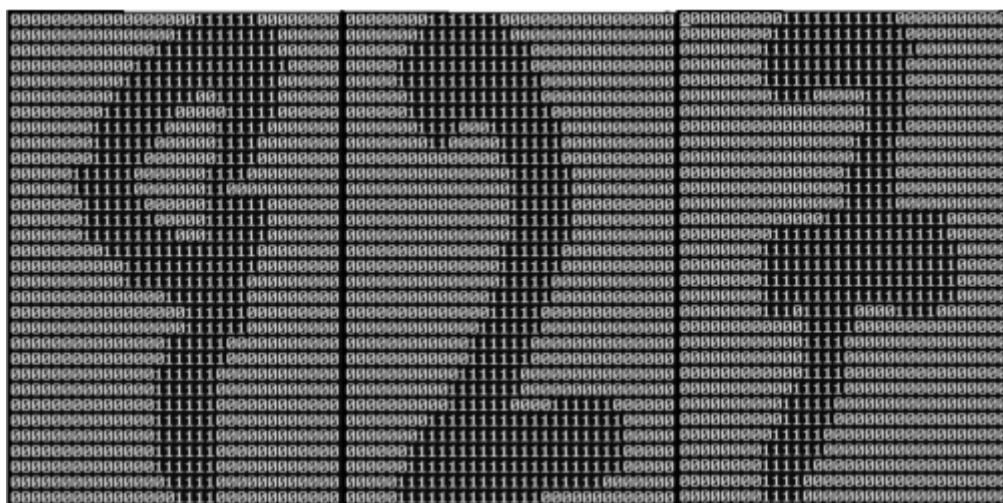
neigh = knn()#这里面有参数可以设置，也可以默认
neigh.fit(X_train,y_train)#用训练数据进行训练或者叫拟合

neigh.predict(X_test)#用测试数据进行预测
```

## 二 k-近邻算法之手写数字识别

### (一) 利用自行构建的knn进行识别

实际图像存储在第2章源代码的两个子目录内: 目录trainingDigits中包含了大约2000个例子, 每个例子的内容下图所示, 每个数字大约有200个样本; 目录testDigits中包含了大约900个测试数据. 我们使用目录trainingDigits中的数据训练分类器, 使用目录testDigits中的数据测试分类器的效果.



### 1 将图像格式处理为一个向量

为了使用前面两个例子的分类器, 我们必须将图像格式化处理为一个向量. 我们将把一个32×32的二进制图像矩阵转换为1×1024的向量, 这样前两节使用的分类器就可以处理数字图像信息了.

先编写一段函数img2vector, 将图像转换为向量: 该函数创建1×1024的NumPy数组, 然后打开给定的文件, 循环读出文件的前32行, 并将每行的头32个字符值存储在NumPy数组中, 最后返回数组.

创建kNN\_Hsu05.py, 在py文件中新建im2vector函数, 具体代码如下:

```
import numpy as np
.....

函数说明: 将32x32的二进制图像转换称1x1024向量
```

```

Parameters:
    filename - 文件名
Returns:
    returnVect - 返回的二进制图像的1x1024向量
"""
def img2vector(filename):
    #创建1x1024零向量
    returnVect = np.zeros((1,1024))
    #打开文件
    with open(filename) as fr:
        #按行读取, 注意readlines是读取所有行, readline是读取一行
        for i in range(32):
            #读取一行数据(readline是每一次读取一行)
            lineStr = fr.readline()
            #每一行的前32个元素依次添加到returnVect中
            for j in range(32):
                #returnVect只有一行, 也就是0, i和j都是从0开始, 31结束, 共32
                returnVect[0, 32*i+j] = int(lineStr[j])
    #返回转换后的1x1024向量
    return returnVect

if __name__ == '__main__':
    filename = '0_3.txt'
    returnVect = img2vector(filename)
    print(returnVect)
    print(returnVect.shape)

```

结果如下:

In [3]: `import numpy as np`

"""

函数说明:将32x32的二进制图像转换称1x1024向量

Parameters:

filename - 文件名

Returns:

returnVect - 返回的二进制图像的1x1024向量

"""

```
def img2vector(filename):
    #创建1x1024零向量
    returnVect = np.zeros((1,1024))
    #打开文件
    with open(filename) as fr:
        #按行读取, 注意readlines是读取所有行, readline是读取一行
        for i in range(32):
            #读取一行数据(readline是每一次读取一行)
            lineStr = fr.readline()
            #每一行的前32个元素依次添加到returnVect中
            for j in range(32):
                #returnVect只有一行, 也就是0, i和j都是从0开始, 31结束, 共32
                returnVect[0,32*i+j] = int(lineStr[j])
    #返回转换后的1x1024向量
    return returnVect

if __name__ == '__main__':
    filename = '0_3.txt'
    returnVect = img2vector(filename)
    print(returnVect)
    print(returnVect.shape)
```

```
[[0. 0. 0. ... 0. 0. 0.]]
```

```
(1, 1024)
```

## 2 测试算法: 使用 k-近邻算法识别手写数字

上面的img2vector函数是处理一个txt文件, 下面要处理'trainDigits'文件夹下和'testDigits'文件夹下的所有txt文件.

在kNN\_Hsu05.py中新建handwritingClassTest函数, 并将前面的classify0函数和img2vector加入进去, 具体代码如下:

```
import numpy as np
import operator
from os import listdir

"""
函数说明: KNN算法, 分类器

Parameters:
    inX - 用于分类数据(即测试集)
    dataSet - 用于训练的数据(即训练集)
    labels - 分类的标签(即什么类型的电影)
    k - KNN算法参数, 选择距离最小的k个点
Returns:
    sortedClassCount[0][0] - 分类结果
"""
```

```

def classify0(inX, dataSet, labels, k):
    #shape[0]返回的是dataSet的函数
    dataSetSize = dataSet.shape[0]
    #np.tile(a,(b,c))函数含义是在列向量方向上重复a共b次(横向), 在行向量方向上重复a共c次(纵向)
    #np.tile(inX, (dataSetSize,1))的含义就是构建dataSetSize行inX,接着减去DataSet对应的.
    diffMat = np.tile(inX, (dataSetSize,1)) - dataSet
    #求得每一行差额的平方
    sqDiffMat = diffMat**2
    #然后求和, axis=1是按行求和,axis=0是按列求和
    sqDistances = sqDiffMat.sum(axis=1)
    #开平方,求得距离
    distances = sqDistances**0.5
    #numpy中的argsort()方法(也是numpy的顶级函数),返回distances中元素从小到大排列后的索引值.等价:a.argsort()==np.argsort(a)
    sortedDistIndices = distances.argsort()
    #构建一个空的字典,用于记录类别次数
    classCount = {}
    for i in range(k):
        #提取出前k个元素的类别
        voteIlabel = labels[sortedDistIndices[i]]
        #字典的get方法,返回指定键的值,如果指定键不存在,那么返回默认值,这里是0.
        #比如键是"爱情片",最开始字典是空的,那么返回0,后面加了1,那么就是{"爱情片":1}
        #接着如果还是"爱情片",那么返回1,再加1,就是2,那么字典变成了{"爱情片":2}
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    #sorted函数,里面有参数key,根据一定的方式排序.这里的key=operator.itemgetter(1)表示的是根据字典的值进行排序
    #参数reverse表示是否进行降序排序,True表示是,默认否,即false
    sortedClassCount = sorted(classCount.items(),
key=operator.itemgetter(1),reverse=True)
    #则第一个字典的键就是[0][0],也就是返回次数最多的类别,就是我们要的最终类别
    return sortedClassCount[0][0]

```

"""

函数说明:将32x32的二进制图像转换称1x1024向量

Parameters:

filename - 文件名

Returns:

returnVect - 返回的二进制图像的1x1024向量

"""

```

def img2vector(filename):
    #创建1x1024零向量
    returnVect = np.zeros((1,1024))
    #打开文件
    with open(filename) as fr:
        #按行读取,注意readlines是读取所有行,readline是读取一行
        for i in range(32):
            #读取一行数据(readline是每一次读取一行)
            lineStr = fr.readline()
            #每一行的前32个元素依次添加到returnVect中
            for j in range(32):
                #returnVect只有一行,也就是0,i和j都是从0开始,31结束,共32
                returnVect[0,32*i+j] = int(lineStr[j])
    #返回转换后的1x1024向量
    return returnVect

```

```
"""
```

函数说明:手写数字分类测试

Parameters:

无

Returns:

无

```
"""
```

```
def handwritingClassTest():
```

```
    #测试集的Labels
```

```
    hwLabels = []
```

```
    #返回训练文件夹trainingDigits目录下的文件名, listdir返回的是列表
```

```
    trainingFileList = listdir('trainingDigits')
```

```
    #返回训练文件夹下文件的个数
```

```
    m = len(trainingFileList)
```

```
    #初始化训练的矩阵
```

```
    trainingMat = np.zeros((m,1024))
```

```
    #从文件名中解析出训练集的分类
```

```
    for i in range(m):
```

```
        #获取文件的名字
```

```
        filenameStr = trainingFileList[i]
```

```
        #获得分类的数字,因为文件名是'真实数字_文件序列号.txt',所以[0]就是真实数字
```

必要.

```
        #书上多了一步:fileStr = fileNameStr.split('.')[0], 这步是获取数字如:0_3, 没有
```

```
        classNumStr = int(filenameStr.split('_')[0])
```

```
        #将获取的真实数字(也就是真实类别)添加到hwLabels中
```

```
        hwLabels.append(classNumStr)
```

```
        #将每个文件的1x1024数据存储在trainingMat矩阵中,trainingMat循环完成后,共 mx1024
```

```
        trainingMat[i,:] = img2vector('trainingDigits/%s' % (filenameStr))
```

```
    #返回测试文件夹testDigits目录下的文件列表
```

```
    testFileList = listdir('testDigits')
```

```
    #测试集数据的数量
```

```
    mTest = len(testFileList)
```

```
    #错误检测计数
```

```
    errorCount = 0
```

```
    #从文件名中解析出测试集的分类进行分类测试
```

```
    #这部分的代码类似于上面训练集中的代码
```

```
    for i in range(mTest):
```

```
        #获取文件的名字
```

```
        filenameStr = testFileList[i]
```

```
        #获得分类的数字,因为文件名是'真实数字_文件序列号.txt',所以[0]就是真实数字
```

```
        classNumStr = int(filenameStr.split('_')[0])
```

```
        #获取测试集的1x1024向量,就一个1x1024,用于预测最后的数字
```

```
        vectorUnderTest = img2vector('testDigits/%s' % (filenameStr))
```

```
        #获得预测结果
```

```
        classifierResult = classify0(vectorUnderTest,trainingMat,hwLabels,3)
```

```
        print("分类的结果为%d\t真实的结果为%d" % (classifierResult, classNumStr))
```

```
        if classifierResult != classNumStr:
```

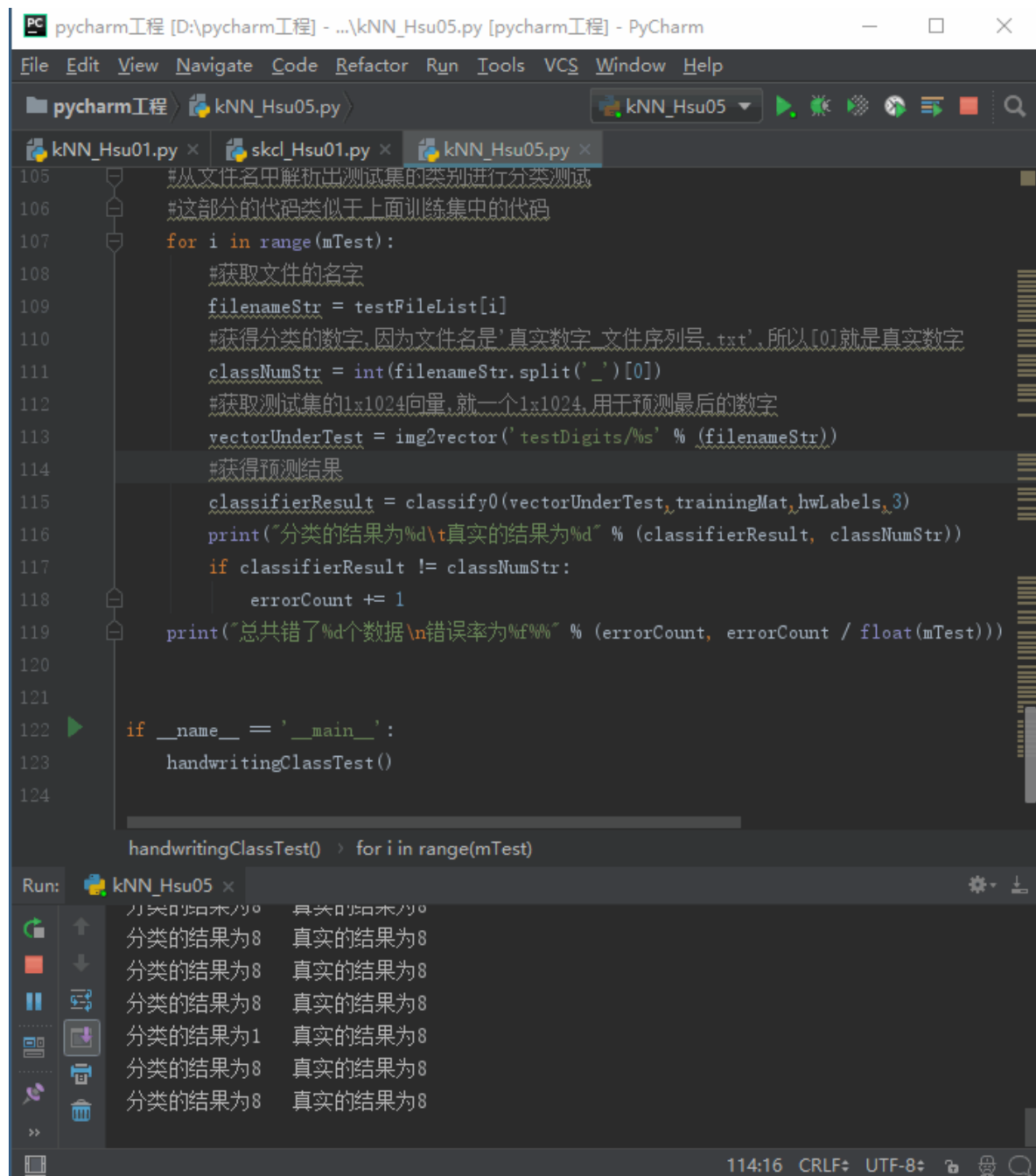
```
            errorCount += 1
```

```
    print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount /
float(mTest)))
```

```
if __name__ == '__main__':
```

```
    handwritingClassTest()
```

结果如下:



```
pycharm工程 [D:\pycharm工程] - ...\kNN_Hsu05.py [pycharm工程] - PyCharm
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pycharm工程 > kNN_Hsu05.py > kNN_Hsu05

kNN_Hsu01.py x skcl_Hsu01.py x kNN_Hsu05.py x
105 #从文件名中解析出测试集的分类进行测试
106 #这部分代码类似于上面训练集中的代码
107 for i in range(mTest):
108     #获取文件的名称
109     filenameStr = testFileList[i]
110     #获得分类的数字, 因为文件名是'真实数字_文件序列号.txt', 所以[0]就是真实数字
111     classNumStr = int(filenameStr.split('_')[0])
112     #获取测试集的1x1024向量, 就一个1x1024, 用于预测最后的数字
113     vectorUnderTest = img2vector('testDigits/%s' % (filenameStr))
114     #获得预测结果
115     classifierResult = classify0(vectorUnderTest, trainingMat, hwLabels, 3)
116     print("分类的结果为%d\t真实的结果为%d" % (classifierResult, classNumStr))
117     if classifierResult != classNumStr:
118         errorCount += 1
119 print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount / float(mTest)))
120
121
122 if __name__ == '__main__':
123     handwritingClassTest()
124

handwritingClassTest() > for i in range(mTest)

Run: kNN_Hsu05 x
分类的结果为8 真实的结果为8
分类的结果为8 真实的结果为8
分类的结果为8 真实的结果为8
分类的结果为1 真实的结果为8
分类的结果为8 真实的结果为8
分类的结果为8 真实的结果为8
```

这段代码需要注意的有:

1. `listdir('a')`是返回a文件夹下的所有文件名, 以字符串的形式存储在列表中, 如:



```

In [6]: from os import listdir
In [7]: a = listdir('机器学习实战')
In [8]: a
Out[8]:
['.ipynb_checkpoints',
 'kNN_Hsu.ipynb',
 'kNN_Hsu01.py',
 'kNN_Hsu02.py',
 'kNN_Hsu02_1.py',
 'kNN_Hsu02_2.py',
 'kNN_Hsu05.py',
 'kNN_Hsu_04.py',
 'pycharm工程',
 'skcl_Hsu01.py',
 '机器学习实战相关书籍']
In [9]: type(a)
Out[9]: list

```

## (二) 利用sklearn进行手写识别

### 1 sklearn中k-近邻算法简介

sklearn.neighbors模块中可以实现k-近邻算法, 具体内容如下图

#### sklearn.neighbors: Nearest Neighbors

The `sklearn.neighbors` module implements the k-nearest neighbors algorithm.

**User guide:** See the [Nearest Neighbors](#) section for further details.

<code>neighbors.BallTree</code>	BallTree for fast generalized N-point problems
<code>neighbors.DistanceMetric</code>	DistanceMetric class
<code>neighbors.KDTree</code>	KDTree for fast generalized N-point problems
<code>neighbors.KernelDensity</code> ([bandwidth, ...])	Kernel Density Estimation
<code>neighbors.KNeighborsClassifier</code> ([...])	Classifier implementing the k-nearest neighbors vote.
<code>neighbors.KNeighborsRegressor</code> ([n_neighbors, ...])	Regression based on k-nearest neighbors.
<code>neighbors.LocalOutlierFactor</code> ([n_neighbors, ...])	Unsupervised Outlier Detection using Local Outlier Factor (LOF)
<code>neighbors.RadiusNeighborsClassifier</code> ([...])	Classifier implementing a vote among neighbors within a given radius
<code>neighbors.RadiusNeighborsRegressor</code> ([radius, ...])	Regression based on neighbors within a fixed radius.
<code>neighbors.NearestCentroid</code> ([metric, ...])	Nearest centroid classifier.
<code>neighbors.NearestNeighbors</code> ([n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
<code>neighbors.NeighborhoodComponentsAnalysis</code> ([...])	Neighborhood Components Analysis
<code>neighbors.kneighbors_graph</code> (X, n_neighbors[, ...])	Computes the (weighted) graph of k-Neighbors for points in X
<code>neighbors.radius_neighbors_graph</code> (X, radius)	Computes the (weighted) graph of Neighbors for points in X

使用sklearn.neighbors.KNeighborsClassifier就可以实现之前的k-近邻.

sklearn.neighbors.KNeighborsClassifier一共有8个参数, 具体如下:



## sklearn.neighbors.KNeighborsClassifier ¶

```
class sklearn.neighbors. KNeighborsClassifier (n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30,
p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs) \[source\]
```

Classifier implementing the k-nearest neighbors vote.

Read more in the [User Guide](#).

**Parameters:** **n\_neighbors** : *int, optional (default = 5)*

Number of neighbors to use by default for `kneighbors` queries.

**weights** : *str or callable, optional (default = 'uniform')*

weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

**algorithm** : *{'auto', 'ball\_tree', 'kd\_tree', 'brute'}, optional*

Algorithm used to compute the nearest neighbors:

- 'ball\_tree' will use `BallTree`
- 'kd\_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.

**leaf\_size** : *int, optional (default = 30)*

Leaf size passed to `BallTree` or `KDTree`. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

**p** : *integer, optional (default = 2)*

Power parameter for the Minkowski metric. When  $p = 1$ , this is equivalent to using `manhattan_distance` (l1), and `euclidean_distance` (l2) for  $p = 2$ . For arbitrary  $p$ , `minkowski_distance` (l\_p) is used.

**metric** : *string or callable, default 'minkowski'*

the distance metric to use for the tree. The default metric is `minkowski`, and with  $p=2$  is equivalent to the standard Euclidean metric. See the documentation of the `DistanceMetric` class for a list of available metrics.

**metric\_params** : *dict, optional (default = None)*

Additional keyword arguments for the metric function.

**n\_jobs** : *int or None, optional (default=None)*

The number of parallel jobs to run for neighbors search. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See [Glossary](#) for more details. Doesn't affect `fit` method.

各个参数具体的含义如下:

- **n\_neighbors**: 默认为5, 就是k-NN的k的值, 选取最近的k个点.
- **weights**: 默认是uniform, 参数可以是uniform、distance, 也可以是用户自己定义的函数.
  - uniform: 是均等的权重, 就说所有的邻近点的权重都是相等的.
  - distance: 是不均等的权重, 距离近的点比距离远的点的影响大.
  - 用户自定义的函数, 接收距离的数组, 返回一组维数相同的权重.
- **algorithm**: 快速k近邻搜索算法, 默认参数为auto, 可以理解为算法自己决定合适的搜索算法. 除此之外, 用户也可以自己指定搜索算法: ball\_tree、kd\_tree、brute方法进行搜索.
  - kd\_tree: 构造kd树存储数据以便对其进行快速检索的树形数据结构. kd树也就是数据结构中的二叉树. 以中值切分构造的树, 每个结点是一个超矩形, 在维数小于20时效率高.

- o ball\_tree: 是为了**克服kd树高维失效而发明的**，其构造过程是以质心 C 和半径 r 分割样本空间, 每个节点是一个超球体.
  - o brute: 是**蛮力搜索, 也就是线性扫描**, 当训练集很大时, 计算非常耗时.
- **leaf\_size**: **默认是30**, 这个是**构造的kd树和ball树的大小**. 这个值的设置会影响树构建的速度和搜索速度, 同样也影响着存储树所需的内存大小. 需要根据问题的性质选择最优的大小.
- **p**: minkowski距离度量里的参数. p=2时是欧氏距离, p=1时是曼哈顿距离.对于任意p, 就是minkowski距离
- **metric**: 用于距离度量, 默认的度量是minkowski距离 (也就是L<sub>p</sub>距离), 当p=2时就是欧氏距离.
- **metric\_params**: 距离公式的其他关键参数. 默认为None.
- **n\_jobs**: 并行处理的个数. 默认为1. 如果为-1, 那么那么CPU的所有cores都用于并行工作

同时, KNeighborsClassifier还有一些方法可以使用, 具体如下:

<code>fit(self, X, y)</code>	Fit the model using X as training data and y as target values
<code>get_params(self[, deep])</code>	Get parameters for this estimator.
<code>kneighbors(self, X, n_neighbors, ...)</code>	Finds the K-neighbors of a point.
<code>kneighbors_graph(self, X, n_neighbors, mode)</code>	Computes the (weighted) graph of k-Neighbors for points in X
<code>predict(self, X)</code>	Predict the class labels for the provided data
<code>predict_proba(self, X)</code>	Return probability estimates for the test data X.
<code>score(self, X, y[, sample_weight])</code>	Returns the mean accuracy on the given test data and labels.
<code>set_params(self, **params)</code>	Set the parameters of this estimator.

## 2 利用sklearn进行手写识别

其他的和上面的都差不多, 在handwritingClassTest函数中,不在用classify0函数, 而是用sklearn.neighbors.KNeighborsClassifier替代了classify0函数. 新建kNN\_Hsu06, 代码如下:

```
import numpy as np
import operator
from os import listdir
from sklearn.neighbors import KNeighborsClassifier as kNN

"""
函数说明:将32x32的二进制图像转换称1x1024向量

Parameters:
    filename - 文件名
Returns:
    returnVect - 返回的二进制图像的1x1024向量
"""
def img2vector(filename):
    #创建1x1024零向量
    returnVect = np.zeros((1,1024))
    #打开文件
    with open(filename) as fr:
        #按行读取, 注意readlines是读取所有行, readline是读取一行
        for i in range(32):
            #读取每一行数据
            lineStr = fr.readline()
            #每一行的前32个元素依次添加到returnVect中
            for j in range(32):
```

```

        #returnVect只有一行,也就是0,i和j都是从0开始,31结束,共32
        returnVect[0,32*i+j] = int(lineStr[j])

#返回转换后的1x1024向量
return returnVect

"""
函数说明:手写数字分类测试

Parameters:
    无
Returns:
    无
"""
def handwritingClassTest():
    #测试集的Labels
    hwLabels = []
    #返回trainingDigits目录下的文件名, listdir返回的是列表
    trainingFileList = listdir('trainingDigits')
    #返回文件夹下文件的个数
    m = len(trainingFileList)
    #初始化训练的Mat矩阵,测试集
    trainingMat = np.zeros((m,1024))
    #从文件名中解析出训练集类别
    for i in range(m):
        #获取文件的名称
        fileNameStr = trainingFileList[i]
        #获得分类的数字,因为文件名是'真实数字_文件序列号.txt',所以[0]就是真实数字
        classNumber = int(fileNameStr.split('_')[0])
        #将获取的真实数字(也就是类别)添加到hwLabel中
        hwLabels.append(classNumber)
        #将每个文件的1x1024数据存储在trainingMat矩阵中
        trainingMat[i,:] = img2vector('trainingDigits/%s' % (fileNameStr))

    #构建KNN分类器,基于sklearn构建
    neigh = KNN(n_neighbors=3, algorithm='auto')
    #训练矩阵,拟合模型,trainingMat为训练矩阵,hwLabels为对应的标签
    neigh.fit(trainingMat,hwLabels)
    #返回testDigits目录下的文件列表
    testFileList = listdir('testDigits')
    #错误检测计数
    errorCount = 0
    #测试集数据的数量
    mTest = len(testFileList)
    #从文件名中解析出测试集类别并进行分类测试
    for i in range(mTest):
        #获取文件的名称
        fileNameStr = testFileList[i]
        #获得分类的数字,因为文件名是'真实数字_文件序列号.txt',所以[0]就是真实数字
        classNumber = int(fileNameStr.split('_')[0])
        #获得测试集的1x1024向量,用于预测
        vectorUnderTest = img2vector('testDigits/%s' % (fileNameStr))
        #获得预测结果
        classifierResult = neigh.predict(vectorUnderTest)
        print("分类返回结果为%d\t真实结果为%d" % (classifierResult, classNumber))
        if classifierResult != classNumber:
            errorCount += 1
    print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount / mTest * 100))

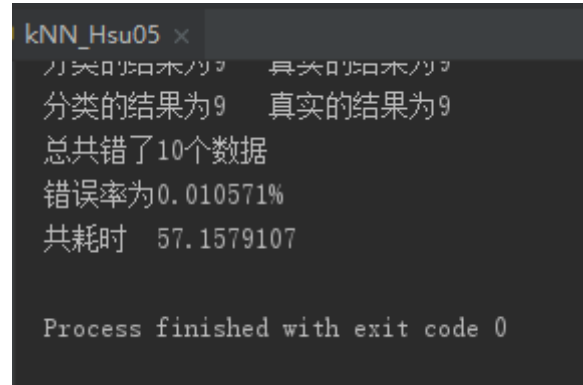
```

```
if __name__ == '__main__':  
    handwritingClassTest()
```

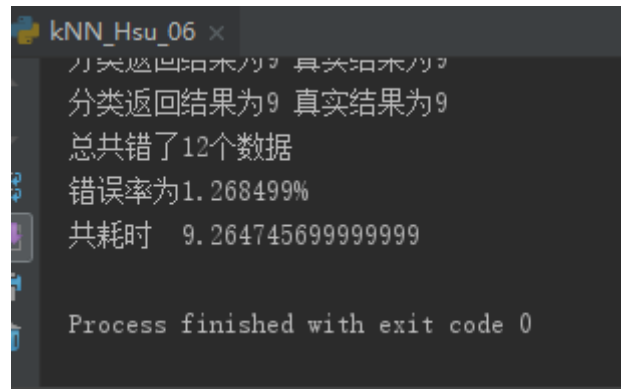
这段代码需要注意的有:

1. `neigh = kNN(n_neighbors=3, algorithm='auto')` 这里的k=3, algorithm参数是指: 快速k近邻搜索算法, 默认参数为auto, 可以理解为算法自己决定合适的搜索算法.

最后来对比下kNN\_Hsu05.py和kNN\_Hus06.py各自需要运行的时间.



```
kNN_Hsu05 x  
分类的结果为9 真实的结果为9  
总共错了10个数据  
错误率为0.010571%  
共耗时 57.1579107  
  
Process finished with exit code 0
```



```
kNN_Hsu_06 x  
分类返回结果为9 真实结果为9  
总共错了12个数据  
错误率为1.268499%  
共耗时 9.264745699999999  
  
Process finished with exit code 0
```

可以看到在错误率相差不大的情况下, 用sklearn模块的py程序运行时间只有9.26s, 而我没自己写的分类器运行了57.16s, 两者相差6倍以上, 相差还是很大的.