**信 息 内 容 安 全 实 验 报 告**

|  |  |
| --- | --- |
| 实验项目名称： | 贝叶斯公式进行垃圾邮件过滤 |
| 班级： | SC011701 |
| 学号： | 2017302238  2017302239  2017302627  2017302250 |
| 姓名： | 郑启睿（编程）  赵显艺（理论）  高杰（报告）  刘昊苏（报告） |
| 指导教师： | 杨黎斌 |
| 实验时间： | 2020.3.28 |

**摘要**

垃圾邮件过滤是现代人生活的必要需求，但是正确识别垃圾邮件的技术难度非常大，传统的垃圾邮件过滤方法不仅效果不理想，而且容易规避。本实验基于朴素贝叶斯公式，通过拉普拉斯平滑处理和5-折交叉验证达到了较为理想的垃圾邮件过滤效果。

**目录**

一、实验目的·······················2

二、实验要求·······················2

三、背景阐述·······················2

四、相关文献·······················3

五、实验理论模型·····················3

六、实验系统设计思路···················5

七、实验结果及分析····················6

**贝叶斯公式进行垃圾邮件过滤**

**一、实验目的**

1、了解贝叶斯公式的相关知识

2、学习运用贝叶斯公式进行垃圾邮件筛选

3、考虑贝叶斯公式在除了垃圾邮件过滤之外的推广

**二、实验要求**

1.实现基于朴素贝叶斯的垃圾邮件过滤

2.掌握朴素贝叶斯算法基本原理和基于概率的文本分类

**三、背景阐述**

1、正确识别垃圾邮件的技术难度非常大。传统的垃圾邮件过滤方法，主要有"关键词法"和"校验码法"等。它们的识别效果都不理想，而且很容易规避。

2、“关键词法”：过滤依据是特定的词语；

3、“校验码法”：是计算邮件文本的校验码，再与已知的垃圾邮件进行对比。

2002年，保罗·格雷厄姆（Paul Graham)提出使用“贝叶斯推断”过滤垃圾邮件。1000封垃圾邮件可以过滤掉995封，且没有一个误判。这种过滤器还具有自我学习的功能，会根据新收到的邮件，不断调整。收到的垃圾邮件越多，其准确率就越高。

**四、相关文献**

[1]李益清.贝叶斯公式的应用[J].科教文汇(上旬刊),2019(11):75-76.

[2]巴林.全概率公式与贝叶斯公式的应用[J].课程教育研究,2018(48):120.

**五、实验理论概述**

1、条件概率

所谓"条件概率"（Conditional probability），就是指在事件B发生的情况下，事件A发生的概率，用P(A|B)来表示。

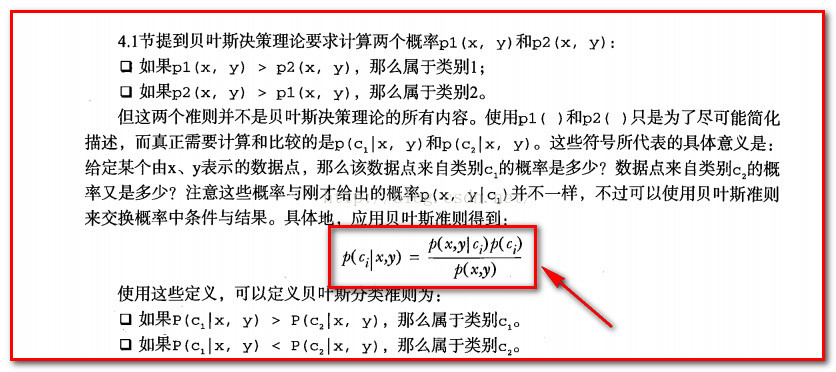
2、先验概率：P(A)这就是先验概率，即在B发生前我们对A事件概率的一个判断。

3、后验概率：P(A|B)=P(A)P(B|A)/P(B)，这就是后验概率。

P(B|A)/P(B)称为可能性函数，调整因子。

后验概率=先验概率\*调整因子

依据条件概率进行文本分类：



4、准备数据：从文本中构建词向量

5、训练

训练算法：从词向量计算概率

预先提供两组已经识别好的邮件，一组是正常邮件，另一组是垃圾邮件。

训练过程：首先解析所有邮件进行分词。然后计算每个词语在正常邮件和垃圾邮件中的出现频率。

6、拉普拉斯平滑处理：

零概率问题，就是在计算实例的概率时，如果某个量x，在观察样本库（训练集）中没有出现过，会导致整个实例的概率结果是0。在文本分类的问题中，当一个词语没有在训练样本中出现，该词语调概率为0，使用连乘计算文本出现概率时也为0。这是不合理的，不能因为一个事件没有观察到就武断的认为该事件的概率是0。

拉普拉斯的理论支撑：

　　为了解决零概率的问题，法国数学家拉普拉斯最早提出用加1的方法估计没有出现过的现象的概率，所以加法平滑也叫做拉普拉斯平滑。

　　假定训练样本很大时，每个分量x的计数加1造成的估计概率变化可以忽略不计，但可以方便有效的避免零概率问题。

应用举例

　　假设在文本分类中，有3个类，C1、C2、C3，在指定的训练样本中，某个词语K1，在各个类中观测计数分别为0，990，10，K1的概率为0，0.99，0.01，对这三个量使用拉普拉斯平滑的计算方法如下：

　　1/1003 = 0.001，991/1003=0.988，11/1003=0.011

在实际的使用中也经常使用加 lambda（1≥lambda≥0）来代替简单加1。如果对N个计数都加上lambda，这时分母也要记得加上N\*lambda。

7、交叉验证

（1）将原始数据随机分为两组，一组作为训练集，一组作为验证集，利用训练集训练分类器，然后利用验证集验证模型，记录最后的分类准确率为此分类器的性能指标。

（2）K-折交叉验证（K-fold Cross Validation，记为K-CV）将原始数据分成K组（一般是均分），将每个子集数据分别做一次验证集，其余的K-1组子集数据作为训练集，这样会得到K个模型，用这K个模型最终的验证集的分类准确率的平均数作为此K-CV下分类器的性能指标。K一般大于等于2，实际操作时一般从3开始取，只有在原始数据集合数据量小的时候才会尝试取2。

**六、实验系统设计思路**



数据来源：50条邮件信息，25个正常邮件ham，25个垃圾邮件spam

数据准备

数据需要拆分成训练集和测试集。

拆分训练集和测试集的思路如下：

1.遍历包含50条数据的email文件夹，获取文件列表

2.取spam的文件作为测试集。

朴素贝叶斯实现的一般过程：

1.收集数据：提供文本文件email。

2.准备数据：将文本文件解析成词条向量。

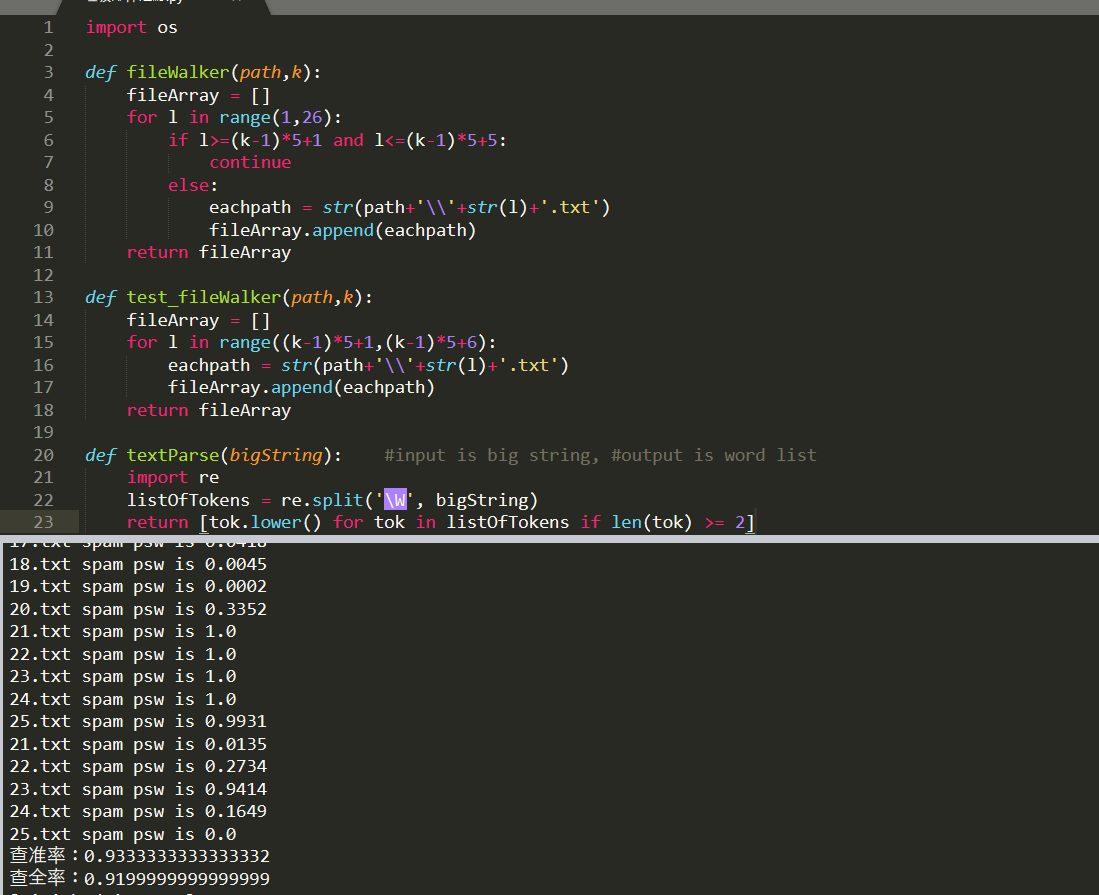
3.分析数据：检查词条确保解析的正确性。

4.训练算法：计算不同的独立特征的条件概率。

5.测试算法：构建一个新的测试函数来计算文档集错误率。

6.使用算法：邮件过滤

**七、实验结果及代码分析**



结果分析：可见正确率为93.3%，实际上严格来说，应当将取多组数据，每一组轮流作为一次测试集，再将多次计算结果求均值，这个模型求出的分类效果才具有可靠性，其次，数据量小导致准确率较小的原因不排除在外。

代码分析：定义函数fileWalker、test\_fileWalker。然后使用textParse分词函数进行分词，之后用email\_parser、get\_word、count\_word\_prob对读取的邮件进行分词、预处理，之后用filter进行邮件分析，然后就是main函数最终实现贝叶斯公式进行垃圾邮件过滤。函数具体原理见实验理论概述。

**附：源代码**

import os

#控制邮件读取第1-第25封邮件

def fileWalker(path,k):

fileArray = []

for l in range(1,26):

if l>=(k-1)\*5+1 and l<=(k-1)\*5+5:

continue

else:

eachpath = str(path+'\\'+str(l)+'.txt')

fileArray.append(eachpath)

return fileArray

def test\_fileWalker(path,k):

fileArray = []

for l in range((k-1)\*5+1,(k-1)\*5+6):

eachpath = str(path+'\\'+str(l)+'.txt')

fileArray.append(eachpath)

return fileArray

#分词函数

def textParse(bigString): #input is big string, #output is word list

import re

listOfTokens = re.split('\W', bigString)

return [tok.lower() for tok in listOfTokens if len(tok) >= 2]

#对读取的邮件进行分词、预处理

def email\_parser(email\_path):

clean\_word = textParse(open(email\_path, encoding='ISO-8859-1').read())

return clean\_word

def get\_word(email\_file,k):

word\_list = []

word\_set = []

email\_paths = fileWalker(email\_file,k) #遍历email\_fil目录，得到所有文件的地址；email\_paths：地址数组

for email\_path in email\_paths:

clean\_word = email\_parser(email\_path)

word\_list.append(clean\_word)#词的列表的列表

word\_set.extend(clean\_word)#词的集合

return word\_list, set(word\_set)

def count\_word\_prob(email\_list, union\_set): #词频向量

word\_prob = {}

for word in union\_set:

counter = 0

for email in email\_list:

if word in email:

counter += 1

else:

continue

prob = (counter+1)/len(email\_list) #拉普拉斯平滑处理

word\_prob[word] = prob #word\_prob[]

return word\_prob

##############################################################

def filter(ham\_word\_pro, spam\_word\_pro, test\_file, k):

test\_paths = test\_fileWalker(test\_file,k)#拿到测试集

T=0

F=0 #T 预测为垃圾邮件的个数， F 预测为正常邮件的个数

for test\_path in test\_paths:

email\_spam\_prob = 0.0 #P(W|S)

spam\_prob = 0.5 #P(S)

ham\_prob = 0.5 #P(W)

file\_name = test\_path.split('\\')[-1]

prob\_dict = {}

words = set(email\_parser(test\_path))

for word in words:

Psw = 0.0

if word not in spam\_word\_pro:

Psw = 0.4

else:

Pws = spam\_word\_pro[word]

Pwh = ham\_word\_pro[word]

Psw = spam\_prob\*(Pws/(Pwh\*ham\_prob+Pws\*spam\_prob))

prob\_dict[word] = Psw

numerator = 1

denominator\_h = 1

L = sorted(prob\_dict.items(), key=lambda e:e[1], reverse=True)

L = L[:15]

for l in L:

numerator \*= l[1]

denominator\_h \*= (1-l[1])

email\_spam\_prob = round(numerator/(numerator+denominator\_h), 4)

if email\_spam\_prob > 0.95: #如果概率大于0.95认为是垃圾邮件

T+=1

print(file\_name, 'spam', 'psw is',email\_spam\_prob)

else:

F+=1

print(file\_name, 'spam', 'psw is',email\_spam\_prob)

return T,F

def main():

ham\_file = r'C:\Users\gaojie\Desktop\email\ham'

spam\_file = r'C:\Users\gaojie\Desktop\email\spam'

chazhun=0.0 #预测为垃圾邮件的有多少是真的垃圾邮件

chaquan=0.0 #真正的垃圾邮件有多少被识别出来了

for k in range(1,6): #25个垃圾邮件、正常邮件，每次选择(k-1)\*5+1 --- (k-1)\*5+5作为测试集，其他的作为训练集，进行5折交叉验证，求出查准率、查全率取平均值

ham\_list, ham\_set = get\_word(ham\_file,k)

spam\_list, spam\_set = get\_word(spam\_file,k)

union\_set = ham\_set | spam\_set#建立词向量

ham\_word\_pro = count\_word\_prob(ham\_list, union\_set)

spam\_word\_pro = count\_word\_prob(spam\_list, union\_set)

TP,FN=filter(ham\_word\_pro, spam\_word\_pro, spam\_file, k)

FP,TN=filter(ham\_word\_pro, spam\_word\_pro, ham\_file, k)

chazhun+=TP/(TP+FP)

chaquan+=TP/(TP+FN)

print("查准率："+str(chazhun/5))

print("\n")

print("查全率："+str(chaquan/5))

if \_\_name\_\_ == '\_\_main\_\_':

main()