实验 KNN

姓名：李昀哲　学号：20123101

1 题目和数据：

题目：KNN

基于 kNN 算法改进约会网站的配对效果
基于 kNN 算法实现手写体数字识别

数据描述:

- **Helen 女士约会网站数据**

海伦收集约会数据已经有了一段时间，她把这些数据存放在文本文件 datingTestSet.txt 中，每个样本数据占据一行，总共有 1000 行。

海伦收集的样本数据主要包含以下 3 种特征:

- 每年获得的飞行常客里程数
- 玩视频游戏所消耗时间百分比
- 每周消费的冰淇淋公升数

```
40920   8.326976    0.953952    largeDoses
14488   7.153469    1.673904    smallDoses
26052   1.441871    0.805124    didntLike
75136   13.147394   0.428964    didntLike
38344   1.669788    0.134296    didntLike
72993   10.141740   1.032955    didntLike
35948   6.830792    1.213192    largeDoses
42666   13.276369   0.543880    largeDoses
67497   8.631577    0.749278    didntLike
35483   12.273169   1.508053    largeDoses
```

- **手写数字识别数据**

数据集目录下有两个子目录：trainingDigits 和 testDigits

- trainingDigits 为训练数据集的文件，文件数量为 1935;
- testDigits 为测试数据集的文件，文件数量为 947。

每个文件的命名格式都为“真实数字_编号.txt”。为了简便处理，实验中，用 txt 文本文件表示图片。原图片中像素值为黑色（0,0,0）的像素点在 txt 中对应的用 0 表示，像素值为白色（255,255,255）的像素点用 1 表示。所以，只需要处理这些文本文件即可，不用再去解析图片格式。每个 txt 文件中，数据共有 32 行和 32 列，这是由于原图片的大小为 32X32。

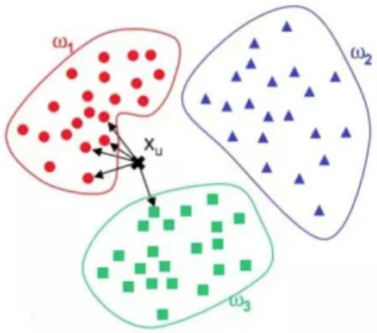
2 算法：

KNN（K-Nearest Neighbor）算法是机器学习算法中最基础、最简单的算法之一。它既能用于分类，也能用于回归。KNN 通过测量不同特征值之间的距离来进行分类。

KNN 算法的思想非常简单：对于任意 n 维输入向量，分别对应于特征空间中的一个点，输出为该特征向量所对应的类别标签或预测值。

KNN 算法是一种非常特别的机器学习算法，因为它没有一般意义上的学习过程。它的工作原理是利用训练数据对特征向量空间进行划分，并将划分结果作为最终算法模型。存在一个样本数据集合，也称作训练样本集，并且样本集中的每个数据都存在标签，即我们知道样本集中每一数据与所属分类的对应关系。

输入没有标签的数据后，将这个没有标签的数据的每个特征与样本集中的数据对应的特征进行比较，然后提取样本中特征最相近的数据（最近邻）的分类标签。



由于 KNN 是基于距离度量的算法，因此距离计算公式常用欧几里得距离、曼哈顿距离和闵可夫斯基距离，三种距离公式如下所示

$$\text{Euclidean Distance} = \mathrm{d}(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

$$\text{Manhattan Distance} = \mathrm{d}(x,y) = (\sum_{i=1}^{m} | x_i - y_i |)$$

$$\text{Minkowski Distance} = \mathrm{d}(x,y) = (\sum_{i=1}^{m} | x_i - y_i |)^{1/p}$$

KNN 的优势在于：

- 易于实现：鉴于算法的简单性和准确性，它是新数据科学家将学习的首批分类器之一。
- 轻松适应：随着新训练样本的增加，算法会根据任何新数据进行调整，因为所有训练数据都存储在内存中。
- 很少的超参数：KNN 只需要 a k 值和距离度量，与其他机器学习算法相比，所需的超参数很少；

KNN 的劣势在于：

- 不能很好地扩展：由于 KNN 是一种惰性算法，因此与其他分类器相比，它占用了更多的内存和数据存储。 从时间和金钱的角度来看，这可能是昂贵的。 更多的内存和存储将增加业务开支，而更多的数据可能需要更长的时间来计算。 虽然已经创建了不同的数据结构（例如 Ball-Tree）来解决计算效率低下的问题，但分类器

是否理想可能取决于业务问题；

- 容易过拟合：由于"维度的诅咒"，KNN 也更容易过拟合。 虽然利用特征选择和降维技术来防止这种情况发生，但 k 的值也会影响模型的行为。 较小的 k 值可能会过度拟合数据，而较大的 k 值往往会"平滑"预测值，因为它是对更大区域或邻域的值进行平均。 但是，如果 k 的值太高，那么可能会欠拟合数据。

3 代码及结果

数据集读入

通过特征中的类别，分别用 1，2，3 表示；

```
def file2matrix(filename):
    """
    函数说明：加载数据集
    parameters:
        fileName - 文件名
    return:
        featureMat - 特征矩阵
        classLabelVector - 类别标签向量(didntLike - 0, smallDoses - 1, largeDoses - 2)
    """
    # 打开文件
    fr = open(filename)
    # 读取文件所有内容
    arrayOLines = fr.readlines()
    # 得到文件行数
    numberOfLines = len(arrayOLines)
    # 返回的NumPy矩阵,解析完成的数据:numberOfLines行,3列
    returnMat = np.zeros((numberOfLines,3))
#     print(returnMat)
    # 返回的分类标签向量
    classLabelVector = []
    # 行的索引值
    index = 0
    for line in arrayOLines:
        # s.strip(rm)， 当rm空时,默认删除空白符(包括'\n','\r','\t',' ')
        line = line.strip()
        # 使用s.split(str="",num=string,cout(str))将字符串根据'\t'分隔符进行切片。
        listFormLine = line.split('\t')
        # 将数据前三列提取出来,存放到returnMat的NumPy矩阵中,也就是特征矩阵
        returnMat[index,:] = listFormLine[0:3]
        # 根据文本中标记的喜欢的程度进行分类,1代表不喜欢,2代表魅力一般,3代表极具魅力
        if listFormLine[-1] == 'didntLike':
            classLabelVector.append(1)
        elif listFormLine[-1] == 'smallDoses':
            classLabelVector.append(2)
        elif listFormLine[-1] == 'largeDoses':
            classLabelVector.append(3)
        index += 1
    return returnMat,classLabelVector
```

数据归一化

在处理不同取值范围的特征值时，我们通常采用的方法是将数值归一化，如将取值范围处理为 0 到 1 或者- 1 到 1 之间。本实验使用如下的公式可以将任意取值范围的特征值转化为 0 到 1 区间内的值;

```
def autoNorm(dataSet):
    """
    函数说明：数据归一化
    parameters:
        dataSet - 特征矩阵
    return:
        normDataset - 归一化特征矩阵
        ranges - 数据范围(Xmax - Xmin)
        minVals - 最小特征值
    """
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = np.zeros(np.shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - np.tile(minVals,(m,1))
    normDataSet = normDataSet / np.tile(ranges,(m,1))
    return normDataSet,ranges,minVals
```

KNN 划分算法

算法的思想是简单的，但其中涉及较多 python 字典的处理，较为复杂;

```
def kNNClassify(inX, dataSet, labels, k):
    """
    函数说明：kNN分类
    parameters:
        inX - 用于要进行分类判别的数据(来自测试集)
        dataSet - 用于训练的数据(训练集)
        labels - 分类标签
        k - kNN算法参数，选择距离最小的k个点
    return:
        predLabel - 预测类别
    """
    # numpy函数shape[0]返回dataSet的行数
    dataSetSize = dataSet.shape[0]
    # 在列向量方向上重复inX共1次(横向),行向量方向上重复inX共dataSetSize次(纵向)
    diffMat = np.tile(inX,(dataSetSize,1)) - dataSet
    # 二维特征相减后平方
    sqDiffMat = diffMat**2
    # sum()所有元素相加,sum(0)列相加,sum(1)行相加
    sqDistance = sqDiffMat.sum(axis = 1)
    # 开方,计算出距离
    distances = sqDistance ** 0.5
    # 返回distances中元素从小到大排序后的索引值
    sortedDistIndices = distances.argsort()
    # 定一个记录类别次数的字典
    classCount = {}
    for i in range(k):
        # 取出前k个元素的类别
        voteIlabel =  labels[sortedDistIndices[i]]
        # dict.get(key,default=None),字典的get()方法,返回指定键的值,如果值不在字典中返回默认值。
        # 计算类别次数
        classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
        # python3中用items()替换python2中的iteritems()
        # key=operator.itemgetter(1)根据字典的值进行排序
        # key=operator.itemgetter(0)根据字典的键进行排序
        # reverse降序排序字典
    sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1),reverse=True)
    # 返回次数最多的类别,即所要分类的类别
    return sortedClassCount[0][0]
```

分类测试

```
def datingClassTest():
    """
    函数说明：测试kNN分类器

    """
    filename = "./kNN_Dating/datingTestSet.txt"
    datingDataMat, datingLabels = file2matrix(filename)

    hoRatio = 0.10
    normMat, ranges, minVals = autoNorm(datingDataMat)
    m = normMat.shape[0
    numTestVecs = int(m * hoRatio)


    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = kNNClassify(normMat[i,:],normMat[numTestVecs:m,:],datingLabels[numTestVecs:m],4)
        print("分类结果:%d\t真实类别:%d" % (classifierResult, datingLabels[i]))
        if classifierResult != datingLabels[i]:
            errorCount += 1.0
        print("错误率:%f%%" % (errorCount / float(numTestVecs) * 100))
```

分类结果

测试集数据量为 100，分类准确率为 96%

```
错误率:3.000000%
分类结果:1      真实类别:1
错误率:3.000000%
分类结果:2      真实类别:2
错误率:3.000000%
分类结果:1      真实类别:1
错误率:3.000000%
分类结果:3      真实类别:3
错误率:3.000000%
分类结果:3      真实类别:3
错误率:3.000000%
分类结果:2      真实类别:2
错误率:3.000000%
分类结果:2      真实类别:1
错误率:4.000000%
分类结果:1      真实类别:1
错误率:4.000000%
```

手写数字数据加载

```
def img2vector(fileName):
    """
    函数说明：将手写体数字文本转化为一维数组
    parameters:
        fileName - 手写体数字文本文件路径
    return:
        returnVect - 1x1024向量
    """

    #创建1x1024零向量
    returnVect = np.zeros((1, 1024))
    fr = open(fileName)
    for i in range(32):
        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[0, 32*i+j] = int(lineStr[j])
    #返回转换后的1x1024向量
    return returnVect
```

```
def loadTrainData(path):
    """
    函数说明：加载目录下所有手写体数字文本文件
    parameters:
        path - 手写体数字文本文件路径
    return:
        hwLabels - 训练集标签
        hwMat - 训练集数据
    """
    #定义测试集的Labels
    hwLabels = []
    trainingFileList = listdir('kNN_hand_writing/trainingDigits')

    #返回文件夹下文件的个数
    m = len(trainingFileList)
    trainingMat = np.zeros((m, 1024))

    #从文件名中解析出训练集的类别
    for i in range(m):
        fileNameStr = trainingFileList[i]
        classNumber = int(fileNameStr.split('_')[0])
        #将获得的类别添加到hwLabels中
        hwLabels.append(classNumber)
        #将每一个文件的1x1024数据存储到trainingMat矩阵中
        trainingMat[i,:] = img2vector('kNN_hand_writing/trainingDigits/' + (fileNameStr))

    neigh =KNN(n_neighbors = 3, algorithm = 'auto')
    #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('kNN_hand_writing/testDigits')
```

分类测试

```
def handwritingClassTest():
    """
    函数说明：测试kNN手写体识别算法

    """
    #定义测试集的Labels
    hwLabels = []
    trainingFileList = listdir('kNN_hand_writing/trainingDigits')
    m = len(trainingFileList)

    #初始化训练的Mat矩阵，训练集
    trainingMat = np.zeros((m, 1024))
    #从文件名中解析出训练集的类别
    for i in range(m):
        fileNameStr = trainingFileList[i]
        classNumber = int(fileNameStr.split('_')[0])
        hwLabels.append(classNumber)
        #将每一个文件的1x1024数据存储到trainingMat矩阵中
        filename = 'kNN_hand_writing/trainingDigits/' + fileNameStr
        trainingMat[i,:] = img2vector(filename)

    from sklearn.neighbors import KNeighborsClassifier as kNN
    neigh = kNN(n_neighbors = 3, algorithm = 'auto')
    #拟合模型, trainingMat为测试矩阵,hwLabels为对应的标签
    neigh.fit(trainingMat, hwLabels)
    testFileList = listdir('kNN_hand_writing/testDigits')

    #错误检测计数
    errorCount = 0.0
    mTest = len(testFileList)
    for i in range(mTest):
        fileNameStr = testFileList[i]
        classNumber = int(fileNameStr.split('_')[0])
        #获得测试集的1x1024向量,用于训练
        vectorUnderTest = img2vector('kNN_hand_writing/testDigits/' + fileNameStr)
        classifierResult = neigh.predict(vectorUnderTest)
        print(fileNameStr+"分类返回结果为%d\t真实结果为%d" % (classifierResult, classNumber))
        if(classifierResult != classNumber):
            errorCount += 1.0
    print("总共错了%d个数据\n错误率为%f%%" % (errorCount, errorCount/mTest))
```

测试结果

共测试 946 组数据，准确率为 99.987%

```
3_9.txt分类返回结果为3  真实结果为3
7_66.txt分类返回结果为7 真实结果为7
4_65.txt分类返回结果为4 真实结果为4
8_90.txt分类返回结果为8 真实结果为8
2_6.txt分类返回结果为2  真实结果为2
9_8.txt分类返回结果为9  真实结果为9
0_70.txt分类返回结果为0 真实结果为0
4_20.txt分类返回结果为4 真实结果为4
5_15.txt分类返回结果为5 真实结果为5
3_16.txt分类返回结果为3 真实结果为3
共测试946组数据,12个数据分类错误
错误率为0.012685%
```