中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	计科 2 班	专业（方向）	计算机科学与技术
学号	21311274	姓名	林宇浩

一、 实验题目

k 近邻算法实现文本分类

二、 实验内容

1. 算法原理

K 近邻算法（KNN）是一种基于实例的学习算法，主要用于分类和回归问题。它的基本思想是，如果一个样本在特征空间中的 k 个最相似的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN 算法的原理很简单，算法首先将所有的训练样本标记好类别，然后计算新样本与每个训练样本之间的距离（欧氏距离、曼哈顿距离等）。然后按照距离从小到大排序，选取距离最近的 k 个训练样本。最后根据这 k 个训练样本的类别，通过投票等方式决定新样本的类别。例如，如果 k=3，那么对于一个新样本，我们找到距离它最近的 3 个训练样本，然后看这三个训练样本属于哪个类别最多，就将新样本归为那个类别。KNN 算法的优点是简单易懂，不需要训练模型，对于大规模数据处理也很有效，同时也可以用于多分类问题。缺点是它需要保存所有的训练数据，对于高维数据需要降维处理以避免维数灾难。

2. 伪代码

对于每个测试样本 t：
    对于每个训练样本 i：
        计算测试样本 t 与训练样本 i 之间的距离 dist(t, i)；
    选择距离测试样本 t 最近的 k 个训练样本，记为 S(t)；
    对于测试样本 t：
        计算 S(t)中出现最多的类别标签，作为测试样本 t 的预测标签；
返回所有测试样本的预测标签。

3. 关键代码展示

```
import time

train_sentences = []  # 训练集样本句子
train_category = []  # 训练集样本类别
```

```
# 下面进行读取数据
file = open("Classification/train.txt", mode='r')
line = file.readline()
line = file.readline()
while line:
    record = 0
    length = len(line)
    for i in range(length):
        if line[i] == ' ':
            record += 1
            if record == 1:
                train_category.append(int(line[i + 1]))
            elif record == 3:
                train_sentences.append(line[i + 1:length - 1])
                break
    line = file.readline()

test_sentences = []  # 测试集样本句子
test_category = []  # 测试集样本类别

# 下面进行读取数据
file = open("Classification/test.txt", mode='r')
line = file.readline()
line = file.readline()
while line:
    record = 0
    length = len(line)
    for i in range(length):
        if line[i] == ' ':
            record += 1
            if record == 1:
                test_category.append(int(line[i + 1]))
            elif record == 3:
                test_sentences.append(line[i + 1:length - 1])
                break
    line = file.readline()


# 数据处理，使用 TF-IDF 处理句子
from sklearn.feature_extraction.text import TfidfVectorizer
transfer = TfidfVectorizer(smooth_idf=True, norm='l2')
train_data = transfer.fit_transform(train_sentences)
test_data = transfer.transform(test_sentences)
vocabulary = transfer.vocabulary_

def Calculate_Distance(data1, data2):  # 欧拉距离
    distance = 0
    for i in range(len(data1)):
        distance += (data1[i] - data2[i])**2
    return distance**0.5

# def Calculate_Distance(data1, data2):  # 余弦相似度
#     mul = 0
#     len1 = 0
#     len2 = 0
#     for i in range(len(data1)):
#         mul += data1[i] * data2[i]
#         len1 += data1[i]**2
#         len2 += data2[i]**2
#     return -mul / ((len1**0.5) * (len2**0.5))


from sortedcontainers import SortedList
train_size = len(train_data.toarray())
test_size = len(test_data.toarray())
test_predict = []  # 存储预测类别
```

```
T1 = time.time()
for i in range(test_size):  # 遍历测试集
    distances = SortedList()  # 有序表

    # 遍历训练集
    for j in range(int(train_size**0.5)):
        distances.add([Calculate_Distance(test_data.toarray()[i],
train_data.toarray()[j]), train_category[j]])
    for j in range(int(train_size**0.5), train_size):
        distances.add([Calculate_Distance(test_data.toarray()[i],
train_data.toarray()[j]), train_category[j]])
        distances.pop(-1)

    # 判断哪个类别最多
    count = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    for k in range(len(distances)):
        count[distances[k][1]] += 1
    test_predict.append(count.index(max(count)))

# 判断正确率
correct = 0
for i in range(test_size):
    if test_category[i] == test_predict[i]:
        correct += 1
print("正确率:", correct / test_size)

T2 = time.time()
print("用时:", T2 - T1)  # 输出用时
```

4. 创新点&优化

1、没有全部计算完再进行排序，而是在计算距离的过程中边计算边排序，同时使用了快表数据结构实现的有序表来对距离进行排序，并且保持有序表的大小不超过 k。相比使用堆保持排序或全部计算完后再统计，使用快表保持排序有更好的效率，大约有 50%左右的速度提升。

2、对数据进行了正则化处理，能够提高 3%左右的正确率。

三、 实验结果及分析

设置	正确率	用时
基准设置（使用欧拉距离和速度优化）	0.297	243.142
对数据正则化处理	0.303	244.778
测试集训练集反转	0.378	398.896
改用余弦相似度计算距离	0.270	213.523
改用曼哈顿距离	0.378	212.863
曼哈顿距离+正则化+测试集训练集反转	0.374	568.278

可以看到，对于此次实验的文本分类数据，在测试了的三种距离中，使用曼哈顿距离的效果最好。相比使用其他两种距离进行计算，曼哈顿距离的正确率有很大提升，同时曼哈顿距离计算简单，其计算用时也最少。

对测试集和训练集进行反转后正确率有很大的提升，主要原因是此次实验的实验数据中，测试集数据比训练集多很多，通常情况下，训练集数据应比测试集数据多，这样能达到更好

的效果，这也从本次实验得到了验证。

最后一行的程序将前面获得了有效提升的设置进行了整合，然而正确率并没有进一步提升，同时用时反而近乎翻倍。这说明了程序的优化并不是简单的叠加，需要根据实际情况选择适当的配置才能达到最好的效果。

四、 参考资料

课件 ppt