第5章 KNN算法

5.1 KNN算法简介

5.1.1 KNN算法原理

K最近邻(K-Nearest Neighbor,KNN)分类算法是一个理论上比较成熟的方法,也是最简单的机器学习算法之一。该算法最初由Cover和Hart于1968年提出,它根据距离函数计算待分类样本X和每个训练样本间的距离(作为相似度),选择与待分类样本距离最小的K个样本作为X的K个最近邻,最后以X的K个最近邻中的大多数样本所属的类别作为X的类别。

KNN算法中,所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。KNN方法虽然从原理上也依赖于极限定理,但在类别决策时,只与极少量的相邻样本有关。由于KNN方法主要靠周围有限的邻近的样本,而不是靠判别类域的方法确定所属类别的,因此对于类域的交叉或重叠较多的待分样本集来说,KNN方法较其他方法更为适合。

KNN算法大致包括如下3个步骤。

(1) 算距离:给定测试对象,计算它与训练集中的每个对象的距离。

(2) 找邻居:圈定距离最近的k个训练对象,作为测试对象的近邻。

(3) 做分类:根据这k个近邻归属的主要类别,来对测试对象分类。

因此最为关键的就是距离的计算。一般而言,定义一个距离函数$d(x,y)$,需要满足下面几个准则。

(1) $d(x,x)=0$

(2) $d(x,y)\geqslant 0$

(3) $d(x,y)=d(y,x)$

(4) $d(x,k)+d(k,y)\geqslant d(x,y)$

距离计算有很多方法,大致分为离散型特征值计算方法和连续型特征值计算法方法两大类。

1. 连续型数据的相似度度量方法

1) 闵可夫斯基距离

闵可夫斯基距离(Minkowski distance)是衡量数值点之间距离的一种常见的方法,假设数值点P和Q坐标为$P=(x_1,x_2,\cdots,x_n)$和$Q=(y_1,y_2,\cdots,y_n)$,则闵可夫斯基距离定义为

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

该距离最常用的 p 是 2 和 1，前者是欧几里得距离(Euclidean distance)，即 $\sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$，后者是曼哈顿距离(Manhattan distance)，即 $\left(\sum_{i=1}^{n} |x_i - y_i|\right)$。当 p 趋近于无穷大时，闵可夫斯基距离转化为切比雪夫距离(Chebyshev distance)，即 $\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} = \max_i |x_i - y_i|$。

2) 余弦相似度

为了解释余弦相似度，先介绍一下向量内积的概念。向量内积定义如下。

$$\text{Inner}(x,y) = <x,y> = \sum_i x_i y_i$$

向量内积的结果是没有界限的，一种解决方法是除以长度之后再求内积，这就是应用广泛的余弦相似度(cosine similarity)。

$$\text{CosSim}(x,y) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\ \sqrt{\sum_i y_i^2}} = \frac{<x,y>}{\|x\|\ \|y\|}$$

余弦相似度与向量的幅值无关，只与向量的方向相关。需要说明的是，余弦相似度受到向量的平移影响，上式如果将 x 平移到 $x+1$，余弦值就会改变。怎样才能实现这种平移不变性？这就是下面提到的皮尔逊相关系数(pearson correlation)，或简称为相关系数。

3) 皮尔逊相似系数

$$\text{Corr}(x,y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2}\ \sqrt{\sum_i (y_i - \bar{y})^2}}$$

$$= \frac{<x - \bar{x}, y - \bar{y}>}{\|x - \bar{x}\|\ \|y - \bar{y}\|} = \text{CosSim}(x - \bar{x}, y - \bar{y})$$

皮尔逊相似系数具有平移不变性和尺度不变性，它计算出了两个向量的相关性，$\bar{x}$、$\bar{y}$ 表示 x、y 的平均值。

2. 离散型数据的相似性度量方法

1) 汉明距离

两个等长字符串 s1 和 s2 之间的汉明距离(hamming distance)定义为将其中一个变为另外一个所需要做的最小替换次数。例如，1011101 与 1001001 之间的汉明距离是 2，2 143 896 与 2 233 796 之间的汉明距离是 3，toned 与 roses 之间的汉明距离是 3。

在一些情况下，某些特定的值相等并不能代表什么。举个例子，用 1 表示用户看过该电影，用 0 表示用户没有看过，那么用户看电影的信息就可用 0、1 表示成一个序列。考虑到电影基数非常庞大，用户看过的电影只占其中非常小的一部分，如果两个用户都没有看过某一部电影(两个都是 0)，并不能说明两者相似。反而言之，如果两个用户都看过某一部电影(序列中都是 1)，则说明用户有很大的相似度。在这个例子中，序列中等于 1 所占的权重应该远远大于 0 的权重，这就引出下面要说的杰卡德相似系数(Jaccard similarity)。

2) 杰卡德相似系数

在上面的例子中，用 $M11$ 表示两个用户都看过的电影数目，$M10$ 表示用户 A 看过而用

户 B 没有看过的电影数目，$M01$ 表示用户 A 没看过而用户 B 看过的电影数目，$M00$ 表示两个用户都没有看过的电影数目。Jaccard 相似性系数可以表示如下。

$$J = \frac{M11}{M01 + M10 + M11}$$

5.1.2 KNN 算法应用举例

实现 KNN 算法的步骤如下。

(1) 初始化距离为最大值。

(2) 计算测试样本和每个训练样本的距离 dist。

(3) 得到目前 k 个最近邻样本中的最大距离 maxdist。

(4) 如果 dist 小于 maxdist，则将该训练样本作为 k 最近邻样本。

(5) 重复步骤(2)、(3)和(4)，直到测试样本和所有训练样本的距离都计算完毕。

(6) 统计 k 个最近邻样本中每个类别出现的次数。

(7) 选择出现频率最大的类别作为测试样本的类别。

【例 5.1】 表 5-1 为进行 KNN 分类算法的简单数据表，数据序列号为 $T_1 - T_{10}$，每条数据有 1、2 两个特征，以及它所属的类别。对于测试样本 $T=\{18,8\}$，采用 KNN 算法求其所属类别。

表 5-1 训练集

序　　号	特征 1	特征 2	类　　别
T_1	2	4	L1
T_2	4	3	L2
T_3	10	6	L3
T_4	12	9	L2
T_5	3	11	L3
T_6	20	7	L2
T_7	22	5	L2
T_8	21	10	L1
T_9	11	2	L3
T_{10}	24	1	L1

解：本例采用欧几里得距离度量方法，且设 $k=4$。

$d(T,T_1) = \sqrt{(18-2)^2+(8-4)^2} = 16.49$；$d(T,T_2) = \sqrt{(18-4)^2+(8-3)^2} = 14.87$

$d(T,T_3) = \sqrt{(18-10)^2+(8-6)^2} = 8.25$；$d(T,T_4) = \sqrt{(18-12)^2+(8-9)^2} = 6.08$

$d(T,T_5) = \sqrt{(18-3)^2+(8-11)^2} = 15.3$；$d(T,T_6) = \sqrt{(18-20)^2+(8-7)^2} = 2.24$

$d(T,T_7) = \sqrt{(18-22)^2+(8-5)^2} = 5$；$d(T,T_8) = \sqrt{(18-21)^2+(8-10)^2} = 3.61$

$d(T,T_9) = \sqrt{(18-11)^2+(8-2)^2} = 9.22$；$d(T,T_{10}) = \sqrt{(18-24)^2+(8-1)^2} = 9.22$

所以距离 T 最近的 4 个样本为$\{T_6,T_8,T_7,T_4\}$，它们对应的标签为$\{L_2,L_1,L_2,L_2\}$，所以 T 的标签为 L_2。

【例 5.2】 对于训练集 T 如表 5-2 所示，用 KNN 算法对实例 $x=(1,2)$进行分类。其

中采用欧几里得距离衡量实例间的相似度，$k=5$，采样多数表决作为分类决策规则。

表 5-2　训练样本集

实　　例	横　坐　标	纵　坐　标	实例所属类
1	0.189 710 406	0.495 005 825	c1
2	0.147 608 222	0.054 974 147	c1
3	0.000 522 375	0.865 438 591	c1
4	0.527 680 069	0.479 523 385	c1
5	0.801 347 606	0.227 842 936	c1
6	0.738 640 292	0.585 987 036	c1
7	0.246 734 526	0.666 416 217	c1
8	0.083 482 814	0.625 959 785	c1
9	0.660 944 558	0.729 751 855	c1
10	0.769 029 085	0.581 446 488	c1
11	1.725 421 437	0.968 593 022	c2
12	1.689 711 349	0.418 810 168	c2
13	1.104 582 683	1.259 766 77	c2
14	0.063 982 032	1.229 426 838	c2
15	0.979 139 978	0.385 020 792	c2
16	0.085 304 822	1.270 395 834	c2
17	0.563 733 712	1.077 193 356	c2
18	1.390 326 079	0.998 232 027	c2
19	1.071 602 112	0.890 366 331	c2
20	0.247 864 555	0.980 714 587	c2
21	1.694 938 712	1.920 935 475	c3
22	2.843 799 364	0.246 213 621	c3
23	2.213 524 961	0.190 213 502	c3
24	2.356 676 968	1.540 132 256	c3
25	2.817 425 118	0.903 918 194	c3
26	1.401 204 561	1.944 595 219	c3
27	0.075 684 544	2.526 619 837	c3
28	1.988 424 186	0.992 486 986	c3
29	2.695 458 414	0.354 465 595	c3
30	1.394 519 825	2.291 871 235	c3
31	3.272 816 156	0.400 886 161	c4
32	0.835 786 696	3.620 614 236	c4
33	2.701 564 709	1.873 872 8	c4
34	3.648 529 897	0.416 046 299	c4
35	3.579 766 702	0.285 811 251	c4
36	3.588 765 404	0.786 632 765	c4
37	2.021 712 57	3.045 703 547	c4
38	0.323 449 693	3.108 962 146	c4
39	0.436 616 848	3.303 235 431	c4
40	2.413 871 935	2.104 409 863	c4

解：

待分类实例与 T 中实例的距离如表 5-3 所示，40 个训练样本距离按照行优先的顺序排列。

表 5-3　距离结果表

实　　例	实例 x 与各实例的距离	实例所属类
1	1.709 262 032	c1
2	2.123 604 792	c1
3	1.512 013 595	c1
4	1.592 148	c1
5	1.783 256 413	c1
6	1.437 964 381	c1
7	1.531 618 288	c1
8	1.651 662 879	c1
9	1.314 720 1	c1
10	1.437 234 021	c1
11	1.260 966 54	c2
12	1.725 068 993	c2
13	0.747 584 625	c2
14	1.212 399 536	c2
15	1.615 113 922	c2
16	1.170 038 251	c2
17	1.020 735 214	c2
18	1.075 124 886	c2
19	1.111 941 43	c2
20	1.266 747 994	c2
21	0.699 421 913	c3
22	2.544 673 409	c3
23	2.178 983 708	c3
24	1.432 498 076	c3
25	2.122 364 103	c3
26	0.405 012 086	c3
27	1.063 808 025	c3
28	1.411 405 273	c3
29	2.362 702 416	c3
30	0.490 749 132	c3
31	2.779 003 121	c4
32	1.628 912 678	c4
33	1.706 232 848	c4
34	3.086 036 283	c4
35	3.097 360 054	c4
36	2.859 014 929	c4
37	1.461 982 381	c4
38	1.299 044 788	c4
39	1.419 796 875	c4
40	1.417 721 859	c4

由表 5-3 可知，c1 类有 0 个，c2 类有 2 个，c3 类有 3 个，c4 类有 0 个，则可判定 x 属于 c3 类。

5.2 KNN 算法的特点及改进

5.2.1 KNN 算法的特点

KNN 算法优点如下。

(1) 算法思路较为简单，易于实现。

(2) 当有新样本要加入训练集中时，无须重新训练(即重新训练的代价低)。

(3) 计算时间和空间线性于训练集的规模，对某些问题而言是可行的。

KNN 算法缺点如下。

(1) 分类速度慢。KNN 算法的时间复杂度和空间复杂度会随着训练集规模和特征维数的增大而快速增加，因此每次新的待分类样本都必须与所有训练集一同计算比较相似度，以便取出靠前的 K 个已分类样本，KNN 算法的时间复杂度为 $O(m * n)$，这里 m 是特征个数，n 是训练集样本的个数。

(2) 各属性的权重相同，影响准确率。当样本不均衡时，如一个类的样本容量很大，而其他类的样本容量很小时，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。该算法只计算“最近的”邻居样本，如果某一类的样本数量很大，那么有可能目标样本并不接近这类样本，却会将目标样本分到该类下，从而影响分类准确率。

(3) 样本库容量依赖性较强。

(4) K 值不好确定。K 值选择过小，导致近邻数目过少，会降低分类精度，同时也会放大噪声数据的干扰；K 值选择过大，如果待分类样本属于训练集中包含数据较少的类，那么在选择 K 个近邻的时候，实际上并不相似的数据也被包含进来，从而造成噪声增加而导致分类效果的降低。

5.2.2 KNN 算法的改进策略

1. 从降低计算复杂度的角度

当样本容量较大以及特征属性较多的时候，KNN 算法分类的效率就将大大地降低。可以采用的改进方法如下。

(1) 进行特征选择。使用 KNN 算法之前对特征属性进行约简，删除那些对分类结果影响较小(或不重要)的特征，则可以加快 KNN 算法的分类速度。

(2) 缩小训练样本集的大小。在原有训练集中删除与分类相关性不大的样本。

(3) 通过聚类，将聚类所产生的中心点作为新的训练样本。

2. 从优化相似性度量方法的角度

很多 KNN 算法基于欧几里得距离来计算样本的相似度，但这种方法对噪声特征非常敏感。为了改变传统 KNN 算法中特征作用相同的缺点，可以再度量相似度距离公式中给特征赋予不同权重，特征的权重一般根据各个特征在分类中的作用而设定，计算权重的方法有很多，例如信息增益的方法。另外，还可以针对不同的特征类型，采用不同的相似度度量

公式，更好地反映样本间的相似性。

3. 从优化判决策略的角度

传统的 KNN 算法的决策规则存在的缺点是，当样本分布不均匀（训练样本各类别之间数目不均衡，或者即使基本数目接近，但其所占区域大小不同）时，只按照前 k 个近邻顺序而不考虑它们的距离会造成分类不准确，采取的方法很多，例如可以采用均匀化样本分布密度的方法加以改进。

4. 从选取恰当 K 值的角度

由于 KNN 算法中的大部分计算都发生在分类阶段，而且分类效果很大程度上依赖于 K 值的选取，到目前为止，没有成熟的方法和理论指导 k 值的选择，大多数情况下需要通过反复试验来调整 K 值的选择。

5.3 KNN 源代码结果分析

KNN 算法的 Java 源程序如下所示，包含 3 个文件，即 KNN.java、KNNNode.java 和 TestKNN.java，相关程序和实验数据可从 github 中下载，网址为 https://github.com/guanyao1/knn.git。

1. KNN.java

```
package knn;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.PriorityQueue;
/* KNN算法主体类
 * @author liuwei
 */
public class KNN {
    //设置优先级队列的比较函数，距离越大，优先级越高
    private Comparator<KNNNode> comparator = new Comparator<KNNNode>(){
        @Override
        public int compare(KNNNode o1,KNNNode o2){
            if(o1.getDistance() >= o2.getDistance()){
                return 1;
            }else{
                return 0;
            }
        }
    }
    /*获取K个不同的随机数
     * @param k随机数的个数
     * @param maxRange随机数最大的范围
```

```
     * @return 生成的随机数数组
     */
    public List<Integer> getRandNum(int k,int maxRange){
        List<Integer> rand = new ArrayList<Integer>(k);
        for (int i = 0; i < k; i++) {
            int temp = (int) (Math.random() * maxRange);
            if(!rand.contains(temp)){
                rand.add(temp);
            }else{
                i--;
            }
        }
        return rand;
    }
    /*计算测试元组与训练元组之间的距离
     * @param testDis 测试元组
     * @param tranDis 训练元组
     * @return 距离值
     */
    public double calDistance(List<Double> testDis,List<Double> tranDis){
        double distance = 0.00;          //测试元组与训练元组之间的距离
        for (int i = 0; i < testDis.size(); i++) {
            distance += (testDis.get(i) - tranDis.get(i)) * (testDis.get(i) - tranDis.
            get(i));
        }
        return distance;
    }
    /*执行 KNN 算法,获取测试元组的类别
     * @param datas 训练数据集
     * @param testData 测试元组
     * @param k 设定的 K 值
     * @return 测试元组的类别
     */
        public String knn(List<List<Double>> datas,List<Double> testData,int k){
        PriorityQueue<KNNNode> pq = new PriorityQueue<KNNNode>(k,comparator);
//使用指定的初始容量 k 创建一个 PriorityQueue,
//并根据指定的比较器 comparator 对元素进行排序
        List<Integer> randNum = getRandNum(k, datas.size());
        for (int i = 0; i < k; i++) {
            int index = randNum.get(i);
            List<Double> currData = datas.get(index);
            String c = currData.get(currData.size() - 1).toString();
            KNNNode node = new KNNNode(index, calDistance(testData, currData), c);
            pq.add(node);
        }
```

```
            for (int i = 0; i < datas.size(); i++) {
                List<Double> t = datas.get(i);
                double distance = calDistance(testData, t);
                KNNNode topNode = pq.peek();
                if(topNode.getDistance() > distance){
                    pq.remove();
                    pq.add(new KNNNode(i, distance, t.get(t.size() - 1).toString()));
                }
            }
            return getMostClass(pq);
        }
        /*获取所得到的K个最近邻元组的种类
         * @param pq存储K个最近近邻元组的优先级队列
         * @return多数类的名称
         */
        private String getMostClass(PriorityQueue<KNNNode> pq){
            Map<String,Integer> classCount = new HashMap<String, Integer>();
            for (int i = 0; i < pq.size(); i++) {
                KNNNode node = pq.remove(); //获取并移除此队列的头
                String category = node.getCategory();
                if(classCount.containsKey(category)){
                    classCount.put(category, classCount.get(category) + 1);
                }else{
                    classCount.put(category, 1);
                }
            }
            int maxIndex = -1;
            int maxCount = 0;
            Object[] classes = classCount.keySet().toArray();
            for (int i = 0; i < classes.length; i++) {
                if(classCount.get(classes[i]) > maxCount){
                    maxIndex = i;
                    maxCount = classCount.get(classes[i]);
                }
            }
            return classes[maxIndex].toString();
        }
    }
```

2. KNNNode.java

```
package knn;
// KNN结点类,用来存储最近邻的K个元组的相关信息
public class KNNNode {
    private int index;                    //元组标号
```

```
    private double distance;                //与测试元组的距离
    private String category;                //所属类别

    public KNNNode(int index, double distance, String category) {
        this.index = index;
        this.distance = distance;
        this.category = category;
    }
    public int getIndex() {
        return index;
    }
    public void setIndex(int index) {
        this.index = index;
    }
    public double getDistance() {
        return distance;
    }
    public void setDistance(double distance) {
        this.distance = distance;
    }
    public String getCategory() {
        return category;
    }
    public void setCategory(String category) {
        this.category = category;
    }
}
```

3. TestKNN.java

```
package knn;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
// KNN 算法测试类
public class TestKNN {

/* 程序执行入口
 * @param args
 */
public static void main(String[] args) {
```

```
        TestKNN testKnn = new TestKNN();
        String dataFile = "C:\\Users\\wang4\\Desktop\\data.txt";
        String testFile = "C:\\Users\\wang4\\Desktop\\test.txt";
        try {
            List<List<Double>> datas = new ArrayList<List<Double>>();
            List<List<Double>> testDatas = new ArrayList<List<Double>>();
            testKnn.read(datas, dataFile);
            testKnn.read(testDatas, testFile);
            KNN knn = new KNN();
            for (int i = 0; i < testDatas.size(); i++) {
                List<Double> test = testDatas.get(i);
                System.out.print("测试元组: ");
                for (int j = 0; j < test.size(); j++) {
                    System.out.print(test.get(j) + " ");
                }
                    System.out.print("类别为: ");
                    System.out.println(Math.round(Float.parseFloat((knn.knn(datas, test, 3)))));
                }
            } catch (IOException e) {
                e.printStackTrace();
            }

        }
        /*从数据文件中读取数据
         * @param datas 存储数据的集合对象
         * @param path 数据文件的路径
         * @throws IOException
         */
        public void read(List<List<Double>> datas,String path) throws IOException{
            BufferedReader br = null;
            List<Double> list = null;
            try {
                br = new BufferedReader(new FileReader(new File(path)));
                String data = br.readLine();//读取一个文本行
                while(data != null){
                    String t[] = data.split(" ");
                    list = new ArrayList<Double>();
                    for (int i = 0; i < t.length; i++) {
                        list.add(Double.parseDouble(t[i]));
                    }
                    datas.add(list);
                    data = br.readLine();
                }
            } catch (FileNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
```

程序运行界面如图 5-1 所示。

从上面的测试情况可以直观地看出，KNN 算法的预测结果的正确率还是很高的。

```
测试元组: 1.0 1.1 1.2 2.1 0.3 2.3 1.4 0.5 类别为: 1
测试元组: 1.7 1.2 1.4 2.0 0.2 2.5 1.2 0.8 类别为: 1
测试元组: 1.2 1.8 1.6 2.5 0.1 2.2 1.8 0.2 类别为: 1
测试元组: 1.9 2.1 6.2 1.1 0.9 3.3 2.4 5.5 类别为: 0
测试元组: 1.0 0.8 1.6 2.1 0.2 2.3 1.6 0.5 类别为: 1
测试元组: 1.6 2.1 5.2 1.1 0.8 3.6 2.4 4.5 类别为: 0
```

图 5-1　KNN 程序运行结果

5.4　基于阿里云数加平台的 KNN 算法应用实例

KNN 算法属于分类算法，其总体目的是通过训练有标签数据，给无标签数据赋标签。它通过计算预测数据与训练数据的距离，选出距离最近的 K 条记录，在这 K 条记录中，某一个或某几个变签个数最多的为那个数据的预测标签，可以自行建立一个带标签的训练表和一个不带标签的预测表来做 KNN 分类算法。数据图如图 5-2 和图 5-3 所示。

f0	f1	f2	f3	label
1	2	2	2	good
1	3	3	3	good
1	4	5	3	bad
0	3	6	2	bad
0	4	4	2	bad
0	2	4	2	good
1	3	2	3	bad
1	4	3	2	good
0	2	5	2	bad
1	4	6	3	bad

图 5-2　训练集数据表

f0	f1	f2	f3
0	3	4	3
1	2	6	3
0	3	5	2
1	4	2	2

图 5-3　预测集数据表

操作流程图如图 5-4 所示，图上方左边为训练表，右边为预测表。

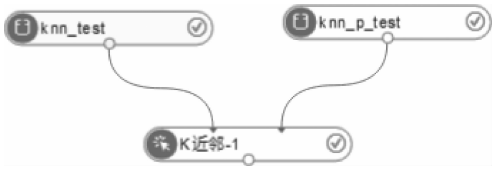

图 5-4　操作流程图

K 近邻的字段设置与参数设置如图 5-5 所示，其中，字段设置为：选择训练表特征列必选，选择训练表中所需的特征列即可，这里选择的是“f0”“f1”“f2”“f3”；选择训练表的标签列必选，选择训练表中的标签列即可，这里选择的是 label 列；选择预测表特征列可选，当预测表的特征列与训练表的特征列重名时，这里便可不选。产出表附加 ID 列可选，它是用来标识该列的身份，进而知道某列对应的预测值，默认预测表特征列都作为附加 ID 列，这里选

择的是默认值。参数设置的近邻个数，可自行设置，这里选择的是5。

图 5-5　K 近邻的字段设置与参数设置图

实验结果如图5-6所示，得到的是对预测表的标签预测。

f0	f1	f2	f3	prediction_result	prediction_score	prediction_detail
0	3	4	3	bad	0.6	{"bad": 0.6, "good": 0.4}
1	2	6	3	bad	0.8	{"bad": 0.8, "good": 0.2}
0	3	5	2	bad	0.8	{"bad": 0.8, "good": 0.2}
1	4	2	2	good	0.6	{"bad": 0.4, "good": 0.6}

图 5-6　K 近邻的实验结果

5.5　小　　结

KNN算法根据距离函数计算待分类样本 $\boldsymbol{X}$ 和每个训练样本的距离（作为相似度），选择与待分类样本距离最小的 $\boldsymbol{K}$ 个样本作为 $\boldsymbol{X}$ 的 $\boldsymbol{K}$ 个最近邻，最后以 $\boldsymbol{X}$ 的 $\boldsymbol{K}$ 个最近邻中的大多数样本所属的类别作为 $\boldsymbol{X}$ 的类别。

如何度量样本之间的距离（或相似度）是KNN算法的关键步骤之一。常见的相似度度量方法包括闵可夫斯基距离（当参数 $p=2$ 时为欧几里得距离，参数 $p=1$ 时为曼哈顿距离）、余弦相似度、皮尔逊相似系数、汉明距离和杰卡德相似系数等。

思 考 题

1. 简述 KNN 算法的原理。
2. 简述 KNN 算法的优缺点。
3. 简述 KNN 算法的改进策略。
4. 基于阿里云数加平台对 KNN 算法进行操作。