Python 数据分析与数据挖掘（Python for Data Analysis&Data Mining）

Chap 12 手写体数字识别

内容：

- kNN算法原理概述
- kNN算法的实现方式
- 手写体数字的预处理
- 基于kNN算法的手写体数字识别系统的实现
- 不同分类算法对于手写体数字识别系统的性能比较
- 算法：支持向量机（Support Vector Machine，SVM）， k-最近邻（k-Nearest Neighbor，kNN），决策树（Decision Tree, DT），朴素贝叶斯（Naive Bayes，NB）算法和不同算法性能比较
- 应用领域：手写体数字、字母等识别，目标识别等

实践：

- kNN算法的实现方式
- 手写体数字的预处理
- 基于不同分类算法的手写体数字识别系统性能的比较

实例：

- 实例1：kNN算法的实现方式
- 实例2：基于kNN算法的手写体数字识别分类系统
- 实例3：比较几种分类算法对于手写体数字识别系统的性能

作业4：手写体数字识别系统

这节课是在前面数据分析的基础上，对手写体数字图像进行预处理和构建识别系统。本节课通过手写体数字图像实例来进行数字图像的实践分析和识别分类，也适用于其他字符的多种图像类型数据。此外，本节课比较了基于多种不同分类算法的识别系统的性能。

注意，本节课中未涉及更多图像处理操作，具体细节如果有兴趣，可以选读图像处理相关课程。

准备工作：导入库，配置环境等

In [ ]:

```
from __future__ import division
import os, sys

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

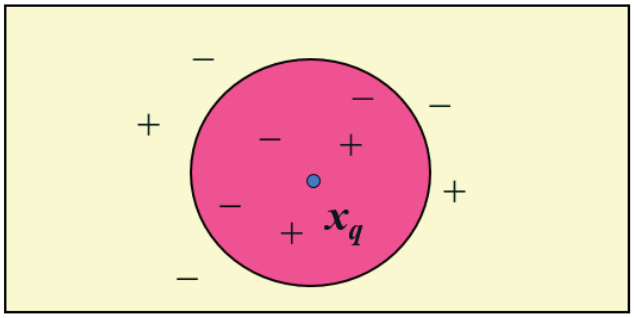
实例1：kNN算法(k-Nearest Neighbor)的实现方式

1. kNN算法原理概述

模型表达形式

模型可以是树，概率，超平面或者带权重的多层网络，或者是一个函数方程等。

kNN（k-Nearest Neighbor）：k最近邻算法



K=5

懒惰学习 vs. 急切学习

学习法	内容	时间开销	算法
懒惰学习法（Lazy learning）	只存储训练数据或仅仅只进行小的处理，一直等到新的测试数据到来才开始进行学习	没有训练学习时间，预测时间较多	kNN，案例推理（Case-based reasoning）
急切学习（Eager learning）	给定训练数据，在对新的数据进行预测之前，已经建立好分类模型	训练学习时间多，预测时间较少	决策树，SVM

kNN算法

- 所有的样本都对应是一个n维空间中的点，即，n维的向量，例如 $A = [a_1, a_2, \ldots, a_n]$
- 最近邻居定义为欧式距离，给定两个样本点， $X_1 = [x_{11}, x_{12}, \ldots, x_{1n}]$，和 $X_2 = [x_{21}, x_{22}, \ldots, x_{2n}]$，则这两个样本点的距离为：
  - $dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$
- 目标函数可以是离散的，或者连续值
  - 对于离散值，kNN算法返回与预测样本 x_q 最相似（即距离最近的）k个训练样本的类标
  - 对于连续值，kNN算法返回与预测样本 x_q 最相似（即距离最近的）k个训练样本的均值
- 距离加权(Distance-weighted)kNN算法，可以根据最近的k个邻居的距离进行加权，$w = \frac{1}{d(x_q, x_i)^2}$
- kNN算法具有噪音鲁棒性（robust），即对噪音不敏感，因为只受最近的k个邻居影响
- 维度灾难(Curse of dimensionality)：邻居之间的距离可能会受到不相关特征的影响，怎么消除？对不相关特征属性进行缩减或删除

案例推理（Case-based reasoning）

- 使用问题解决数据库去解决新的问题
- 存储符号描述（案例或元组），而不是欧式空间中的点

- 应用：客户服务，产品故障解决，病情诊断，法律法令案件等

2. kNN算法的三种实现方式

- 自己实现kNN算法
- 借用现成的第三方库

1. kNN算法的第一种实现方式

- 创建数据
- 加载数据
- 自己实现kNN
- 预测新样本

In [ ]:

```
from  numpy import *  # 导入科学计算包NumPy
import operator  # 导入运算符模块

# 创建数据集和标签
def createTrainDataSet():
    traindataset = array([[1.0,1.1], [1.0,1.0],[0,0],[0,0.1]])
    labels = ['A','A','B','B']
    return traindataset,labels
```

In [ ]:

```
trdataset,labels = createTrainDataSet()
print trdataset
print labels
```

In [ ]:

```
print trdataset.shape[0]  # 返回数据集的行数，即样本个数
print trdataset.shape[1]  # 返回数据集的列数，即样本的维度
```

In [ ]:

```
# 第一个kNN算法的实现
# 计算一个输入测试数据inX与已知样本的距离，并返回类别标签
def kNN(newInput, dataSet, labels, k): ## inX是待分类测试样本，dataSet是训练样本，labels是训练样本标签
    dataSetSize = dataSet.shape[0] ## 行数，即样本个数

    ## step 1: calculate Euclidean distance
    # tile(A, reps): Construct an array by repeating A reps times
    # the following copy numSamples rows for dataSet
    diff = tile(newInput, (dataSetSize, 1)) - dataSet # Subtract element-wise
    squaredDiff = diff ** 2 # squared for the subtract
    squaredDist = sum(squaredDiff, axis = 1) # sum is performed by row
    distances = squaredDist ** 0.5 # squared for the subtract

    ## step 2: sort the distance
    # argsort() returns the indices that would sort an array in a ascending order
    ## 按距离排序的索引
    sortedDistIndicies = distances.argsort()

    classCount = {}# define a dictionary (can be append element)
    for i in range(k):
        ## step 3: choose the min k distance  ## 选择距离最小的k个点
        voteIlabel = labels[sortedDistIndicies[i]]
        ## step 4: count the times labels occur
        # when the key voteLabel is not in dictionary classCount, get()
        # will return 0
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1

    ## step 5: the max voted class will return
    sortedClassCount = sorted(classCount.iteritems(),key=operator.itemgetter(1),reverse=True) ## 排序
    return sortedClassCount[0][0]
```

In [ ]:

```
# 新测试样本为[0.6,0.3]
kNN([0.6,0.3],trdataset, labels, 3)
```

In [ ]:

```
# 新测试样本为[1.2,0.8]
kNN([1.2,0.8],trdataset, labels, 3)
```

2. kNN算法的第二种实现方式

- 将上述代码保存到knn.py文件中
- 改变当前路径到存储knn.py文件的位置 cd L12/knn
- 打开python开发环境 python
- 导入knn的程序模块 import knn 或者reload(knn)
- 创建两个变量group和labels，group,labels=knn.createTrainDataSet()
- 检查两个变量的值是否正确，group, labels
- 测试样本为[0.6,0.3] knn.kNN([0.6,0.3],group, labels, 3) 结果为B类
- 测试样本为[1.2,0.8] knn.kNN([1.2,0.8],group, labels, 3) 结果为A类

3. kNN算法的第三种实现方式

- 调用Sklearn库实现的k最近邻算法 (http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.NearestNeighbors.html#sklearn.neighbors.NearestI

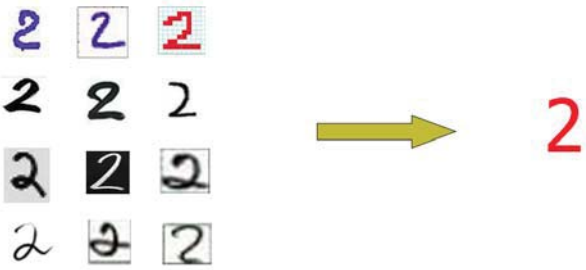
```
class sklearn.neighbors.NearestNeighbors(n_neighbors=5, radius=1.0, algorithm='auto',
leaf_size=30, metric='minkowski', p=2, metric_params=None, n_jobs=1, **kwargs)[source]
```

```
from sklearn.neighbors import KNeighborsClassifier  # 导入k最近邻算法
k = 3 # 设定最近邻居个数K
kNN = KNeighborsClassifier(n_neighbors=k) # 构造k=3最近邻模型
kNN.fit(x_train, y_train)  # 使用训练数据和训练数据对应的类标来训练模型
```

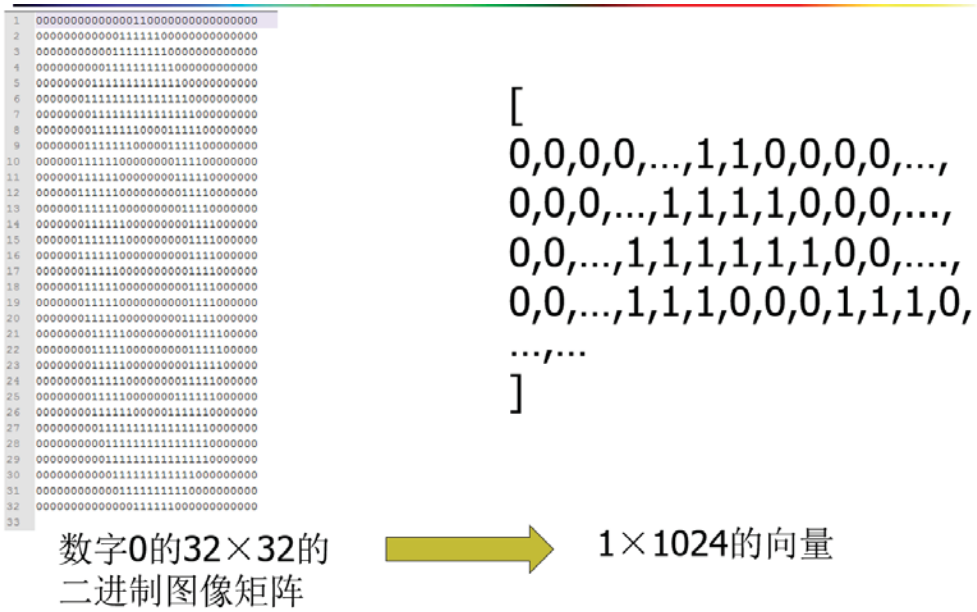
可以比较不同的k参数，即n_neighbors（默认为5）

实例2：基于kNN的手写体数字识别分类系统

1. 问题描述



每个数字是 $32 * 32$ 的二进制图像矩阵，存放在data目录下面



2. 图像转为向量, img2vector(filename)

- testVector = kNN.img2vector('data/0_5.txt')
- testVector[0, 0:31] //显示图像的第一行，与文本编辑器打开的文件进行比较
- testVector[0, 32:63] //显示图像的第二行

In [ ]:

```
## 首先将数据处理成分类器可以识别的文本格式，将32×32的二进制图像矩阵转换为1×1024的向量
def img2vect(filename):
    returnVect = zeros((1, 1024))    ##首先创建1×1024的Numpy数组
    fr = open(filename)
    for i in range(32):    ## 循环读出文件的前32行，并将每行的前32个字符值存储在Numpy数组中
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i + j] = int(lineStr[j])
    return returnVect ## 最后返回数组
```

In [ ]:

```
testVect = img2vect('data/0_5.txt')
print testVect[0,0:31] # 显示图像的第一行
print testVect[0, 32:63] # 显示图像的第二行
```

3. 获取训练和测试目录下的内容

- from os import listdir
- trainingFileList = listdir('trainingDigits/')## 获取训练目录下的内容
- print len(trainingFileList) # 训练数据样本个数
- testFileList = listdir('testDigits/')## 获取训练目录下的内容
- print len(testFileList) # 测试数据样本个数

In [ ]:

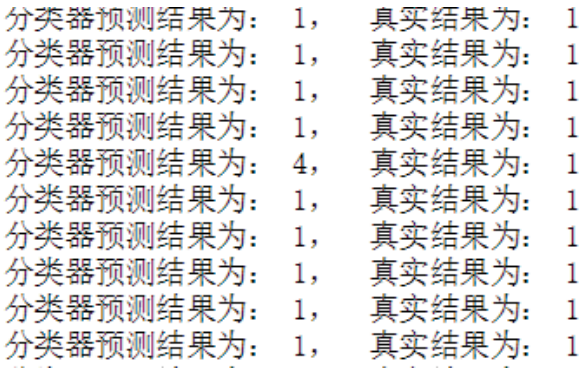
```
# 获取训练和测试目录中的内容
from os import listdir

trainingFileList = listdir('data/trainingDigits/')## 获取训练目录下的内容
print "训练样本个数：%d" % (len(trainingFileList))  # 训练数据样本个数

testFileList = listdir('data/testDigits/')## 获取训练目录下的内容
print "测试样本个数： %d" % (len(testFileList))  # 测试数据样本个数
```

4. 手写数字识别系统的测试代码, handwritingClassTest()

- handwritingClassTest() //测试系统的输出结果



In [ ]:

```
## 手写数字识别系统的测试代码
def handwritingClassTest():
    hwLabels = []
    trainingFileList = listdir('data/trainingDigits/') ## 获取训练目录的内容
    m = len(trainingFileList) ## 获取训练样本个数
    trainingMat = zeros((m, 1024)) ## 矩阵每行存储一个图像
    for i in range(m):
        fileNameStr = trainingFileList[i]  ##从训练数据文件名解析分类数字
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vect('data/trainingDigits/%s' % fileNameStr)

    ## 解析测试数据文件
    testFileList = listdir('data/testDigits')
    errorCount = 0.0  ## 统计识别错误的文件个数
    mTest = len(testFileList)  ## 测试样本个数
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        vectorUnderTest = img2vect('data/testDigits/%s' % fileNameStr) ## 二进制图像转成向量
        classifierResult = kNN(vectorUnderTest, trainingMat, hwLabels, 3) ##进行kNN分类
        print "分类器预测结果为： %d,  真实结果为： %d" % (classifierResult, classNumStr)
        if (classifierResult != classNumStr): errorCount += 1.0
    print "\n 测试样本个数为： %d " % mTest
    print " 预测错误个数为： %d " % errorCount
    print " 预测错误率为： %2.2f%%" % (errorCount/float(mTest)*100.0)
    print " 预测准确率为： %2.2f%%" % ((1-errorCount/float(mTest))*100.0)
```

In [ ]:

```
handwritingClassTest()
```

kNN小节

- kNN是最简单有效的分类数据的算法。
- 实际kNN算法的执行效率不高，每个测试向量（946个测试样本）做近2000次距离计算（1934个训练样本），每个距离计算包括了1024（32*32）个维度的浮点运算，总计执行近900次（946个测试样本）。
- 此外，还要为测试向量准备2MB的存储空间。
- kNN的另一个缺陷是，无法给出任何数据的基础结构信息，无法知晓平均样本和典型样本具有什么特征。
- 为了改进（减少存储空间和计算时间开销），k决策树是kNN算法的优化版，可以节省大量的计算开销。

实例3：比较几种分类算法对于手写体数字识别系统的性能

kNN算法也使用第三方库函数，方便，快捷，非常灵活。

In [ ]:

```
#导入库
import numpy as np
from  numpy import *  # 导入科学计算包NumPy
from os import listdir  # 从os中导入函数，列出给定目录的文件名
import operator# 导入运算符模块
```

In [ ]:

```
## 将数据处理成分类器可以识别的格式，将32×32的二进制图像矩阵转换为1×1024的向量
def img2vect(filename):
    returnVect = zeros((1, 1024))    ##首先创建1×1024的Numpy数组
    fr = open(filename)
    for i in range(32):    ## 循环读出文件的前32行，并将每行的前32个字符值存储在Numpy数组中
        lineStr = fr.readline()
        for j in range(32):
            returnVect[0,32*i + j] = int(lineStr[j])
    return returnVect ## 最后返回数组
```

In [ ]:

```
# 定义加载训练数据
def load_trainingData():
    hwLabels = []
    trainingFileList = listdir('./data/trainingDigits') ## 获取训练目录的内容
    m = len(trainingFileList) ## 获取训练样本个数
    trainingMat = zeros((m, 1024)) ## 矩阵每行存储一个train图像
    for i in range(m):
        fileNameStr = trainingFileList[i]  ##从训练数据文件名解析分类数字
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        hwLabels.append(classNumStr)
        trainingMat[i,:] = img2vect('./data/trainingDigits/%s' % fileNameStr)
    return trainingMat, hwLabels
```

In [ ]:

```
# 加载数据
trainingMat, hwLabels = load_trainingData()
len(trainingMat),len(hwLabels)
```

In [ ]:

```
# 定义加载测试数据
def load_testData():
    testFileList = listdir('./data/testDigits')
    goldLabels = []  ## 统计文件个数
    mTest = len(testFileList)  ## 测试样本个数
    testMat = zeros((mTest, 1024)) ## 矩阵每行存储一个train图像
    for i in range(mTest):
        fileNameStr = testFileList[i]
        fileStr = fileNameStr.split('.')[0]
        classNumStr = int(fileStr.split('_')[0])
        goldLabels.append(classNumStr)
        testMat[i,:]= img2vect('./data/testDigits/%s' % fileNameStr) ## 二进制图像转成向量
    return testMat, goldLabels
```

In [ ]:

```
testMat,goldLabels = load_testData()
len(testMat),len(goldLabels)
```

In [ ]:

```
# 导入不同分类算法的库

from sklearn.neighbors import KNeighborsClassifier # Sklearn中kNN算法
from sklearn.svm import SVC # SKlearn中SVM算法
from sklearn.tree import DecisionTreeClassifier # SKlearn中决策树算法

# Sklearn中NB算法
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
```

In [ ]:

```
## 基于几种不同分类算法的手写数字识别系统的代码
def handwritingClassTest():
    trainingMat, hwLabels = load_trainingData()
    testMat,goldLabels = load_testData()
    mTest = len(testMat) ## 测试样本个数

    ## 调用sklearn库中的分类算法
    ensemble =["kNN", "SVC", "DT", "GaussianNB", "MultinomialNB", "BernoulliNB"]
    for a in ensemble:
        classifierResult = []
        print a + ":"
        if a == "kNN": clf = KNeighborsClassifier(algorithm='kd_tree', n_neighbors = 3)
        if a == "SVC": clf = SVC(C=1.0,kernel='linear')
        if a == "DT": clf = DecisionTreeClassifier(criterion='entropy', random_state=0)
        if a == "GaussianNB" : clf = GaussianNB()
        if a == "MultinomialNB" : clf = MultinomialNB()
        if a == "BernoulliNB" : clf = BernoulliNB()

        clf.fit(trainingMat,hwLabels)  # 训练模型
        classifierResult = clf.predict(testMat)  # 应用模型，预测测试数据

        errorCount = 0.0  ## 统计识别错误的样本个数
        for i in range(mTest):
            if classifierResult[i] != goldLabels[i]:
                errorCount += 1.0

        print "\t 测试样本个数为： %d " % mTest
        print "\t 预测错误个数为： %d " % errorCount
        print "\t 预测错误率为： %2.2f%% " % (errorCount/float(mTest)*100)
        print "\t 预测准确率为： %2.2f%%" % ((1-errorCount/float(mTest))*100)
```

In [ ]:

```
handwritingClassTest()
```

小节

改变各个算法的参数，观察结果的变化情况

作业4：手写体数字识别系统

- 手写体数字识别系统：运用并比较多种不同算法提高手写体数字识别系统的准确性
- 使用目前提供的数据做训练数据构建识别系统
- 应用多种不同策略提高识别系统性能，包括：
  - 不同的算法
  - 算法的不同参数
  - 多种算法的综合（Ensemble策略）
- 系统构建后，在后续提供的测试数据上进行预测，提交结果的格式情况如下：
  - 预测结果文件的命名为，学号-姓名-作业4.txt
  - 预测结果文件的格式为： 每行对应一个测试文件，每行格式为：测试文件的序号\t预测类标