

一 KNN 算法的概述

KNN 算法的核心思想：如果一个样本在特征空间中的 k 个最相邻的样本中的大多数是属于某一个类别，则该样本也属于这个类别，并具有这个类别上样本的特性。这就要求待分类的样本周围的 k 个邻居样本都已经准确分类，该样本所属的类别一般按照 k 邻居的属性进行投票，得票最高的属性就是待分类样本的属性。

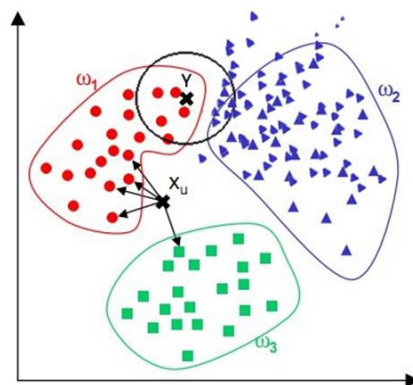
KNN 算法我的总结就是：“近朱者赤近墨者黑”



图一

如图二所示，用 KNN 算法判别点 X_u 属于 w_1 , w_2 , w_3 三块区域中的哪一块，选取距离 X_u 最近的 5 个点作为邻居集，如图所示其中 4 个邻居属于 w_1 ，只有一个邻居属于 w_2 ，所以我们选取 w_1 作为该点所属的区域。

但是这种算法也有其弊端，举例说明。如图二所示，现在运用该算法判断 Y 点所属的区域。我们选取距离该点为 r 以内的点为它的邻居点集，如图所示黑色的圆内的点都属于我们所选的邻居点集。但是很明显其邻居点集中蓝色的点占绝大多数，按照我们刚刚说过的判定原则 Y 点应该属于 w_2 区域，但是很明显 Y 很靠近 w_1 区域，也就是说我们判断失败了，失败的原因是数据分布的密集程度不一样，那有什么好的解决方法呢？不用说，先人的智慧都是无穷的。



图二

我们可以通过对每一个邻居设置权值来改变最终的结果，就像现实生活中，领导会听取一些意见，但是影响他决策的人往往就他身边的几个谋士，那又有一个问题出来了，如何设置权重，其实很简单，正如上面举得听取意见的例子。当然是和领导走的近，领导的亲信他们的影响比较大。运用到算法中就是，谁和待分类目标最近，谁的权重最大，距离越

远，其所占的权重越小，对分类的影响就越低，根据距离加上权重比如： $1/d$ (d : 距离)。按照这个思路， Y 就能被准确的分类。

算法的优点：简单，易于理解，容易实现，通过对 K 的选择可具备丢噪音数据的健壮性。

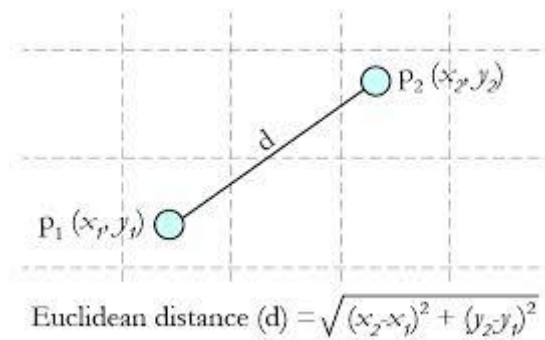
算法的缺点：需要大量空间储存所有已知实例，算法复杂度高（需要比较所有已知实例与要分类的实例）当其样本分布不平衡时，比如其中一类样本过大（实例数量过多）占主导的时候，新的未知实例容易被归类为这个主导样本，因为这类样本实例的数量过大，但这个新的未知实例实际并未接近目标样本。

二 算法的基本步骤

1. 读取训练集（已知属性的样本集）和测试集（需要待分类的样本集）
2. 选择参数 K (K 的值就是要选择的邻居的个数，一般为奇数，便于投票出结果，不会出现 1: 1 的情况，建议选择 3, 5, 7, 9)
3. 根据少数服从多数的投票法则(majority-voting)，让未知实例归类为 K 个最邻近样本中最多数的类别。

三 距离的确定

关于距离的衡量方法，一般选用欧式距离。


$$\text{Euclidean distance } (d) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
$$E(x, y) = \sqrt{\sum_{i=0}^n (x_i - y_i)^2}$$

四 python 实战应用

1. 首先读取数据（数据集在文件中，请自行下载观看），分为测试集和训练集。

```
train = []
tests = []
a = float(5/12)
with open('irisdata.txt', 'rt') as file:
    lines = csv.reader(file)
    datas = list(lines)
    # print(len(datas)) 测试
    for x in range(len(datas)-1):
        for y in range(4):
            datas[x][y] = float(datas[x][y])
            if random.random() < a:
                train.append(datas[x])
            else:
                tests.append(datas[x])
# print(tests, train)
```

2. 我们需要定义一个欧式函数，因为求距离会一直用到。

```
def get_distance(neighbors1, neighbors2, leng):
    distance = 0
    #计算所有维度的差的平方和，此处传入的leng其实就是数据的维度
    #举例[3, 4, 5, 8]的维度就是4
    for x in range(leng):
        distance += math.pow((neighbors1[x]-neighbors2[x]), 2)
    return math.sqrt(distance)
```

4. 下面开始写主程序，第一步根据距离选取 k 个邻居集。

```
for x in range(len(tests)):
    # traintrain[x]
    #获取某个待分类数据的k个邻居
    distances = []
    leng = len(tests[x]) - 1 # 测试集的维度
    for y in range(len(train)): # 对训练集中的每一个数据计算它到测试元的距离
        # testone
        dist = get_distance(tests[x], train[y], leng)
        distances.append((train[y], dist))
        # distances.append(dist)
    distances.sort(key=operator.itemgetter(1)) # 对距离从小到大进行排序
    neighbors_t = []
    for x in range(k):
        neighbors_t.append(distances[x][0])
```

5. 根据邻居集的属性进行投票，选择票数最多的属性作为预测值。

```
classVotes = {}
for x in range(len(neighbors_t)):
    response = neighbors_t[x][-1] # 提取邻居的属性
    if response in classVotes: # 对k个邻居进行属性投票
        classVotes[response] += 1
    else:
        classVotes[response] = 1
# print(classVotes)
sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), r
# print(sortedVotes)
result = sortedVotes[0][0]
pre_results.append(result)
```

- 6 最后显示判断的准确率（运行这么多次，这次效果最好，一般在 93 到 96 之间）。
（该处代码很简单不显示）

```
D:\anaconda\python.exe D:/python运营代码/python/机器学习/临近取样/模式识别作业.py
预测正确率 98.85057471264368
```

详细代码请看附录。