《模式识别》作业 1

1 问题描述

k 近邻算法是模式识别的基本算法之一。该算法的基本思想是存在一个训练样本集，且样本集中每个数据都存在标签。输入没有标签的新数据后，计算新数据与训练集数据的模式特征的距离，在距离最近的 k 各数据中统计标签的频率，频率最高的标签即为新数据的标签。其中：距离度量为可测的，k 为模型的超参数。
使用文件 iris.txt 中的数据作为实验数据。在该数据文件中，每行为一个模式，前四个数据为模式特征，最后一个数据为模式类别标签。
针对下列问题，给出具有统计意义的答案及解释：
1）最优 k 值
2）最优预测正确率

2 基本思路

实现 k 近邻分类器：计算测试集中的一个实例到训练集所有实例的欧氏距离，将距离按从小到大排序，取距离最小的 k 个实例，即 k 个近邻。统计 k 近邻实例中标签的频率，频率最高的标签即为此训练集实例的预测标签。

3 算法

3.1 算法描述

1）数据集分类:读取 txt 文件中的数据集,按 8:2 分为训练集 train_set 和测试集 test_set。在抽取数据时，按照标签平均抽样，每种标签抽取 40 条训练集数据、10 条测试集数据，训练集和测试集各自合并三种标签之后打乱数据的顺序。
2）k 近邻分类器： 计算测试集实例 test_set[i]到训练集每个实例的欧式距离并在数组 distances 中存放实例与距离的对应关系，将距离由小到大排序，取前 k 个最小距离的实例。统计 k 个最小距离实例的标签频率，频率最高的标签即为此测试集实例 test_set[i]的预测标签。
3）取不同的 k 值，对测试集 train_set 中每个实例 test_set[i]都执行 k 近邻算法，将 knn 函数返回的预测结果与数据集中的标签比较，在 accuracy 数组中记录出不同 k 值下的预测准确率。
4）由于测试集数据不多，对上述步骤执行 5 次，计算平均准确率
5）找出准确率最高的 k 值，输出这个 k 值以及最高准确率

3.2 算法实现

```
import numpy
from sklearn.model_selection import train_test_split
import random
import matplotlib.pyplot as plt
import operator
```

```
# 从数据集中抽取训练集和验证集
def data_sampling():
    s = numpy.loadtxt('iris.txt', dtype=float, delimiter=",")  # 读取 txt 文件数据
    # 从三类中均匀取值
    s1 = s[0:50, :] # label=1
    s2 = s[50:100, :] # label=2
    s3 = s[100:150, :] # label=3
    x1_train, x1_test = train_test_split(s1, test_size=10)
    x2_train, x2_test = train_test_split(s2, test_size=10)
    x3_train, x3_test = train_test_split(s3, test_size=10)
    # 合并抽取出的训练集和验证集
    x_train = numpy.r_[x1_train, x2_train, x3_train]
    x_test = numpy.r_[x1_test, x2_test, x3_test]
    # 打乱训练集
    index = [i for i in range(120)]
    random.shuffle(index)
    train_set = x_train[index]
    # 打乱验证集
    index = [i for i in range(30)]
    random.shuffle(index)
    test_set = x_test[index]
    return train_set, test_set

# 计算模式特征的欧氏距离
def compute_distance(test, train, row):
    distance = 0
    for i in range(row):
        distance += (test[i] - train[i])**2
    return distance**0.5

# k 近邻
def knn(train_set, test, k):
    # 计算测试集实例到训练集实例的欧式距离
    distances = [] # 存放训练集实例与距离
    for i in range(len(train_set)):
        dist = compute_distance(test, train_set[i], len(test)-1)
        distances.append((train_set[i], dist))
    #按距离由小到达进行排序
    distances.sort(key=operator.itemgetter(1))

    #在所有距离中取前 k 个，即 k 近邻
    neighbors = [] # 存放 k 近邻的训练集实例
    for i in range(k):
```

```
        neighbors.append(distances[i][0])

    #统计 k 近邻中各标签的频率
    label = {}
    for i in range(len(neighbors)):
        train_label = neighbors[i][4] # 训练集实例的标签
        if train_label in label:
            label[train_label] += 1
        else:
            label[train_label] = 1
    # 按标签频率降序排列
    result = sorted(label.items(), key = operator.itemgetter(1), reverse=True)
    return result[0][0] # 返回标签中的众数


def f():
    accuracy = [0] * 50

    # 多次验证
    for j in range(0, 5):
        # 训练集：测试集 = 8:2
        train_set, test_set = data_sampling()
        # k 分别取值 1~50 进行训练测试
        for k in range(0, 50):
            for i in range(len(test_set)): # 验证测试集中每一条实例
                train_result = knn(train_set, test_set[i], k + 1)
                if train_result == test_set[i][4]:

                    # 训练得到的结果与训练实例的标签是否一致
                    accuracy[k] = accuracy[k] + 1 # 若一致,准确度+1

    # 画图
    x = range(1, 51)  # x 轴范围
    res = numpy.c_[x, accuracy]
    plt.scatter(res[:,0], res[:,1])  # 散点图
    plt.title("kNN")  # 图形标题
    plt.xlabel('k value')  # x 轴名称
    plt.ylabel('accuracy')  # y 轴名称
    plt.xticks(x)  # 设置 x 轴刻度
    plt.show()  # 显示图形

    # 找到准确率最高的 k 值
    max_accuracy = accuracy[0] / (30 * 5)
    for i in range(1, 50):
        accuracy[i] = accuracy[i] / (30 * 5)
```

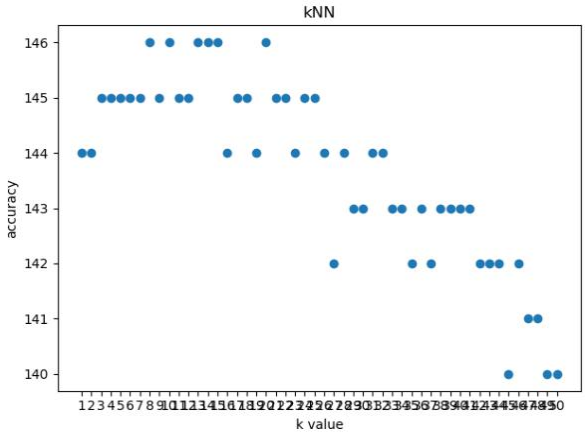
```
        if max_accuracy < accuracy[i]:
            max_accuracy = accuracy[i]

    # 打印正确率最大的 k 值
    for i in range(0, 50):
        if max_accuracy == accuracy[i]:
            best_k = i + 1
            print('最优 k 值：', best_k)
    print('最优预测正确率：', max_accuracy * 100, "%")

    return best_k, max_accuracy

#统计结果
def main():
    f_k=[]
    f_accuracy=[]
    # 记录每次运行的结果
    for i in range(10):
        print('第', i+1, '次运行结果：')
        best_k, best_accuracy=f()
        f_k.append(best_k)
        f_accuracy.append(best_accuracy)
    # 计算均值
    k_mean = numpy.mean(f_k)
    accuracy_mean = numpy.mean(f_accuracy)
    print('----10 次运行结果统计----')
    print('k 的均值：', k_mean, '正确率的均值：', accuracy_mean* 100, "%")
    print()

if __name__ == "__main__":
    main()
```

4 结果与分析

4.1 实验步骤

1）分析 k 近邻算法原理
2）运行 pycharm 软件
3）编写 knn 算法及整体程序
4）运行程序，记录结果
5）统计并分析程序运行结果

4.2 实验结果



第一次运行 k 与正确率的散点图

```
第 1 次运行结果：
最优k值： 8
最优k值： 10
最优k值： 13
最优k值： 14
最优k值： 15
最优k值： 20
最优预测正确率： 97.33333333333334 %
第 2 次运行结果：
最优k值： 5
最优k值： 9
最优k值： 11
最优预测正确率： 96.66666666666667 %
第 3 次运行结果：
最优k值： 7
最优k值： 8
最优k值： 12
最优k值： 16
最优k值： 17
最优预测正确率： 97.33333333333334 %

第 4 次运行结果：
最优k值： 9
最优k值： 10
最优k值： 11
最优k值： 12
最优k值： 13
最优预测正确率： 96.0 %
第 5 次运行结果：
最优k值： 7
最优k值： 9
最优k值： 10
最优k值： 11
最优k值： 15
最优k值： 17
最优k值： 19
最优预测正确率： 96.66666666666667 %
第 6 次运行结果：
最优k值： 11
最优预测正确率： 96.0 %
第 7 次运行结果：
最优k值： 15
最优k值： 16
最优预测正确率： 98.66666666666667 %
```

```
第 8 次运行结果：
最优k值： 5
最优k值： 6
最优预测正确率： 97.33333333333334 %
第 9 次运行结果：
最优k值： 17
最优预测正确率： 98.0 %
第 10 次运行结果：
最优k值： 11
最优预测正确率： 97.33333333333334 %
----10次运行结果统计----
k的均值： 14.1 正确率的均值： 97.13333333333333 %
```

4.3 结果分析

1）最优 k 值：14

2）最优预测正确率：97.13%

本次实验 k 值从 1 取到 50，实验结果表明，k 过小，容易受局部噪声影响；k 过大，可能会出现拖尾现象。每一次实验中最高准确率可能对应多个 k 值，对 10 次运行结果取均值得到最优 k 值为 14，其对应的最优预测正确率为 97.13%。knn 算法在样本不平衡或样本容量较小时误分的概率会增加，若要继续优化准确率，可以继续改进数据集抽样的方式。