**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 | **系统结构班** | 专业（方向） | **计算机科学与技术** |
| 学号 | **21307358** | 姓名 | **曾慧蕾** |

# 实验题目

在给定的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。需要对上次给的数据（第七次实验的）进行重新划分，训练集：测试集为8：2.

**要求：**

* **文本的特征可以使用TF或TF-IDF，对TF均使用拉普拉斯平滑技巧（可以使用sklearn库提取特征）**
* **利用k-NN完成对测试集的分类，并计算准确率**
* **需要对参数k对正确率的影响进行探讨。**
* **需要提交简要报告+代码**

# 实验内容

1. 算法原理

**k–近邻（k-NN）算法：**

k-NN是有监督的机器学习模型，是一种常用的分类算法。其核心思想是“近朱者赤，近墨者黑”，即与测试样本最相似的训练样本，其类别很可能与测试样本的类别相同。在分类过程中算法会根据给定的k值寻找与当前分类项最近的k个已知用例，并根据这些已知用例中的情感分类的数量来给待分类项进行分类。

k-NN算法的优缺点如下所示：

**优点：**

1. 简单易懂：KNN算法的原理简单易懂，易于理解和实现。
2. 适用于多分类问题和非线性问题：KNN算法不需要假设数据的分布形式，适用于多分类问题和非线性问题。
3. 鲁棒性强：KNN算法对于噪声数据和异常值的影响较小，具有较强的鲁棒性。

**缺点：**

1. 计算量大：KNN算法需要计算测试样本与所有训练样本之间的距离，计算量随着样本数量的增加而增加，对于大规模数据的处理效果不佳。
2. 存储开销大：KNN算法需要将所有训练样本存储在内存中，对于大规模数据的处理效果不佳。
3. 预测速度慢：KNN算法在预测阶段需要计算测试样本与所有训练样本之间的距离，预测速度较慢。
4. 需要确定K值：KNN算法需要确定K值，K值的选择对于分类结果的影响较大。
5. 对于高维数据的处理效果不佳：KNN算法对于高维数据的处理效果不佳，容易出现维数灾难问题。

k-NN算法步骤如下：

1. 收集训练样本数据和测试样本数据，并将训练样本数据和测试样本数据转换为特征向量。
2. 对于每个测试样本，计算它与所有训练样本之间的距离，并找到与测试样本最相似的k个训练样本。
3. 根据k个最相似的训练样本的类别，来预测测试样本的类别。具体来说，如果k个最相似的训练样本中，大多数属于某个类别，那么测试样本就被归为这个类别。
4. 使用测试样本数据来评估算法的性能，计算分类准确率等指标。
5. 根据评估结果，对算法进行优化，例如调整k值、使用加权距离等。
6. 伪代码

输入：train\_list，test\_list，k

输出：对test\_list的分类结果 prediction

# 计算测试样本与所有训练样本之间的距离

distances = []

for test\_point in test\_list:

    for train\_point in train\_list:

        distance = math.sqrt((test\_point[0] - train\_point[0]) \*\* 2 + (test\_point[1] - train\_point[1]) \*\* 2)

        distances.append((distance, train\_point[2]))

# 找到与测试样本最相似的k个训练样本

k\_nearest\_neighbors = sorted(distances)[:k]

# 根据k个最相似的训练样本的类别，来预测测试样本的类别

labels = [neighbor[1] for neighbor in k\_nearest\_neighbors]

prediction = max(set(labels), key=labels.count)

# 返回预测结果

return prediction

1. 关键代码展示（带注释）

首先是对数据集以8：2的比例进行分类。只需读取全部数据再从中随机取数即可。

sum=1

i=length

while i>0:

    temp=random.randint(0,len(sample)-1)

    if temp in num:

        continue

    else:

        num.append(temp)

        ex=str(sum)+" "+get\_str(sample[temp])

        train\_new.append(ex)

        sum+=1

        i-=1

for i in range(0,len(sample)):

    if i not in num:

        ex=str(sum)+" "+get\_str(sample[i])

        #print(sample[i])

        test\_new.append(ex)

        sum+=1

接下来是knn算法的部分。

首先要对训练集和数据集进行读取

useless\_word="D:\\code\\python\\file\\lab8\\useless.txt"

    file="D:\\code\\python\\file\\lab8\\new\_train.txt"

    useless\_list=read(useless\_word)

    train\_list=read(file)

    train\_list.pop(0)

    emotion=[]

    example=[]

    for i in range(0,len(train\_list)): #分类

        temp\_list=train\_list[i]

        temp\_list=temp\_list[3:]

        sentense=""

        for j in range(0,len(temp\_list)):

            if temp\_list[j] not in useless\_list:

                sentense=sentense+temp\_list[j]+" " #make sentense

        sentense=sentense[:-1]

        example.append(sentense)

        emotion.append(train\_list[i][1]) #一一对应 将单词和情感分开构造后面的独热编码

file="D:\\code\\python\\file\\lab8\\new\_test.txt"

    test\_list=read(file)

    test\_list.pop(0)

    test=[]

    for i in range(0,len(test\_list)): #分类

        temp\_list=test\_list[i]

        temp\_list=temp\_list[3:]

        sentense=""

        for j in range(0,len(temp\_list)):

            if temp\_list[j] not in useless\_list:

                sentense=sentense+temp\_list[j]+" " #make sentense

        sentense=sentense[:-1]

        test.append(sentense)

然后是提取各自的文本特征，并存储在两个矩阵中。这里使用的是TF-IDF值作为文本特征来进行提取

    tv = TfidfVectorizer(use\_idf=False)

    tv\_fit = tv.fit\_transform(example) #频率

    ft\_name = tv.get\_feature\_names\_out() #提取的特征词

arr = tv\_fit.toarray()  #矩阵转列表

    tv1 = TfidfVectorizer(use\_idf=False)

    tv\_fit1 = tv.fit\_transform(test) #频率

    ft\_name1 = tv.get\_feature\_names\_out() #提取的特征词

    arr1 = tv\_fit1.toarray()  #矩阵转列表

接下来是k-NN分类算法，根据k的经验公式，一般取k= N^0.5，N为训练集实例个数，计算出k≈16.以下是代码部分：

def judge(x,arr,emotion,k): #计算测试集中每个用例与训练集中每个用例的距离 选择距离最小的那类感情

    dist=[]

    for i in range(0,len(arr)):

        d=distance(x,arr[i])

        temp=(d,emotion[i])

        dist.append(temp)

    dist=sorted(dist,key=lambda x:x[0])

    x\_emotion=[0,0,0,0,0,0]

    for i in range(0,k): #选择前k个最近的样本并根据情感分类

        num=int(dist[i][1])-1

        x\_emotion[num]+=1

    ans=x\_emotion.index(max(x\_emotion))

    return ans+1

judge函数中涉及到的距离计算函数有：

def distance(a,b): #距离函数

    return ous(a,b)

    # return man\_ha(a,b)

    # return cos(a,b)

def ous(a,b): #两个向量之间的欧式距离

    ans=0

    for i in range(0,len(a)):

        # if (a[i]==0 and b[i]!=0) or (a[i]!=0 and b[i]==0):

        #    ans+=1

        ans+=(a[i]-b[i])\*(a[i]-b[i])

    ans=ans\*\*0.5

    return ans

def man\_ha(a,b): #曼哈顿

    ans=0

    for i in range(0,len(a)):

        if a[i]==b[i]: ans+=0

        else: ans+=1

    return ans

def cos(a,b):

    l1=sum([i\*\*2 for i in a]) \*\*0.5

    l2=sum([i\*\*2 for i in b]) \*\*0.5

    dot\_p=sum([a[i]\*b[i] for i in range(len(a))])

    ans=dot\_p/(l1\*l2)

    return ans

1. 创新点&优化（如果有）

本次实验优化包括有1、对距离度量函数的调整，2、对k值的调整

距离度量函数包括欧氏距离、曼哈顿函数以及向量内积。具体代码已在上面展示。

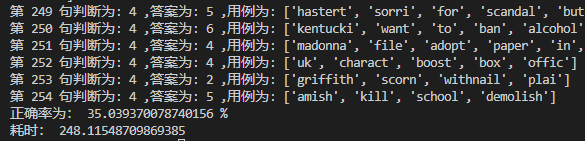
K值调整从16出发分为向上选取和向下选取，找到一个递增的趋势后朝该趋势逐渐逼近，直到出现转折。

具体结果将在第三部分展示。

# 实验结果及分析

1. 实验结果展示示例

当distance函数为ous函数，k值为16时，knn算法正确率如下所示：



2. 评测指标展示及分析

* 对距离度量函数的调整

在这一情况下我们保证k=8

Ous：

Man\_ha：

Cos：

综合时间和正确率来看，Man\_ha的效果要最好

* 对k参数的调整

选用man\_ha函数，从16开始改变k值看正确率，结果如下所示：

|  |  |
| --- | --- |
| K值 | 正确率\% |
| 5 | 22.83464 |
| 6 | 17.32283 |
| 7 | 36.61417 |
| 8 | 37.00787 |
| 10 | 35.82677 |
| 13、16、20、25、30…… | 35.03937 |

**综上来看，当k=8时，knn算法的正确率最高，因此选用k=8**

# 参考资料

[算法原理参考：机器学习——KNN算法](https://blog.csdn.net/weixin_45936137/article/details/124624405)

超算习堂-课时19-第十一讲 实验课

[停用表](https://github.com/sansenlian/stop_words/blob/master/english.txt)