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	专业(方向)	学号	姓名
计科三班	计算机科学与技术	21307185	张礼贤

一、实验题目

在给定文本数据集完成文本情感分类训练，在测试集完成测试，计算准确率

二、实验内容

1. 算法原理

- 数据预处理：将文本转换成数值特征向量，常用的方法有词袋模型（Bag-of-Words，BoW）、TF-IDF等，在本次实验中使用的是TF-IDF方法
    - TF：词频=某词出现数（常用的是标准化TF：某词出现数/本文档词语总数）
    - IDF：逆文档频率=log(文档总数/(包含该词的文档数+1)) 【+1避免分母为0】
- 计算距离：使用欧氏距离、曼哈顿距离、余弦相似度等计算未知样本和训练集中每个样本之间的距离或相似度
    - 欧式距离：通过将测试集转化为列表，计算每一行列表项的元素差值平方和再开根号得到两个行向量之间的距离
    - 曼哈顿距离：同上，省去平方操作和开根号操作
    - 余弦相似度：利用余弦求向量的公式求cos值：
        - 余弦值作为衡量两个个体间差异的大小的度量

    - 为正且值越大，表示两个文本差距越小，为负代表差距越大

- 确定K值：选择一个适当的K值，即要考虑到训练集的大小、类别数量、噪声和偏差等因素，避免欠拟合和过拟合，一般根据经验选择k=√N。
- 找到K个最近邻居：根据计算出来的距离或相似度，找到与未知样本最接近的K个训练样本。
- 多数表决法：对于分类问题，采用多数表决法，即选择K个最近邻居中出现最频繁的类别作为未知样本的预测结果，如果出现次数一样则可以根据其位置权值进行判断

2.关键代码及注释

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from timeit import default_timer as timer
import math

label_dict = {"joy": 4,"disgust": 2,"anger": 1 ,"fear": 3,"sad": 5
,"surprise":6}

def distance(x1, x2):
    """计算两个向量的欧氏距离"""
    x1 = np.array(x1)
    x2 = np.array(x2)
    return np.sqrt(np.sum((x1 - x2)**2))

def read_file(filename):
    """读取并分离输入文件中的信息"""
    texts = []  #存储测试文本信息
    labels = [] #存储测试用的标签
    with open(filename, 'r') as f:
        for line in f:
            s = line.strip().split(' ')
            if s[0] == 'documentId':
                continue
            text = s[3]
            for c in s[4:]:
                text += ' ' + c
            label = label_dict[s[2]]
            texts.append(text)
            labels.append(label)
    return texts, labels

def knn_dis(train_set, train_labels, test_set, k):
    n_train = train_set.shape[0]    #获取训练集的行的数量
    n_test = test_set.shape[0]      #获取测试集的行的数量
    dis = np.zeros((n_test, n_train))   #建立全为零的二维数组，存储向量的距
离
    test = test_set.toarray()
    train = train_set.toarray()     #转换成列表，将矩阵square化，便可以计算
其向量距离
```

```
    for i in range(n_test):
        for j in range(n_train):
            dis[i][j] = distance(test[i], train[j])
    pred_labels = np.zeros(n_test, dtype=int)   #存储预测值
    for i in range(n_test):
        indices = np.argsort(dis[i])[:k]    #进行排序，获取前k个距离最短的元素的索引
        k_labels = train_labels[indices]    #根据索引获取label，存储再k_labels中
        pred_label = np.argmax(np.bincount(k_labels))   #调用np库计算众数
        pred_labels[i] = pred_label #将获得的众数预测值赋值给pred_labels的第i个元素
    return pred_labels

def knn_cos(train_set, train_labels, test_set, k=5):
    """k-NN分类算法"""
    n_train = train_set.shape[0]
    n_test = test_set.shape[0]  #获取两个测试集的行的数量
    sim = cosine_similarity(test_set, train_set)    #调用库函数计算余弦相似度
    pred_labels = np.zeros(n_test, dtype=int)   #创建一个一维数组存储结果
    for i in range(n_test):
        indices = np.argsort(sim[i])[-k:]   #反向排序，这点与欧氏距离不相同，这个是相似度越大越好
        k_labels = train_labels[indices]    #将相对应的标签存储到k_labels中
        k_sim = sim[i][indices]     #将相对应的训练集样本存储到k_sim中
        max_count = 0
        pred_label = 0
        for label in set(k_labels):
            count = sum(k_labels == label)  #计算邻居中标签与当前标签相同的个数
            weight = sum(k_sim[k_labels == label])  #计算当前标签的相似度权重
            if count > max_count or (count == max_count and weight > max_weight):
            #如果count大于max_count或者在两者相同的情况下权重weight大于max_weight则更新
                max_count = count
                max_weight = weight
                pred_label = label
        pred_labels[i] = pred_label
    return pred_labels
```

```
def calculate_accuracy(predict, test):
    res = [0.0,0.0,0.0,0.0,0.0,0.0]
    length = [0,0,0,0,0,0]
    count = 0
    for i in range(len(test)):
        if(predict[i] == test[i]):
            res[test[i]-1] += 1.0
            count += 1

        length[test[i]-1] += 1
    for i in range(6):
        res[i] /= length[i]
    return res,count


train_texts, train_labels = read_file('train.txt')  #通过读文件获取测试集和标签集
test_texts, test_labels = read_file('test.txt')

train_labels = np.array(train_labels)   #将标签集合转换为np数组，才能作为参数传入knn函数中

tfidf = TfidfVectorizer()   #创建tfidf类
train_set = tfidf.fit_transform(train_texts)    #前者调用fit_transform函数固定化匹配模板
test_set = tfidf.transform(test_texts)  #后者直接调用transform函数直接套用前面的模板，达到对齐的效果

tic = timer()
pred_labels = knn_cos(train_set, train_labels, test_set, 16)   #调用knn函数，返回预测列表
toc = timer()
accuracy = calculate_accuracy(pred_labels, test_labels)   #统计正确率
for i in range(6):
    print(f'The accuracy of the {i+1}th mood is : {accuracy[0][i]:.4f}')
print(f'The total accuracy: {accuracy[1]/(len(test_labels)):.4f}')
print(f"Your cost_time is {(toc-tic):.4f} seconds!")
```

3.创新点&优化

1. **矩阵列对齐方便计算欧氏距离**
   通过将采集到的数据集转换为array数组，从而实现将原本稀疏存储的矩阵对齐为列数一致的矩阵，便可以利用行向量方便计算

三、实验结果展示及分析

实验结果展示

- 各个情绪的预测准确率：
  - 余弦相似度(取k值为16)

```
The accuracy of the 1th mood is : 0.0000
The accuracy of the 2th mood is : 0.1538
The accuracy of the 3th mood is : 0.1375
The accuracy of the 4th mood is : 0.6602
The accuracy of the 5th mood is : 0.5347
The accuracy of the 6th mood is : 0.0435
The total accuracy: 0.3810
Your cost_time is 0.0520 seconds!
```

  - 欧式距离(取k值为16)

```
The accuracy of the 1th mood is : 0.0152
The accuracy of the 2th mood is : 0.0385
The accuracy of the 3th mood is : 0.2437
The accuracy of the 4th mood is : 0.4613
The accuracy of the 5th mood is : 0.5941
The accuracy of the 6th mood is : 0.0326
The total accuracy: 0.3340
Your cost_time is 3.1320 seconds!
```

可以看到，无论是那种预测方法，对于每一种情绪的预测并不是很均匀，但保证了在第4、5个情绪的预测准确度一般最大

- 调整参数带来的准确率的影响：

k值	准确率(欧式距离)	准确率(余弦拟合度)
1	0.3000	0.2970
2	0.2770	0.2980
3	0.2980	0.3070
4	0.3100	0.3170
5	0.3470	0.3480
6	0.3530	0.3490
7	0.3350	0.3540
8	0.3490	0.3400
9	0.3440	0.3720
10	0.3520	0.3840
11	0.3510	0.3920
12	0.3420	0.3950
13	0.3420	0.4000
14	0.3380	0.3800
15	0.3270	0.3790
16	0.3340	0.3810
17	0.3300	0.3800
18	0.3380	0.3790
19	0.3290	0.3770
20	0.3240	0.3750

k值	准确率(欧式距离)	准确率(余弦拟合度)

k值	搜索时间(欧式距离)	搜索时间(余弦拟合度)
1	3.2392 seconds	0.0228 seconds
2	3.1121 seconds	0.0295 seconds
3	1.9840 seconds	0.0368 seconds
4	1.9618 seconds	0.0406 seconds
5	1.9684 seconds	0.0451 seconds
6	1.9684 seconds	0.0493 seconds
7	1.9632 seconds	0.0523 seconds
8	1.9775 seconds	0.0553 seconds
9	1.9839 seconds	0.0596 seconds
10	1.9835 seconds	0.0645 seconds
11	1.9877 seconds	0.0636 seconds
12	1.9691 seconds	0.0687 seconds
13	1.9824 seconds	0.0698 seconds
14	2.0123 seconds	0.0703 seconds
15	1.9986 seconds	0.0724 seconds
16	1.9702 seconds	0.0743 seconds
17	1.9957 seconds	0.0752 seconds
18	1.9812 seconds	0.0796 seconds
19	2.0059 seconds	0.0801 seconds
20	1.9998 seconds	0.0818 seconds

k值	搜索时间(欧式距离)	搜索时间(余弦拟合度)

实验结果分析

- 对于k值的变化：
  在实验结果展示中展示了k从1到20的情况下使用欧氏距离和余弦相似度进行正确率统计的表格，可以发现，当k靠近根号N时正确率趋于表中的极大值，其具体数值因距离的计算方式不同而不同(欧式为 10；余弦相似度为 13)，因此我们在利用K近邻算法进行文本归类问题求解的时候可以考虑将k赋值成根号N附近的值，可以得到更高的正确率
- 对于搜索时间与正确率的思考：
  - 在使用欧氏距离计算文本相似度时，需要将每篇文本表示成一个向量，然后计算这些向量之间的欧氏距离。然而，在高维空间中，文本向量的维度往往非常高，这会导致计算欧氏距离的时间复杂度非常高，因为需要进行大量的维度计算。此外，欧氏距离对于文本中的稀疏特征不太敏感，这会导致计算结果不够准确，搜索时间花费相对更多
  - 相比欧氏距离，余弦相似度不需要进行大量的维度计算，只需要计算向量之间的夹角余弦即可，因此计算速度更快。此外，余弦相似度对于文本中的稀疏特征更为敏感，因此计算结果更为准确
  - 第二个表展示了两种不同的距离计算方式的搜索时间的情况，可以看到，余弦相似度的计算方式明显更快，平均集中在0.06 secs；而欧氏距离的计算却平均耗费 1.5secs