**信 息 内 容 安 全 实 验 报 告**

|  |  |
| --- | --- |
| 实验项目名称： | 垃圾邮件过滤系统 |
| 班级： | SC011701 |
| 学号： | 2017302207 |
| 姓名： | 高丽 |
| 指导教师： | 杨黎斌 |
| 实验时间： | 2020.4.1 |

**目 录**

系统概述 1

一 原理以及其在邮件检测中的应用 1

1.1 TF-IDF关键词选择 1

1.2 朴素贝叶斯 2

1.2.1 条件概率、全概率公式、朴素贝叶斯定义 2

1.2.2 贝叶斯联合概率 3

1.2.3 极大似然估计、贝叶斯估计 4

1.2.4 拉普拉斯平滑处理 5

二 三个模型 6

2.1 多项式模型 6

2.2 伯努利模型 6

2.3 高斯模型 6

三 实现和代码分析 7

3.1 总体流程 7

3.2 TF-IDF算法实现 7

3.3 基于联合概率公式和阈值判断的伯努利模型 8

3.4 基于贝叶斯公式展开和极大似然估计、贝叶斯估计的多项式模型 10

四 实验结果分析 12

4.1 两个模型的比较 12

4.2 是否使用TF-IDF的比较 13

4.3 高误判文件spam6和spam17分析 14

# 拉锡区月第第秒er 系统概述

邮件总共50封，25封spam、25封ham，随机选择20%的ham和spam邮件共10个作为测试集，其余40封作为训练集。利用机器学习算法对训练集训练，并对测试集进行垃圾邮件过滤：

（1）特征提取部分，利用TF-IDF算法进行关键词选取。

（2）实现多项式模型和基于阈值判断、联合概率的贝叶斯分类器。

（3）实现伯努利模型和贝叶斯估计的朴素贝叶斯分类器.

# 一 原理以及其在邮件检测中的应用

## 1.1 TF-IDF关键词提取

一般来说，一篇文章的关键词是文章的高频词，但是有些词汇是普遍常用的，尽管词频较高，却无法作为文章的主旨词汇。所以我们引入IF-IDF对这一类词进行过滤：

如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，这里正是我们所需要的关键词。

首先统计指定文档的词频（Term Frequency，缩写为TF）。

考虑到如果某个词比较少见，但是它在这篇文章中多次出现，那么其重要性会偏高。

在词频的基础上，要对每个词分配一个“重要性”权重。较常见的词给予较小的权重，较少见的词给予较大的权重。这个权重叫做"逆文档频率"（Inverse Document Frequency，缩写为IDF），它的大小与一个词的常见程度成反比。

某个词对文章的重要性越高，它的TF-IDF值就越大。所以，排在最前面的几个词，一般就是这篇文章的关键词。

在邮件检测系统中，语料库即测试集。使用TF-IDF计算一篇邮件的所有单词的TF-IDF值，按输入设定的n，选取其最关键的n个关键词。

## 1.2 朴素贝叶斯

**1.2.1条件概率、全概率公式、朴素贝叶斯定义**

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。

称为先验概率：B发生前，我们对A事件的一个判断；为后验概率；B发生后，我们对A事件概率重新进行评定；剩下的项为调整因子。所以条件概率可以写成：

垃圾邮件检测中，我们利用：

（记为式-1）来获得已知出现某个单词的情况下，该邮件为垃圾邮件Spam的概率。这里S是垃圾邮件事件，W代表单词。

全概率公式:

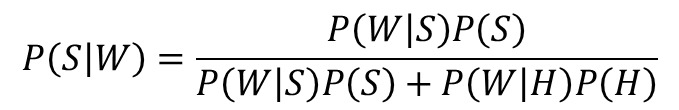
如果事件、… 构成一个完备事件组，即它们两两互不相容，其和为全集；并且P（)大于0，则对任一事件A有

。

或者：，其中A与Bn的关系为交

含义：划分中的事件、是事件A发生的全部“原因”，那么每个“原因”发生的概率与该“原因”导致A发生的概率乘积即为A发生的概率。

则上述公式-1可以改写为



记为式-2，这里H是非垃圾邮件事件。

朴素贝叶斯定义：

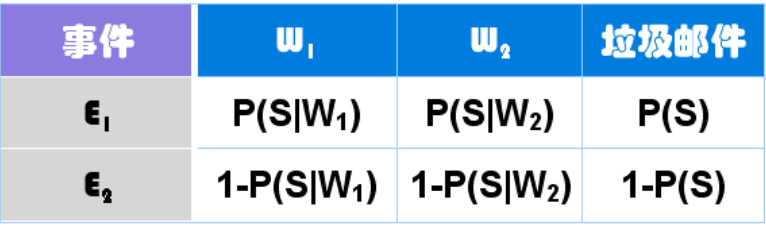
我们需考察，多个单词对判断垃圾邮件的影响，故式-2改为：

为简化问题，需要进行朴素而大胆的假设： 特征项之间相互独立，由此 n 维的联合概率可简化成 n 个分立的概率乘积，这将问题降到易于计算多项式级别。分子分母均为先验概率乘积，统计计算都较为容易。

**1.2.2贝叶斯联合概率**

所谓联合概率，就是指在多个事件发生的情况下，另一个事件发生概率有多大。比如，已知和是两个不同的词语，它们都出现在某封电子邮件之中，那么这封邮件是垃圾邮件的概率，就是联合概率。

在已知和的情况下，无非就是两种结果：垃圾邮件（事件）或正常邮件（事件）。



如果假定所有事件都是独立事件（严格地说，这个假定不成立，但是这里可以忽略），那么就可以计算P()和P()：

由于在和已经发生的情况下，垃圾邮件的概率为：

代入可知：

P(S|W1)记为P1，P(S|W2)记为P2：

扩展得：

先验概率在不同模型下有不同的计算方式。

有时候会简单认为，不过据社会统计，一般一封新邮件是垃圾邮件的概率高达0.8。

**1.2.3极大似然估计、贝叶斯估计**

统计学上分频率学派和贝叶斯学派，那么自然的，对这个问题的解决就有两种模型，一种是频率学派推崇的极大似然估计，一种是贝叶斯学派主张的贝叶斯估计，接下来介绍这邮件检测中的应用：

①极大似然估计

朴素贝叶斯算法中，学习即估计和。

也就是前文的和。

先验概率，即类别可以为S或者H，k=1，2。这里先验概率的极大似然估计就是:

N为样本数量。

条件概率极大似然估计为：

j为我们所取的关键词个数，也就是第j个特征，为它所取的值。I是指示函数，在我的系统中，即为计数。

②贝叶斯估计

极大似然估计可能会出现所要估计的概率值为0的情况，这时会影响到后验概率的计算结果，使分类产生偏差，解决方法是使用贝叶斯估计。条件概率的贝叶斯估计为：

为所有的个数，也就是取值总个数。

就是极大似然估计。

先验概率的贝叶斯估计为：

K就是Y所有可能的类别数量，垃圾邮件检测中值为2。

**1.2.4 拉普拉斯平滑处理**

贝叶斯估计中，常取值为1，这就是拉普拉斯平滑,也就是“拉普拉斯修正”。

实际上是为了避免其他属性携带的信息被训练集中未出现的属性值“抹去”，在估计概率值时使用“平滑”。

# 二 三个模型

三个模型均是给出了先验概率和类条件概率的计算方法：

## 2.1 多项式模型

多项式模型以单词为粒度。

先验概率

类条件概率=

## 2.2 伯努利模型

伯努利模型以文件为粒度。

先验概率

类条件概率=

注意：这里+1、+2是使用了拉普拉斯平滑，防止新出现的单词被忽略。

## 2.3 高斯模型

当特征是连续变量的时候，运用多项式模型就会导致很多（不做平滑的情况下），此时即使做平滑，所得到的条件概率也难以描述真实情况。所以处理连续的特征变量，应该采用高斯模型。

有些特征可能是连续型变量，比如说人的身高，物体的长度，这些特征可以转换成离散型的值，比如如果身高在160cm以下，特征值为1；在160cm和170cm之间，特征值为2；在170cm之上，特征值为3。也可以这样转换，将身高转换为3个特征，分别是f1、f2、f3，如果身高是160cm以下，这三个特征的值分别是1、0、0，若身高在170cm之上，这三个特征的值分别是0、0、1。

不过这些方式都不够细腻，高斯模型可以解决这个问题。高斯模型假设这些一个特征的所有属于某个类别的观测值符合高斯分布。

本垃圾邮件检测不适用高斯模型，故不作建模阐述。

# 三 实现和代码分析

## 3.1 总体流程

（1）文件转换编码全部统一为utf-8；（convertformat.py）

（2）随机选择20%的ham和spam放入test文件夹中，剩余放入train文件夹中，然后预处理：（init\_sets.py）

①读取train每个文件，分词、去停用词、去标点符号;（wordfreq.py）

②并计算TF-IDF值，取前n个值较大的词作为关键词；（tfidf.py中）

（3）利用多项式模型和伯努利模型训练；（bayes.py）

（此次垃圾检测的具体的算法已在原理一栏的应用部分给出，这里不作阐述）

（4）训练好的模型去检测test下的文件，并计算正确率；（bayes.py main.py/main2.py）

（5）以上过程运行50次，每次计算机的正确率作平均值，得出平均正确率，并比较两个模型的平均正确率以及使用TF-IDF与否的正确率。（main2.py）

（6）分析误判文件。

说明：

以上所有除了分词和去停用词以外，所有内容均为手动实现。

main.py利用了一个简单的贝叶斯分类器，并且使用联合概率分布，使用阈值判断，未作平滑处理，模型为伯努利模型，这个是一个初版本；

main2.py使用联合概率分布、阈值判断的伯努利模型以及最大似然估计的多项式模型。均作了平滑处理。

## 3.2 TF-IDF算法实现

使用nltk库进行分词。corpus.idf和corpus.tf帮助计算TF-IDF。

外部使用直接调用：

getallwordtfidf\_text(info\_list,contents\_list)：计算tfidf；

get\_maxtfidf\_word(info\_list,num\_maxtfidfwords)：获得几个最大tfidf值；

getallword(contents\_list,repeat=False)：获得单词。

from nltk.text import TextCollection

import nltk

def getallwordtfidf\_text(info\_list,contents\_list):#第一个是集合的信息表，contents\_list是所有文章转化为句子的总列表

    for key ,val in info\_list.items():

        #每一篇

        content=val['content']

        info\_list[key]['tfidf']=computeTFIDF\_text(texts=contents\_list,singletext=content)

    return info\_list

def computeTFIDF\_text(texts,singletext):#texts是句子字符串列表（语料库）,singletext单个句子字符串,

    texts=[nltk.word\_tokenize(text) for text in texts]#对句子列表所有句子分词

    corpus = TextCollection(texts)

    words=nltk.word\_tokenize(singletext) #单词列表

    tfidf\_words={}

    #计算机tfidf

    for word in words:

        idf=corpus.idf(word)#tf

        tf=corpus.tf(word, words)#idf

        tfidf=idf\*tf

        tfidf\_words[word]=tfidf

    return tfidf\_words

def get\_maxtfidf\_word(info\_list,num\_maxtfidfwords):#获得几个最大tfidf值的单词

    for key  in info\_list.keys():

        #每一篇

        d= info\_list[key]['tfidf']

        L=list(d.items())       # 得到列表

        L.sort(key=lambda x:x[1],reverse=True)  #

        if num\_maxtfidfwords>len(L):

            info\_list[key]['maxtfidf']=L

        else:

            info\_list[key]['maxtfidf']=L[:num\_maxtfidfwords]

        for i in L[num\_maxtfidfwords:]:

            info\_list[key]['wordslist'].remove(i[0])#删除低tfidf 的概率

    return info\_list

#h获得所有单词

def getallword(contents\_list,repeat=False):#repeat允许重复

    text=' '.join(contents\_list)

    allwords1=nltk.word\_tokenize(text)#对句子列表所有句子分词

    #去重

    if repeat:

        return allwords1

    else:

        allwords=[]

        for i in allwords1:

            if not i in allwords:

                allwords.append(i)

        return allwords

## 3.3 基于联合概率和阈值判断的伯努利模型

①初版本：

参数说明：

Threshold为可以设定的阈值，一般设为0.8/0.9；

usetfidf为bool变量，表示是否使用tfidf，如果使用，则只用关键词进行判断，否则，使用全部分词后的结果中的词汇。

这个算法中，第一次出现的词P（S|W）设为0.4；未做平滑处理。

里面调用的函数均在我写的其他文件中，由于并不关键，这里就不粘贴了。

def naivebayes(file,filename,wordfreq\_intrainham\_dict,wordfreq\_intrainspam\_dict,Threshold,usetfidf=True):

    Ps=0.5#先验概率

    Ph=0.5

    words=[]

    if usetfidf==True:#使用tfidf过滤

        words=file['wordslist']

    else:#不使用tfidf

        words,sent=wf.WordsListofFile(filename=filename,dirpath=thispath)

    Ps\_w\_dict={}

    countnewword=0

    newword=[]

    for word in words:#

        #第一次出现

        if word not in wordfreq\_intrainspam\_dict.keys():

            Ps\_w=0.4

            countnewword+=1

            newword.append(word)

            #原先的spam中没有这个单词

            # Update(filename,info\_train\_hamlist,allwordslist,sumofhamtrain)

        elif word not in wordfreq\_intrainham\_dict.keys():

            Ps\_w=0.4

            countnewword+=1

            newword.append(word)

        #其他词

        else:

            Pws=wordfreq\_intrainspam\_dict[word]

            Pwh=wordfreq\_intrainham\_dict[word]

            Pw=(Pws\*Ps)+(Pwh\*Ph)

            Ps\_w=(Pws\*Ps)/(Pw)

        Ps\_w\_dict[word]=Ps\_w#添加

    son =1

    mother=1

    motherpart1=son

    motherpart2=1

    # print(Ps\_w\_dict)

    #联合概率

    for word in Ps\_w\_dict.keys():#朴素贝叶斯  相互独立

        son\*=Ps\_w\_dict[word]

        motherpart1=son

        motherpart2\*=(1-Ps\_w\_dict[word])

        mother=motherpart1+motherpart2

    p=son/mother

    if p>Threshold:

        return False,Ps\_w\_dict,p#垃圾邮件

    else:

        return True,Ps\_w\_dict,p

②添加平滑处理的版本

由于当时计算词频单独写了一个函数，所以加入平滑处理后无法使用原先的版本，故另写了一个版本

#作了平滑处理的伯努利模型  联合概率分布 阈值判断

def bernoulli(file,filename,info\_train\_spamlist,info\_train\_hamlist,allwordslist,Threshold,usetfidf=True):

    Ps=0.5#先验概率

    Ph=0.5

    words=[]

    if usetfidf==True:#使用tfidf过滤

        words=file['wordslist']

    else:#不使用tfidf

        words,sent=wf.WordsListofFile(filename=filename,dirpath=thispath)

    Ps\_w\_dict={}

    for word in words:#  #每一个单词

        #第一次出现

        counts=0

        counth=0

        for each in info\_train\_spamlist.keys():#每一篇

            if(wf.isWordexistsinFile(info\_train\_spamlist,each,word)):

                #如果存在

                counts+=1

        counts+=1

        for each in info\_train\_hamlist.keys():#每一篇

            if (wf.isWordexistsinFile(info\_train\_hamlist,each,word)):

                counth+=1

        counth+=1#拉普拉斯平滑

        Pws = counts/(config.sumofspamtrain+2)#仅此处不同

        Pwh = counth/(config.sumofhamtrain+2)

        Pw=(Pws\*Ps)+(Pwh\*Ph)

        Ps\_w=(Pws\*Ps)/(Pw)

        Ps\_w\_dict[word]=Ps\_w#添加

    son =1

    mother=1

    motherpart1=son

    motherpart2=1

    #联合概率

    for word in Ps\_w\_dict.keys():#朴素贝叶斯

        son\*=Ps\_w\_dict[word]

        motherpart1=son

        motherpart2\*=(1-Ps\_w\_dict[word])

        mother=motherpart1+motherpart2

    p=son/mother

    if p>Threshold:

        return False,Ps\_w\_dict,p#垃圾邮件

    else:

        return True,Ps\_w\_dict,p

## 3.4 基于贝叶斯公式展开和极大似然估计、贝叶斯估计的多项式模型

参数说明：

uselapras为可选项，是否使用拉普拉斯平滑，默认为True。

当不使用拉普拉斯平滑时，即为极大似然估计。

考虑到贝叶斯估计可以弥补极大似然估计的缺点，故默认使用贝叶斯估计。

#多项式 极大似然估计、贝叶斯估计

def polynomial(file,filename,allwords\_trainhamlist\_r,allwords\_trainspamlist\_r,allwordslist\_r,usetfidf=True,uselapras=True):#返回PH和PS

    num\_words\_th\_r=len(allwords\_trainhamlist\_r)

    num\_words\_ts\_r=len(allwords\_trainspamlist\_r)

    num\_words\_t\_r=len(allwordslist\_r)

    #先验概率

    allwordslist,num\_words\_t = unrepeat(allwordslist\_r)#去重

    if usetfidf==True:#使用tfidf过滤

        words=file['wordslist']

    else:#不使用tfidf

        words,sent=wf.WordsListofFile(filename=filename,dirpath=thispath)

    Ps\_w\_dict={}

    lamda=0

    K=2

    if uselapras==True:

        #使用拉普拉斯平滑

        lamda=1

    Klamda=K\*lamda

    Ps=(num\_words\_ts\_r+lamda)/(num\_words\_t\_r+Klamda)#平滑处理

    Ph=(num\_words\_th\_r+lamda)/(num\_words\_t\_r+Klamda)

    Ph\_t=Ph

    Ps\_t=Ps

    for word in words:#将每个单词作为一个特征,以单词为粒度

        scountword=countword\_inlist(word,allwords\_trainhamlist\_r)#在ham下出现次数之和

        Pw\_h=(scountword+lamda)/(len(allwords\_trainhamlist\_r)+num\_words\_t)#类ham下单词在各个文档中出现过的次数之和+lamda/类下单词的总数(可重复)+总训练样本的不重复单词

        scountword=countword\_inlist(word,allwords\_trainspamlist\_r)

        Pw\_s=(scountword+lamda)/(len(allwords\_trainspamlist\_r)+num\_words\_t)

        Ph\_t\*=Pw\_h

        Ps\_t\*=Pw\_s

    if Ph\_t>=Ps\_t:

        return True,Ps\_t#返回垃圾邮件的概率

    else:

        return False,Ps\_t

# 四 实验结果和分析

说明：均运行了50次，比较50次的平均正确率和不同情况下的正确率分布。历次结果都在mylog.txt中附加保存。

## 4.1 两个模型的比较

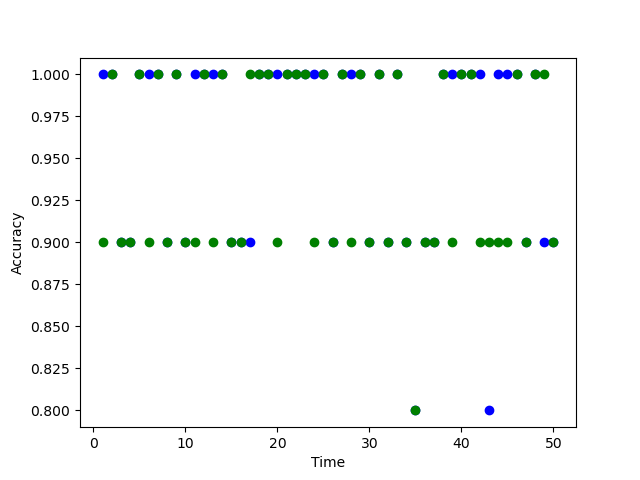
50次运行情况：每次test和train都是随机的：

给出结果：

平均正确率：

伯努利模型： 0.9599999999999997

多项式模型： 0.9439999999999995



两个模型每次运行测试20%文件的正确率比较

蓝色伯努利模型，绿色为多项式模型

可以看出，效果都比较好，但查看mylog.txt可以发现有一些文件经常被误判，如s6.txt和s7.txt。

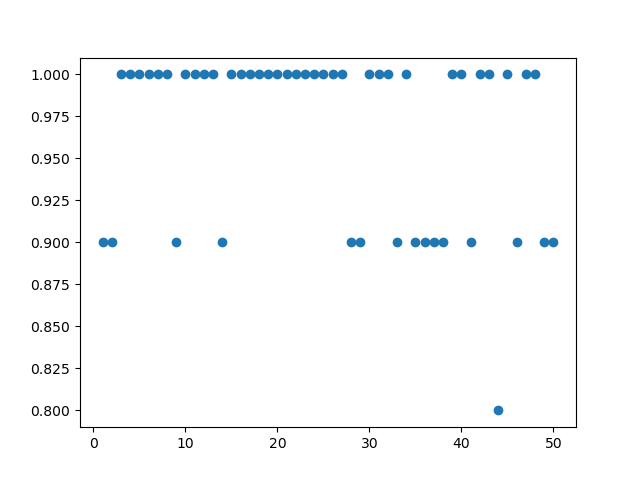
## 4.2 是否使用TF-IDF的比较

此次对比使用伯努利模型：

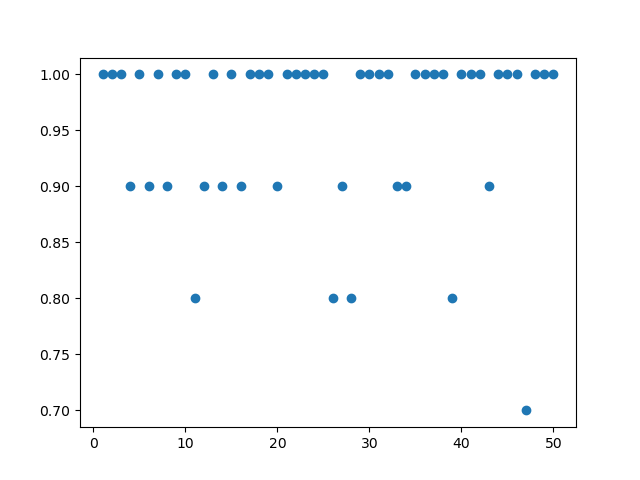
平均正确率：

使用TF-IDF： 0.956

不适用TF-IDF： 0.93222224



伯努利模型使用TF-IDF



伯努利模型未使用TF-IDF

发现，效果相差不大，使用TF-IDF的情况略胜一筹。

## 4.3 误判文件spam6和spam17分析

我发现，spam6和spam17被误判的概率特别高，观察他们每个单词的P(S|W)可以看出：

Fail：s6.txt: 0.2000000000000001

{'oem': 0.5, 'adobe': 0.5, 'microsoft': 0.5, 'softwares': 0.5, 'fast': 0.5, 'order': 0.5, 'download': 0.5, 'office': 0.5, 'professional': 0.33333333333333337, 'plus': 0.5, '2007': 0.5, '2010': 0.33333333333333337, '129': 0.6666666666666667, 'windows': 0.33333333333333337, 'extended': 0.5}

Fail：s17.txt: 0.33333333333333337

{'home': 0.5, 'based': 0.5, 'business': 0.5, 'opportunity': 0.5, 'knocking': 0.5, 'door': 0.33333333333333337, 'don': 0.5, 'rude': 0.5, 'let': 0.25, 'chance': 0.5, 'you': 0.5, 'earn': 0.5, 'here': 0.7499999999999999, 'work': 0.33333333333333337, 'from':

0.6666666666666667}

其中，P(S|W)=0.5的单词很多，这时由于计算P(W|S)时，该单词曾经未出现过，是新单词，故通过拉普拉斯平滑处理后、计算P(S|W)结果为0.5，太多的新单词影响了分类器的判断。故容易误判。