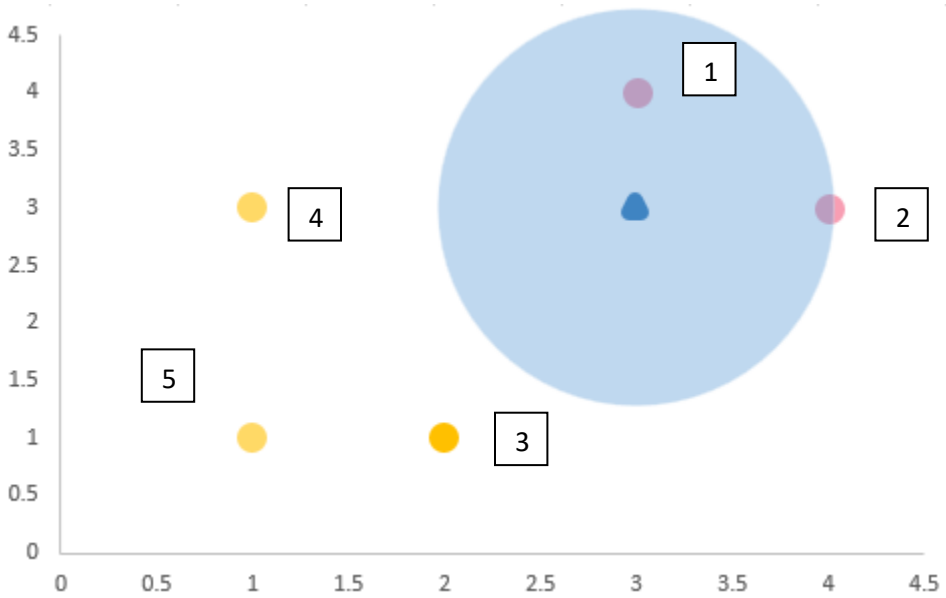
K 近邻算法

一、直观理解

特征空间内有五个点，橙色代表 A 类，粉色代表 B 类，我们往特征空间投入一个实例 X（3，3）。此时我们计算五个点到 X 的欧式几何距离，并按远近顺序排列（如数字所示）。



1. 欧式几何距离。计算公式：$L_P(x_i,x_j)=\left(\sum_{l=1}^{n}\left|x_i^{(l)}-x_j^{(l)}\right|^2\right)^{\frac{1}{2}}$，其中 $x_i=\left(x_i^{(1)},x_i^{(2)},\dots,x_i^{(n)}\right)^T$，$x_j=\left(x_j^{(1)},x_j^{(2)},\dots,x_j^{(n)}\right)^T$。比如（0，0）到（1，1）的欧式距离即是我们熟知的点到点的距离$\sqrt{2}$。

2. k 值的选择问题。我们令 k=2, 即选择距离 X 最近的两个点。我们发现两个点的分类都是 B 类，于是判断 X 为 B 类（多数表决）。如果我们令 k=3，距离 X 最近的三个点中，两个是 B 类，一个是 A 类，于是判断 X 为 B 类。当令 k=5 时，距离 X 最近的五个点中，三个是 A 类，两个是 B 类，于是判断 X 为 A 类。可见，k 临近算法十分简单，模型好坏与 k 值的选择有相关。

1）如果选择较小的 k 值。相似误差减小，估计误差增大。预测结果会对近邻的实例点非常敏感，如果邻近的实例点恰巧是噪声，预测就会出错。此时整体模型变得复杂，容易过拟合。

2）如果选择较大的 k 值。相似误差增大，估计误差减小，这时与输入实例较远的（不相似）训练实例也会对预测起作用，使预测发生错误。K 值增大意味着模型变得简单。

3）在应用中，k 值一般取一个比较小的数值。通常采用交叉验证法来选取最优的 k 值。

* 理论部分参考《统计学习方法》，代码部分引用《机器学习实战》

二、Python 代码实现

1. 生成简单的特征空间

```
from numpy import *
def createDataSet():
    group = array([[1,1],[2,1],[1,3],[3,4],[4,3]])
    labels = ['A','A','A','B','B']
    return group,labels
```

2. 实现 k 近邻算法

```
from numpy import *
import operator

def classify(inX,dataSet,labels,k):
    # 计算距离
    dataSetSize = dataSet.shape[0]
    diffMat = tile(inX,(dataSetSize,1))-dataSet
    sqDiffMat = diffMat ** 2
    sqDistances = sqDiffMat.sum(axis=1)
    distances = sqDistances ** 0.5

    # 按离该点的距离按从近到远的距离排序
    sortedDistIndicies = distances.argsort()

    # 计算前 k 个点的分类，并返回最多点的分类
    classCount = {}
    for i in range(k):
        voteIlabel = labels[sortedDistIndicies[i]]
        classCount[voteIlabel] = classCount.get(voteIlabel, 0) + 1
    sortedClassCount = sorted(classCount.items(),
                              key=operator.itemgetter(1), reverse=True)
    return sortedClassCount[0][0]
```

一些说明：

1）numpy.ndarray.shape　返回数组的维度

```
from numpy import *
group = array([[1,1],[2,1],[1,3],[3,4],[4,3]])
print(group.shape)
```

返回的结果为（5，2），这里 shape[0] = 5

2）numpy.tile　快速重复数组

```
from numpy import *
print(tile([3,3],(5,1)))
```

结果为[[3 3] [3 3] [3 3] [3 3] [3 3]]，即将[3,3]重复 5 行

3）numpy.sum(axis=1) 中 axis=1 表示对行求和

4）numpy.argsort 排序并返回索引数组

```
from numpy import *
group = array([3,1,0,2])
print(group.argsort())
```

结果为[2 1 3 0]，group[2]< group[1]< group[3]< group[0]

5) sorted 排序 operator.itemgetter(item) 对 key 进行排序，reverse=True 表示降序

```
from numpy import *
import operator
group = {'A':3,'B':2}
sorted(group.items(),key=operator.itemgetter(1), reverse=True)
```

这里，我们可以使用简单的样本空间进行分类了。如最开始的图所示，我们提供一个新样本[3,3]进行测试。

测试程序 1：

```
group,labels = createDataSet()
print(classify0([3,3],group,labels,3))
```

结果 1：

```
B
```

测试程序 2：

```
group,labels = createDataSet()
print(classify0([3,3],group,labels, 5))
```

结果 2：

```
A
```

接下来，我们考虑更加复杂的情况。

3. 读取数据（这里的示例为 datingTestSet.txt 文件）

```
def file2matrix(filename):
    fr = open(filename)
    numberOfLines = len(fr.readlines())
    returnMat = zeros((numberOfLines,3))
    classLabelVector = []
    fr = open(filename)
    index = 0
    for line in fr.readlines():
        love_dictionary = {'largeDoses': 3, 'smallDoses': 2, 'didntLike':
1}
        line = line.strip()
        listFromLine = line.split('\t')
        returnMat[index,:] = listFromLine[0:3]
        if (listFromLine[-1].isdigit()):
            classLabelVector.append(int(listFromLine[-1]))
        else:
            classLabelVector.append(love_dictionary.get(listFromLine[-1]))
        index += 1
    return returnMat,classLabelVector
```

一些说明：

1）str.strip 用于移除字符串头尾指定的字符(默认为空格)

2）这个函数返回了两个数组，一个是样本数据，另一个是标签

4. 归一化

有时候我们发现一个对象可能有几个不同的维度，维度的大小相差甚远，比如[10000,8,0]形成的空间坐标，致使第一个维度对点的空间位置的影响极大，因此必须要归一化，让每一个维度在[0,1]之间。这里使用的归一化公式为：

$$\text{newValue} = \frac{(oldValue - min)}{(max - min)}$$

此时[10000,8,0]转化为[1,0.0008,0]

```
def autoNorm(dataSet):
    minVals = dataSet.min(0)
    maxVals = dataSet.max(0)
    ranges = maxVals - minVals
    normDataSet = zeros(shape(dataSet))
    m = dataSet.shape[0]
    normDataSet = dataSet - tile(minVals,(m,1))
    normDataSet = normDataSet/tile(ranges,(m,1))
    return normDataSet,ranges,minVals
```

5. 模型评价

我们需要使用训练集来训练一个 k 邻近算法模型，并使用测试集进行模型评价。我们这里使用 90%的样本进行训练，10%的样本进行测试。

```
def datingClassTest():
    hoRatio = 0.10   # 测试集占总样本的比例
    datingDataMat,datingLabels = file2matrix('datingTestSet.txt')
    normMat,ranges,minVals = autoNorm(datingDataMat)
    m = normMat.shape[0]
    numTestVecs = int(m*hoRatio) # 测试集的样本数
    errorCount = 0.0
    for i in range(numTestVecs):
        classifierResult = classify(normMat[i,:],normMat[numTestVecs:m,:],
                                    datingLabels[numTestVecs:m],3)
        print("The classifier cam back with : {}, the real answer is :{}."
              .format(classifierResult,datingLabels[i]))
        if (classifierResult != datingLabels[i]):errorCount += 1.0
    print("The total error rate is : {}"
          .format(errorCount/float(numTestVecs)))
```

我们发现模型正确率为 95%。

```
……

The classifier cam back with : 1, the real answer is :1.

The classifier cam back with : 3, the real answer is :1.

The total error rate is : 0.05
```

6. 模型预测

```
def classifyPerson():
    resultList = ['not at all','in small doses','in large doses']
    percentTats = float(input(\
        "percentage of time spend playing video games?"))
    ffMiles = float(input("frequent flier miles earned per year?"))
    iceCream = float(input("liters of ice cream consumed per year?"))
    datingDataMat, datingLabels = file2matrix('datingTestSet.txt')
    normMat, ranges, minVals = autoNorm(datingDataMat)
    inArr = array([ffMiles,percentTats,iceCream])
    classifierResult = classify((inArr-\

minVals)/ranges,normMat,datingLabels,3)
    print("You will probably like this person: {}"
          .format(resultList[classifierResult-1]))
```

percentage of time spend playing video games?15

frequent flier miles earned per year?40000

liters of ice cream consumed per year?1

You will probably like this person: in large doses

注意，预测时归一化使用的是训练集的最小值和范围。

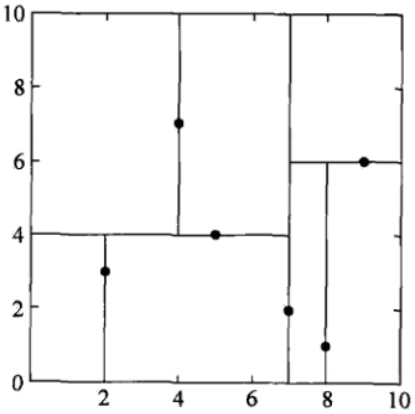
三、算法深入

K 近邻法的实现需要考虑如何快速搜索 k 个最近邻点。最简单的实现方法使线性扫描(linear scan)，即用输入实例与样本空间中的每一个点进行距离计算。我们上面实现 k 近邻法的 python 程序的实现方法就是线性扫描。当训练集很大时，计算非常耗时。

为了提高 k 近邻搜索的效率，可以考虑使用特殊的结构存储训练数据。常用的有 kd 树方法。Kd 树是二叉树，我们这里介绍一下如何构造平衡 kd 树。

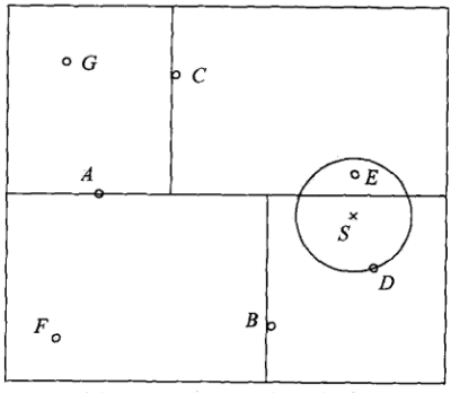
一个二维空间的数据集：$T = \{(2,3)^T, (5,4)^T, (9,6)^T, (4,7)^T, (8,1)^T, (7,2)^T\}$

如右图，$x^{(1)}$坐标的中位数是 8，以平面$x^{(1)} = 7$将空间分为左、右两个子矩形。对于左边的矩形，$x^{(2)}$坐标的中位数是 4，以平面$x^{(2)} = 4$切割矩形。对于右边的矩形，平面$x^{(2)}$坐标的中位数是6，以平面$x^{(2)} = 6$切割矩形。直到子区域没有实例存在时停止。于是我们构造了一个平衡 kd 树。



我们称（7，2）为父节点，（5，4）、（9，6）为子节点。同时（2，3），（4，7）为（5，4）的子节点。

那么，kd 树是如何减少搜索的计算量的呢?



如左图，假设我们有一个实例 S，寻找其最近邻。S 落在 D 所在的矩形，所以我们以$|SD|$为半径作圆，S 的最近邻一定落在圆内。返回 D 的父结点 B，在 B 的另一个子节点 F 中寻找，由于 F 不与圆相交，所以忽略 F 区域内的点。返回 B 的父节点 A，在 A 的子节点 C 的子节点 G 和 E 中寻找。G 与圆不相交，同样忽略。计算$|SE|$，得到最近邻 E。也就是说，kd 树可以减少去 F、G 区域的搜索，进而缩短训练时间。