**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

用朴素贝叶斯法完成文本信息情感分类训练，要求使用拉普拉斯平滑技巧。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，并计算准确率。

**要求：**

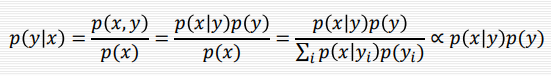
1. **文本的特征可以使用TF或TF-IDF，均使用拉普拉斯平滑技巧（本次实验可以使用sklearn库提取特征）**
2. **利用朴素贝叶斯完成对测试集的分类，并计算准确率**
3. **需要提交简要报告+源代码**

# 实验内容

1. 算法原理

**朴素贝叶斯法：**

做法是根据贝叶斯定理来估计每个类别的后验概率。依据公式为：



然后，对每个情感的后验概率进行对比，选择概率最大的一个情感作为判断结果。

**TF/TF-IDF：**

TF和TF-IDF是文本特征提取中常用的两种算法，它们的原理如下：

* TF是指某个词在文本中出现的频率，即该词在文本中出现的次数除以文本中总词数。TF算法的原理是，一个词在文本中出现的次数越多，它对文本的区分度就越高。
* TF-IDF算法是在TF算法的基础上进一步优化的算法。TF-IDF算法的原理是，一个词在文本中出现的次数越多，它对文本的区分度就越高，但是如果这个词在所有文本中都出现的次数很多，那么它对文本的区分度就会降低。因此，TF-IDF算法不仅考虑了一个词在一个文本中出现的频率，还考虑了这个词在所有文本中出现的频率。具体来说，TF-IDF算法是将一个词的TF值乘以它的IDF值得到的，其中IDF是指一个词在所有文本中出现的次数的倒数，即IDF=log(总文档数/包含该词的文档数)。这样，一个词在某个文本中出现的次数越多，它的TF-IDF值就越高，表示它对该文本的区分度越高。

**拉普拉斯平滑：**

拉普拉斯平滑用于解决在文本分类中出现概率为0的问题。在文本分类中，当某个词在某个类别中没有出现时，它的概率就会变成0，这会导致整个文本的概率为0，从而影响分类的准确性。拉普拉斯平滑技巧可解决这一问题。

拉普拉斯平滑技巧的原理是，在计算概率时，将每个词出现的次数加上一个常数k，再将分母加上k乘以词汇表的大小。这样，即使某个词在某个类别中没有出现，它的概率也不会变成0，而是会有一个较小的概率。



式中𝜆 ≥ 0。等价于在随机变量各个取值的频数上赋予一个正数𝜆 ≥ 0。当𝜆 = 0 时就是极大似然估计。当𝜆 = 1时称为拉普拉斯平滑。

1. 伪代码

用TF-IDF算法求文本特征矩阵：

输入：文本集document

输出：得到的TF-IDF矩阵

vocabulary = set()

for document in documents:

    words = document.split()

    for word in words:

        vocabulary.add(word)

# 计算TF-IDF值

tfidf\_matrix = []

for document in documents:

    tfidf\_vector = []

    words = document.split()

    for word in vocabulary:

        # 计算TF值

        tf = words.count(word) / len(words)

        # 计算IDF值

        idf = math.log(len(documents) / (sum([1 for d in documents if word in d])))

        # 计算TF-IDF值

        tfidf = tf \* idf

        tfidf\_vector.append(tfidf)

tfidf\_matrix.append(tfidf\_vector)

return tfidf\_matrix

对文本特征矩阵进行拉普拉斯平滑处理

输入：代处理的矩阵res，平滑系数k

输出：经过平滑处理后的矩阵

smoothed\_matrix = []

for row in matrix:

    smoothed\_row = []

    row\_sum = sum(row)

    for value in row:

        smoothed\_value = (value + k) / (row\_sum + k \* len(row))

        smoothed\_row.append(smoothed\_value)

    smoothed\_matrix.append(smoothed\_row)

return smoothed\_matrix

朴素贝叶斯文本情感分类算法：

输入：待测文本集

输出：对文本集中每个样例的情感判断结果

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

        temp=test\_list[i]

        answer=int(temp[1])

        ss=temp[3:]

        max=-100000

        judgement=-1

        for j in range(6):

            res=judge(j,arr,ss,word)

            if res>max:

                max=res

                judgement=j+1

#最后的judgement即为判断结果 以数字代表情感。

#计算每种情感下的概率

def judge(num,arr,ss,word):

    res=0

    for i in range(0,len(arr[num])):

        mul=1

        for j in range(0,len(ss)):

            if ss[j] in word[num]:

                index=find(word[num],ss[j])

                mul\*=arr[num][i][index] #后验概率公式

        res+=mul

return res

1. 关键代码展示（带注释）

首先是对文本的特征处理函数。由于本次任务允许使用sklearn，因此主要用TfidfVectorizer来处理。

文本特征提取：

def classify(set):

    tv=TfidfVectorizer(use\_idf=True)

    tv\_fit=tv.fit\_transform(set)

    ft\_name = tv.get\_feature\_names\_out()

    res = tv\_fit.toarray()

    res=lpls(res,0.0001)

    res\_norm = np.array(res) / np.sum(res,axis=1,keepdims=True)

    res\_norm=extend(res\_norm)

    arr.append(res\_norm)

    word.append(ft\_name)

由于特征提取得到的矩阵中每个元素都过小，影响判断结果，因此对每个值进行放大：

def extend(res):

    for i in range(0,len(res)):

        for j in range(0,len(res[0])):

            res[i][j]\*=10000000000

    return res

拉普拉斯平滑处理：

def lpls(res,k):

    sum=0

    for i in range(0,len(res)):

        sum=0 #每到一个新的行都要更新

        for j in range(0,len(res[i])):

            sum+=res[i][j]

        for j in range(len(res[i])):

            res[i][j]=(res[i][j]+k)/(sum+len(res[0])\*k)

    return res

查找np矩阵中的下标：

def find(name,index):

    for i in range(0,len(name)):

        if name[i]==index:

            return i

贝叶斯概率计算：

def judge(num,arr,ss,word):

    res=0

    for i in range(0,len(arr[num])):

        mul=1

        for j in range(0,len(ss)):

            if ss[j] in word[num]:

                index=find(word[num],ss[j])

                mul\*=arr[num][i][index] #后验概率公式

        res+=mul

    return res

训练文本集的分类：

    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]

        if train\_list[i][1]=='1':

            anger\_.append(sentense)

        elif train\_list[i][1]=='2':

            disgust\_.append(sentense)

        elif train\_list[i][1]=='3':

            fear\_.append(sentense)

        elif train\_list[i][1]=='4':

            joy\_.append(sentense)

        elif train\_list[i][1]=='5':

            sad\_.append(sentense)

        elif train\_list[i][1]=='6':

            surprise\_.append(sentense)

    classify(anger\_)

    classify(disgust\_)

    classify(fear\_)

    classify(joy\_)

    classify(sad\_)

    classify(surprise\_)

测试文本集的判断：

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

        temp=test\_list[i]

        answer=int(temp[1])

        ss=temp[3:]

        max=-100000

        judgement=-1

        for j in range(6):

            res=judge(j,arr,ss,word)

            if res>max:

                max=res

                judgement=j+1

        if judgement==answer:

            right+=1

1. 创新点&优化（如果有）

原本的训练集中的词汇数量非常巨大，在简要检查这些词汇过后发现里面有一些词语出现次数非常少（1~2次）并且并无实际含义，如di、s或者一些介词。跟同学讨论过后认为在读取文本时可以忽略这一部分词汇，一来可以减少文本特征的提取，二来可以减少微量词汇对总体判断的影响。代码如下：

useless\_word="D:\\code\\python\\file\\lab7\\useless.txt"

useless\_list=read(useless\_word)

#读取时

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

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

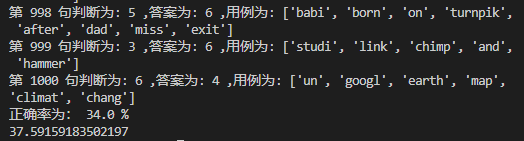
                sentense=sentense+temp\_list[j]+" " #make sentense

        sentense=sentense[:-1]

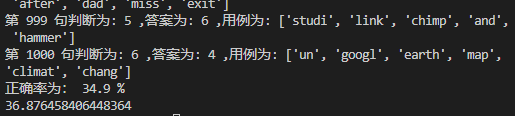
# 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

在未使用停用表时：



在使用停用表后：



2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

在增设停用表后，可见正确率有所提高（虽然不多），运行时间有所减少（虽然不多）。而且在多次重复执行之后二者的差距与上述结果无甚差距，可见增设停用表确实可以提高算法效率。

# 参考资料

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

**超算习堂-课时18-第十讲实验课**

**超算习堂-课时16-第八讲**

[TF-IDF算法讲解\_饿了就干饭的博客-CSDN博客](https://blog.csdn.net/qq_45893319/article/details/119278730)