**编号**



本科生毕业设计

基于朴素贝叶斯的垃圾邮件过滤算法

Spam Filtering System based on Naive Bayes Algorithm

|  |  |
| --- | --- |
| 学 生 姓 名 | 何程斌 |
| 专 业 | 软件工程 |
| 学 号 | 150521310 |
| 指 导 教 师 | 徐晶 |
| 学 院 | 计算机科学技术学院 |

二〇一九年六月

**毕业设计（论文）原创承诺书**

1．本人承诺：所呈交的毕业设计（论文）《基于Spark的新闻实时推送系统的设计与实现》，是认真学习理解学校的《长春理工大学本科毕业设计（论文）工作条例》后，在教师的指导下，保质保量独立地完成了任务书中规定的内容，不弄虚作假，不抄袭别人的工作内容。

2．本人在毕业设计（论文）中引用他人的观点和研究成果，均在文中加以注释或以参考文献形式列出，对本文的研究工作做出重要贡献的个人和集体均已在文中注明。

3．在毕业设计（论文）中对侵犯任何方面知识产权的行为，由本人承担相应的法律责任。

4．本人完全了解学校关于保存、使用毕业设计（论文）的规定，即：按照学校要求提交论文和相关材料的印刷本和电子版本；同意学校保留毕业设计（论文）的复印件和电子版本，允许被查阅和借阅；学校可以采用影印、缩印或其他复制手段保存毕业设计（论文），可以公布其中的全部或部分内容。

以上承诺的法律结果将完全由本人承担！

作 者 签 名：

年

# 摘要

近些年随着互联网的快速发展，电子邮件得到了广泛的使用，但垃圾邮件问题也日益突出。首先介绍了垃圾邮件过滤的课题背景和国内外研究现状，然后介绍了朴素贝叶斯算法，接着提出了一种基于朴素贝叶斯算法的垃圾邮件过滤算法，使用伯努利模型计算方法对过滤算法进行了流程设计与代码实现。在 trec06 数据集上运行了测试，采用交叉验证方法计算出算法性能评估指标，分析了训练集大小、禁用词表对过滤算法性能的影响，最后证明该垃圾邮件过滤算法的有效性。

**关键字：**朴素贝叶斯；反垃圾邮件；邮件分类

# Abstract

In recent years, with the rapid development of the Internet, e-mail has been widely used, but the problem of spam has become increasingly serious. Firstly, the background of spam filtering and the research status are introduced. Then, the naive Bayes algorithm is introduced. Then a spam filtering algorithm based on naive Bayes algorithm is proposed, which is filtered by Bernoulli model. Then the process design and the code implementation are introduced. The test was run on the trec06 dataset. The crossvalidation method was used to calculate the performance evaluation index. The effect brought by size of the training set and the stopwords on the performance of the filtering algorithm were analyzed. Finally, the effectiveness of the spam filtering algorithm was proved.

**Key Words:** Navie Bayes, Anti-Spam, E-mail Classification

**目录**

[摘要 I](#_Toc10653202)

[Abstract II](#_Toc10653203)

[第 1 章 绪论 1](#_Toc10653204)

[1.1 本课题研究的目的及意义 1](#_Toc10653205)

[1.2 国内外研究现状 1](#_Toc10653206)

[1.3 本论文的组织结构 2](#_Toc10653207)

[第 2 章 朴素贝叶斯算法 4](#_Toc10653208)

[2.1 朴素贝叶斯算法分析 4](#_Toc10653209)

[2.2 朴素贝叶斯的三种常用模型 5](#_Toc10653210)

[2.2.1 高斯模型 5](#_Toc10653211)

[2.2.2 多项式模型 6](#_Toc10653212)

[2.2.3 伯努利模型 6](#_Toc10653213)

[2.2.4 本设计采用的模型 6](#_Toc10653214)

[第 3 章 垃圾邮件过滤算法实现 8](#_Toc10653215)

[3.1 系统设计 8](#_Toc10653216)

[3.2 数据集预处理 10](#_Toc10653217)

[3.3 训练算法 10](#_Toc10653218)

[3.4 分类方法 11](#_Toc10653219)

[第 4 章 算法改进 12](#_Toc10653220)

[4.1 稀疏矩阵 12](#_Toc10653221)

[4.2 拉普拉斯平滑 12](#_Toc10653222)

[4.3 小数向下溢出 13](#_Toc10653223)

[第 5 章 系统测试与评估 14](#_Toc10653224)

[5.1 测试环境 14](#_Toc10653225)

[5.2 性能评估指标 14](#_Toc10653226)

[5.3 实验结果 14](#_Toc10653227)

[总结 18](#_Toc10653228)

[参考文献 19](#_Toc10653229)

[致谢 20](#_Toc10653230)

[附录一 21](#_Toc10653231)

# 第 1 章 绪论

## 1.1 本课题研究的目的及意义

随着互联网的飞速发展，电子邮件逐渐成为人们生活十分重要的通信工具。但是，在给人们的生活带来便利的同时，电子邮件也带来了不少问题，其中最严重的问题莫过于泛滥的垃圾邮件。据 Securelist（卡巴斯基实验室旗下的 IT 安全信息资源网站）于 2017 年第三季度发布的全球垃圾邮件数据显示，2017 年第三季度，全球的电子邮件中有 58.02% 是垃圾邮件，9 月份垃圾邮件占比最高，为59.56%; 中国是垃圾邮件的最大发信国，为 12.24%，越南排在第二位（11.17%），美国排在第三位（9.62%），印度位居第四（8.49%）[1]。垃圾邮件占据了网络中巨大的带宽资资源，对用户的日常工作生活带了了极大的困扰。并且，不少垃圾邮件还附带着病毒软件和诈骗链接，给用户的计算机、甚至个人财产带来了威胁。朴素贝叶斯分类器是在特征之间相关性较弱的前提下以贝叶斯定理为基础的简单分类算法，自 20 世纪 50 年代已广泛研究。在 20 世纪 60 年代初就以另外一个名称引入到文本信息检索界中，是文本分类的一种热门（基准）方法，在垃圾邮件识别领域中已有了非常成熟的应用。本文基于朴素贝叶斯算法实现了一个垃圾邮件过滤器，对收集到的大量合法邮件和垃圾邮件作为样本进行有指导的学习，提取邮件的主题与正文，建立一个有效的特征词库，从而对垃圾邮件进行高准确率的识别。

## 1.2 国内外研究现状

将英文文本转换为单词集合非常简单和精确，因为在英语单词之间存在自然分割符号。相比之下中文比英语要复杂的多，一句话可能包含很多含义，它没有独立有效的语义，在不同语境下同一句话甚至能够体现了不同的语义。因此，与国内垃圾邮件分类的发展相比，国外发展快得多。

总体来讲，目前主要有以下几种垃圾邮件过滤技术。

* 黑白名单过滤[2]：将经常发送垃圾邮件的域名或者 IP 放入黑名单列表中，当收到新邮件时查询发件人的域名或者 IP 是否在黑名单列表中，并拒收来自黑名单地址的邮件[3]。白名单则相反，代表用户信任的发件人名单，来自白名单发件人的邮件直接视为合法邮件。但如果对方采用代理 IP、伪造地址等手段发送邮件时，该方法就失效了。
* 关键词过滤：创建垃圾邮件的特征关键词表，当收到新邮件时判断此邮件正文中是否存在这些非法关键词。但该方法太过粗放，带有非法关键词的正常邮件也会被视为垃圾邮件拦截，并且这种技术需要建立一个庞大的特征关键词表。
* 规则评分过滤：通过对大量邮件的综合分析，得到一个庞大的规则库。规则库里的每条规则都对应一个分数，计算收到的新邮件获得的分数[3]，总分超过特定的阈值时该邮件就会被判定为垃圾邮件。但因规则数量有限，无法检测匹配规则条数为 0 的邮件。决策树是规则评分过滤技术的代表，最早出现的决策树相关的学习系统是 Hunt 于 1966 年研发的;1979 年，Quinlan 提出了迭代分类器 ID3 算法;1983 年他又提出了 C4.5 算法[4,5]，改进了 ID3 算法不能处理连续值属性数据的缺点; 该算法随着数据的不断增加而加以调整，弥补了 ID3 算法的不足之处[6]。这些算法虽然在一定程度上能够满足需求，但是它们的根本原理都是预设规则比较结果来判定是否为垃圾邮件。同时这些规则往往都是静态的，缺少自助学习策略，在规律不明显的应用领域中过滤效果很差。
* 内容过滤：它可以分为规则角度和内容统计。本文研究的基于朴素贝叶斯算法的垃圾邮件过滤算法是基于内容的算法，速度较快、效率较高、耗费较少、准确性较高，并且具有自我学习功能，能够不断地动态调整垃圾邮件集和合法邮件集的概率。在文本分类领域具有十分广泛的应用。垃圾邮件过滤领域中最常用的算法便是朴素贝叶斯算法和支持向量机算法[7]。马小龙提出了 SVM-EM 朴素贝叶斯算法，该算法先利用 SVM 算法将数据集分成完整集和缺失集，计算缺失属性数据项与完整属性数据项的相关度，利用 EM 算法对数据不完整属性进行修补处理，最后利用朴素贝叶斯算法分类[8]。

## 1.3 本论文的组织结构

本文共分为五章, 组织结构如下：

第一章绪论。概述了朴素贝叶斯算法和垃圾邮件过滤的研究内容，国内外研究现状。提出了本文要研究的内容。

第二章朴素贝叶斯算法。介绍了朴素贝叶斯算法的基本概念，即朴素贝叶斯算法的步骤和方法，为之后的章节奠定坚实的理论基础。

第三章垃圾邮件过滤算法实现。首先介绍了垃圾邮件的特征等基本概念，结合第二章介绍的朴素贝叶斯算法，详细描述了将朴素贝叶斯算法应用于垃圾邮件过滤领域的过程。

第四章算法改进。介绍了第三章的算法实现的不足，分析了问题出现的原因，以及提出了改进算法的方法。

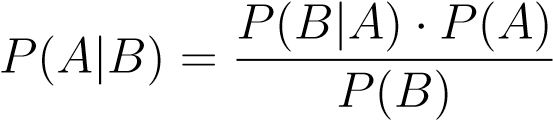
第五章系统测试与评估。本章是本文的重点，对算法进行对比测试，得到了准确率、精确率、召回率等评估数据，并分析了训练集大小、禁用词表对实验结果的影响，由此验证了基于朴素贝叶斯算法的垃圾邮件过滤算法的可行有效性。

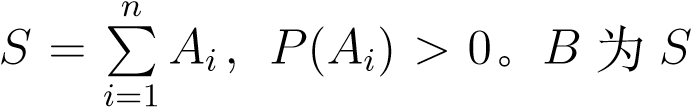
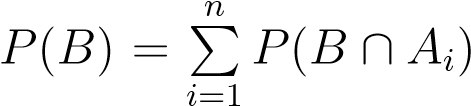
# 第 2 章 朴素贝叶斯算法

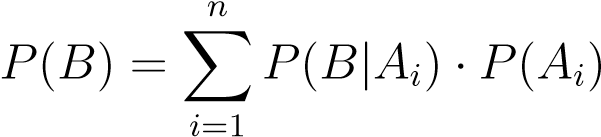
## 2.1 朴素贝叶斯算法分析

贝叶斯定理表示根据基于统计数据得到的某些特征来计算未分类的样本被分类为各个类别的概率，比较概率从而实现分类。在介绍贝叶斯定理之前，需要介绍一下条件概率和完全概率公式。

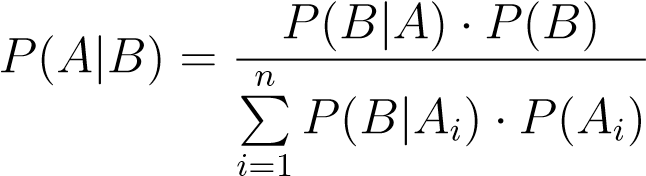
条件概率：指在事件 B 发生的前提下，事件 A 发生的概率[9]，即 ，由此可得 P(A∩B) = P(A|B)·P(B)，同理 P(A∩B) = P(B|A)·P(A)，所以 P(A|B) · P(B) = P(B|A|) · P(A)，由此就得到了条件概率的计算公式：

 (2-1)

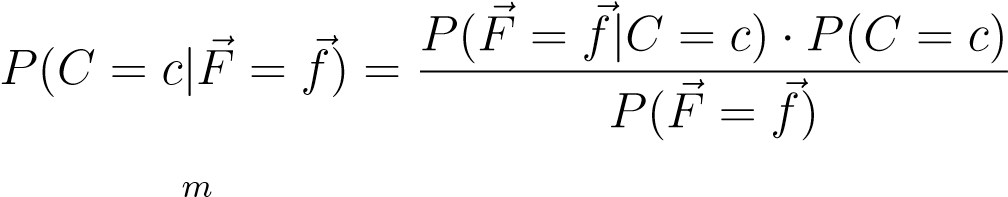
全概率公式[9]：假定样本空间 S，A1,A2,··· ,An 代表 S 的一个划分，即的某一事件，在这种情况下，事件 B 可以被分为 n 个部分，即，在先前的推导中我们已知 P(B ∩ A) =P(B|A) · P(A)，相互结合就可得到全概率公式:

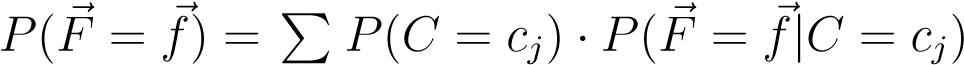
 (2-2)

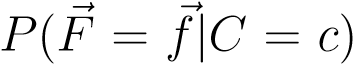
将全概率公式代入条件概率公式，就得到了贝叶斯公式：

 (2-3)

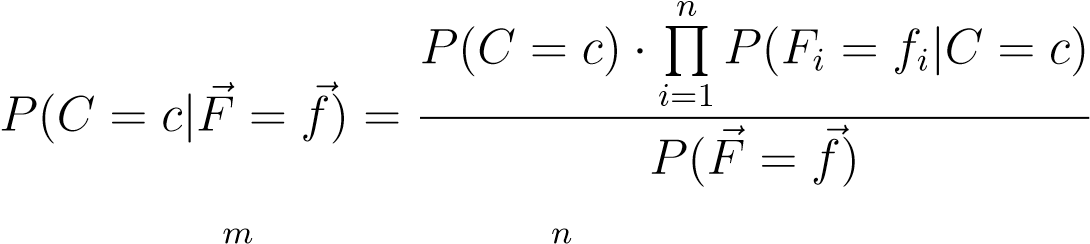
用一个 n 维特征向量 f⃗ 来表示待分类的样本 d，其中 fi(i = 1,2,...,n) 表示第 i 个特征，n 是特征的个数。则该样本 d 分类为 cj(j = 1,2,...,m) 的概率为：

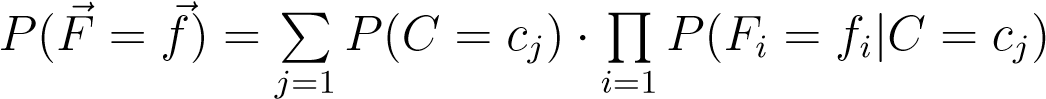
 (2-4)

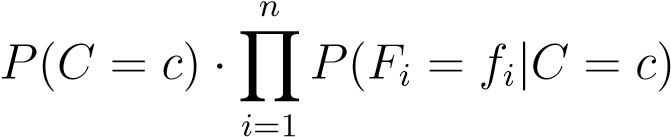
其中，。

但是，往往很难计算（F⃗ 可能有许多不同的值，并且也存在数据稀疏的问题），这导致贝叶斯定理的直接使用是十分困难的。

而朴素贝叶斯则是在贝叶斯定理的基础上，假设所有特征都相互独立，极大了降低了计算难度。与贝叶斯定理相比较，对于特征向量为 f⃗ = ⟨f1,...,fn⟩ 的待分类样本 d，其被分类为 c 分类的概率为：

 (2-5)

同时，由于对于任意分类都是相同的，所以可以将公式进一步省略为：

 (2-6)

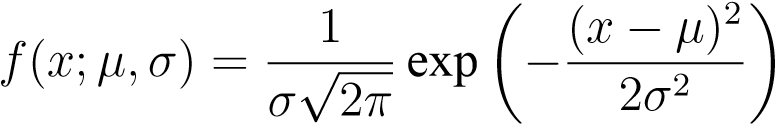
所以，仅需计算所有分类对应的该公式的值，比较大小，即可完成分类，这极大地降低了计算难度。

最为广泛的两种分类算法是决策树算法和朴素贝叶斯算法。和决策树算法相比，朴素贝叶斯算法源自古典数学理论，拥有坚实的数学基础，以及稳定的分类效率[10]。同时，朴素贝叶斯算法所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。理论上，朴素贝叶斯算法与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯算法假设属性之间相互独立，这个假设在实际应用中往往是不成立的，这给朴素贝叶斯算法的正确分类带来了一定影响。但研究表明朴素贝叶斯算法在特定领域（如文本分类，满足所有特征属性相关性较小的前提）对分类结果的准确性的影响并不大。

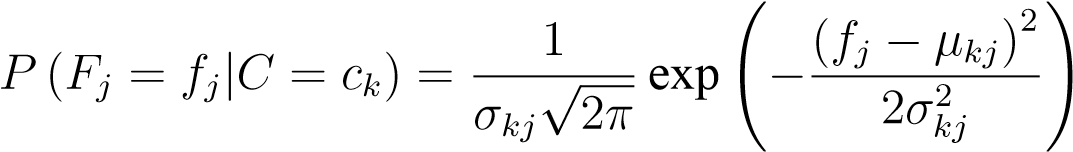
## 2.2 朴素贝叶斯的三种常用模型

### 2.2.1 高斯模型

如果特征 fj 是连续变量，可以假设它符合高斯分布 (正态分布），即假设每个类别 ck 下的 fkj 符合高斯分布。这样，我们可以通过高斯分布的概率密度函数来计算样本中 xj 某个特定值的条件概率 P(Fj = fj|C = Ck) 。高斯分布的概率密度函数为：

 (2-7)

其中 µ 是均值，σ2 是方差。假设在类别 Ck 中，特征 fj 的均值为 µkj ，方差为 σkj2 （这两项可以通过样本数据统计出来）。则

 (2-8)

处理连续数值问题的另一种常用技术是将连续数值离散化。通常当训练样本数量较少或者是精确的分布已知时，通过概率分布的方法是更好的选择。而在样本数量较多的情形下离散化的方法表现更优，因为大量的样本可以学习到数据的实际分布。通常情况下分类任务都会提供大量的数据样本，此时选择离散化方法会比概率分布估计的方法更好。

### 2.2.2 多项式模型

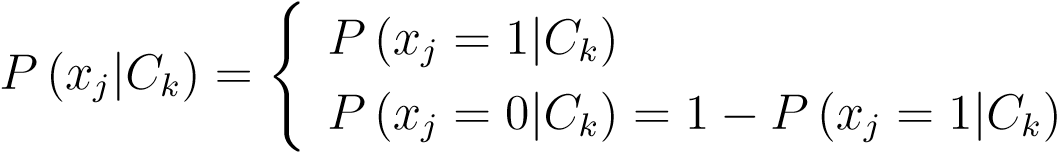
如果特征 fj 是离散值，可以假设它符合多项式分布。可以统计 fj 的某个特征在样本中的频率来估算其概率。假设特征 fj 有 Sj 个可能的取值，并且在 n 个样本中，类别为 ck 、特征 fj 取值为 s 的样本有 mkjs 个。则

P (xj|Ck) = mkjs/mk (2-9)

对于一段文本的特征向量的的每个分量的值是文本中处于该分量位置的单词在词典中的索引，其取值范围是 {1,2,...,n}，n 是词典的大小，特征向量的长度为文本中单词的数量。

### 2.2.3 伯努利模型

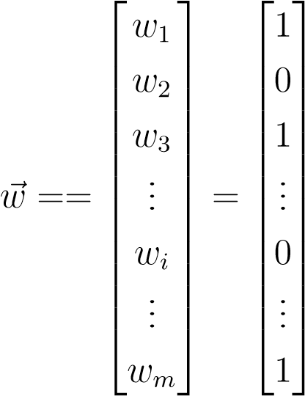
如果特征 fj 是稀疏二项离散值，可以假设它符合伯努利分布。伯努利分布只有两种可能的取值，我们将其编码为 0,1，则

 (2-10)

比如构造一个 5000 个不同单词的向量作为输入特征 w⃗，对于一段文本，其中有出现的单词，在 w⃗ 中对应单词的位置设为 1，其它位置为 0，这样 w⃗ 中的每个特征（单词）的取值为 1 或 0，符合伯努利分布。

### 2.2.4 本设计采用的模型

本设计使用伯努利模型。用向量 w⃗(m×1) 表示一个 m 个单词的字典。当邮件中出现字典 m⃗ 中的第 i 个单词时，将 wi 置 1，否则置 0。例如对于一封具 n 个词语的邮件，它的特征向量为：



邮件包含“今天”、” 促销“、”选购“等词，因此字典中的对应位置置 1。字典中未在邮件中出现过的词，则将对应的位置置为 0。而对于邮件中某些词可能并未出现在字典中，我们可以直接忽略这些词。

# 第 3 章 垃圾邮件过滤算法实现

## 3.1 系统设计

结合朴素贝叶斯算法的垃圾邮件过滤算法的流程图如图 3-1所示：

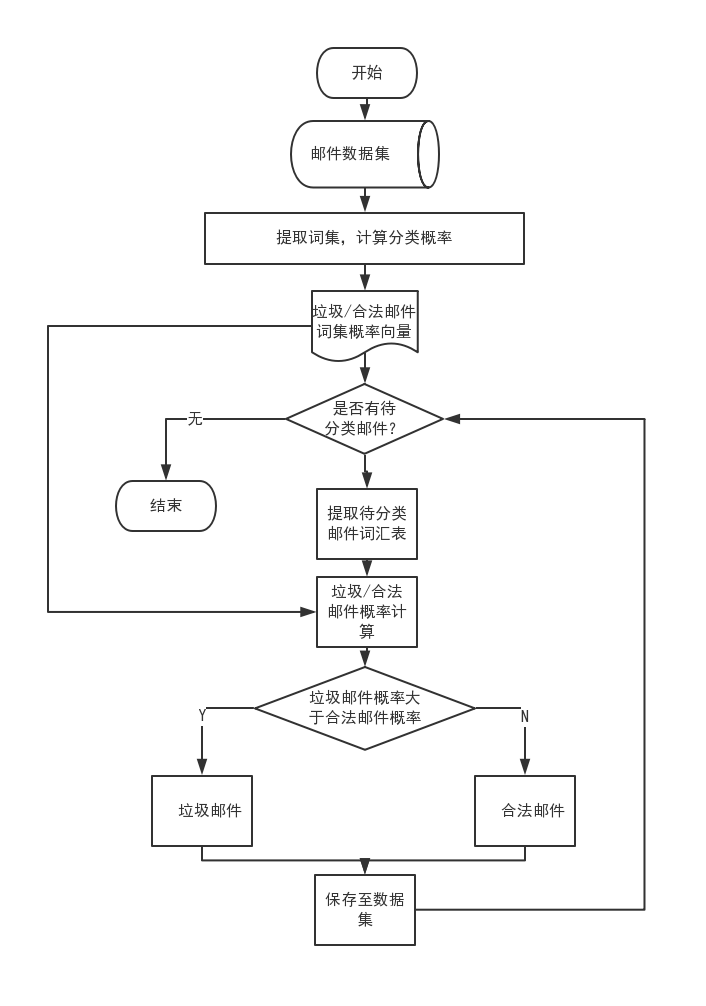


图 3-1: 朴素贝叶斯算法的垃圾邮件过滤算法的流程图

按照流程图所示，整个垃圾邮件过滤系统需要两个方面的问题，一是对现有的邮件数据集进行训练，二是对待分类的新邮件进行分类过滤。按此思想设计，得到如图 3-2所示。

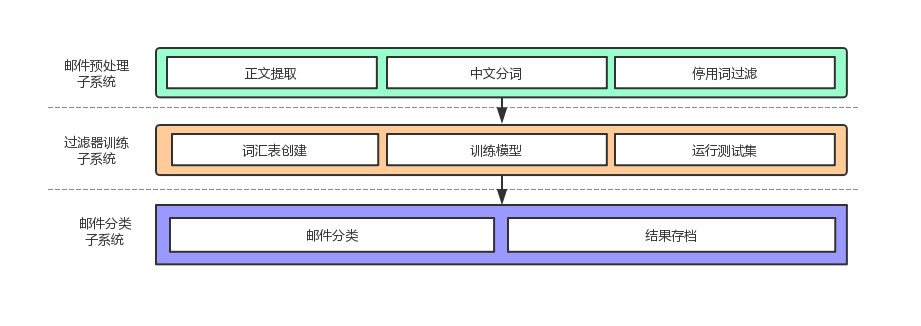


图 3-2: 系统总体架构图

依照此架构，可以将系统分割成三部分实现：

第一部分用于对邮件进行预处理，如图 3-3所示。当数据集足够大时，分词这一步骤是十分耗时的。对于 trec06 数据集来说大约有 60000 封邮件样本，要完成所有 60000 封邮件样本的分词大约需要 27 分钟，这给程序的调试运行带来了极大的困难。为解决该问题，本设计将分词这一步骤独立成单独的系统。并保存分词后的结果，留作稍后的训练步骤使用，从而加快系统运行效率。

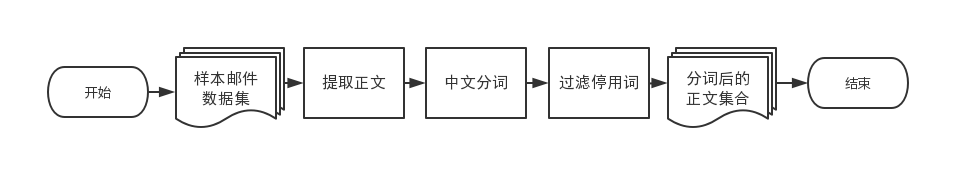


图 3-3: 邮件预处理子系统

第二部分用于对过滤器进行训练，如图 3-4所示。训练子系统在预处理子系统处理后得到的正文词语表集合上进行训练。并且，训练算法这一步骤并非每进行分类一次便要进行一次，我们只需要训练出一个准确率足够高的模型，便可供分类步骤循环使用，所以，训练步骤可以单独成一个子系统。

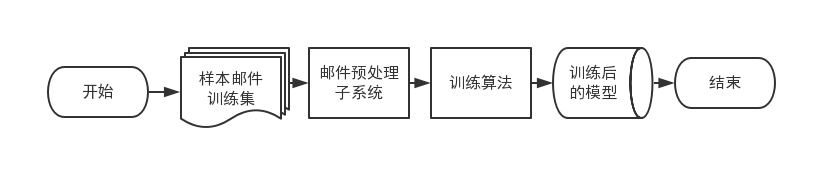


图 3-4: 过滤器训练子系统

第三部分用于对待分类邮件进行分类过滤，如图 3-5所示。分类子系统使用训练子系统得到的模型对待分类邮件进行分类。还可以将分类子系统设置成如 HTTP API 后端程序等形式作为守护进程运行，供外部程序（如邮件客户端）调用接口。

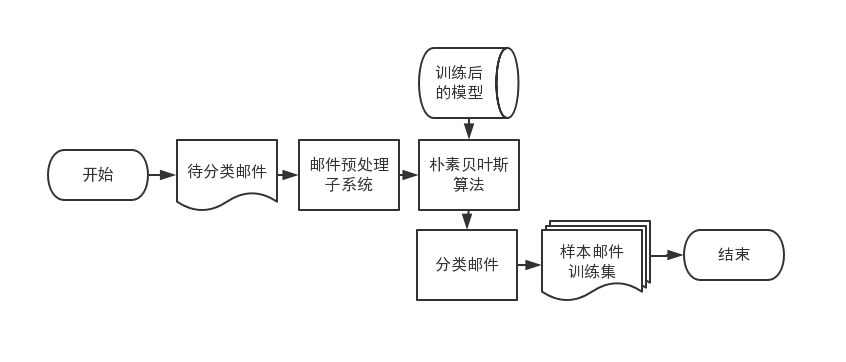


图 3-5: 邮件分类子系统

## 3.2 数据集预处理

本设计采用公开的垃圾邮件语料库 trec06。trec06 由国际文本检索会议提供，分为中英文两部分数据集。该数据集收集于真实的邮件，并且保留了邮件的原始格式。本设计仅使用 trec06 的中文数据集部分 trec06c，大约有 61810 条数据。根据多用途互联网邮件扩展（MIME）标准，对邮件格式进行解析，提取正文，对内容进行分词、过滤停用词后，依照邮件类别以纯文本文件的格式分类存储。

本设计使用的中文分词工具包为北京大学于 2019 年 2 月开源的 pkuseg[11]。相较于业界其他成熟的工具包，如 jieba、THULAC 等，pkuseg 在特定的数据集上拥有更高的粉刺准确率。根据北大研究组的测试结果，pkuseg 分别在示例数据集（MSRA 和 CTB8）上降低了 79.33% 和 63.67% 的分词错误率。

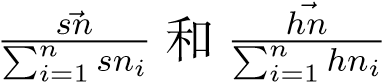
停用词指在自然语言处理中，为节省存储空间、提高处理效率，会过滤掉某些低价值的字或词，如“这”、“那”、“一个”，这些字或词即被称为停用词（Stop Words）[12]。本设计使用的停用词表为哈工大停用词表、百度停用词表和四川大学机器智能实验室停用词库的并集，包含了 2313 个停用词。

## 3.3 训练算法

首先，读取预处理分词完毕后的数据集，随机打乱，并取其中 40000 条为训练集，20000 条为测试集。将训练集中所有的词语做并集得到一个词集。当使用全部 40000 条训练样本时，得到的词集长度约为 140000 左右（本章余下全部数据都是在使用了全部 40000 条测试样本的前提下得到的）。

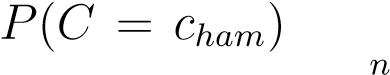
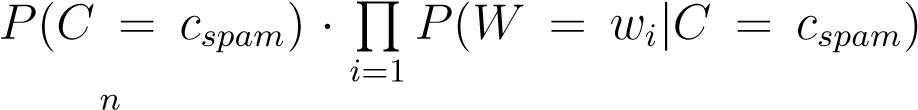
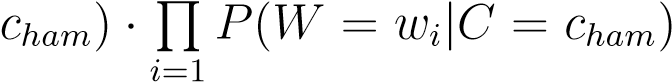
然后需要将训练集转换成大小为 40000 × 140000 的矩阵，每一条邮件数据都用一个长度为 140000 的向量表示，即 w⃗ = ⟨w1,w2,··· ,wn⟩ ，n = 140000。找到每份邮件的词语在词集中的下标，然后将向量对应下标的值设为 1，其余位置则设置为 0，表示未出现该词。

开始训练模型。用 cspam 和 cham 分别表示分类为垃圾邮件和合法邮件。统计训练集中垃圾邮件和合法邮件的数量，除以邮件总数即可得到 P(C = cspam) 和P(C = cham)。

创建两个向量 sn 和 hn ，长度为 140000，分别代表每个词在垃圾邮件中和正常邮件中出现的次数。遍历训练集矩阵，若为垃圾邮件则将向量加到 s⃗n 上，若为合法邮件则将向量加到 hn⃗ 上。最后，计算，即可得到向量 p⃗s = ⟨ps1,ps2,··· ,psn⟩和ph⃗ = ⟨ph1,ph2,··· ,phn⟩。psi 即P(W = wi|C = cspam)，在满足垃圾邮件的前提下，词语 wi 出现的概率，phi 同理。

由此，我们就得到了朴素贝叶斯算法公式的所有因子：P(C = cspam)、P(C = cham)、P(W = wi|C = cspam) 和 P(W = wi|C = cham)

## 3.4 分类方法

首先，将待分类邮件经过预处理子系统处理后得到词语列表，根据先前步骤得到的概率向量量 p⃗s = ⟨ps1,ps2,··· ,psn⟩ 和 ph⃗ = ⟨ph1,ph2,··· ,phn⟩，即可得到该邮件每个词属于垃圾邮件或合法邮件的概率。再结合 P(C = cspam) 和 ，即可根据朴素贝叶斯算法计算得出该邮件属于垃圾邮件的概率，和属于合法邮件的概率 P(C。比较两者。比较两者，取概率较大的作为该邮件的最终分类。

最后，将分类后的邮件归入到数据集中，当分类的邮件数量达到了某一阈值时（如 100），可以将分类后的邮件加入训练集重新训练模型。

# 第 4 章 算法改进

上一章节描述的过滤算法仍然有一些不足之处，如占用内存过大等问题，本章节将解释这些问题出现的原因，和解决的方法。

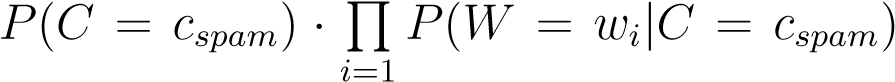
## 4.1 稀疏矩阵

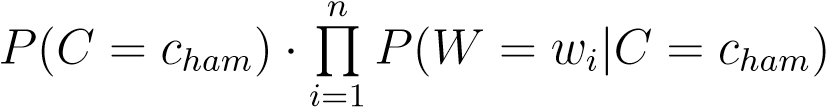
在训练模型的步骤中，我们需要使用 numpy（一个Python语言的数学库），来创建用于训练的矩阵。当使用了整个 trec06 数据集时，即 40000 封邮件，140000 个词语，矩阵大小会是 40000 × 140000，即使将数据类型指定为 uint8（numpy 创建矩阵函数默认数据类型为 float64），所需要的内存空间也将达到 40000 ×140000 × 8 = 5.21GiB，如此巨大的开销是不可接受的。

但是，当我们观察矩阵的结构：矩阵大小 m×n，即 m 个 n 维向量，m 为邮件数量，n 为词集大小。每个向量代表一封邮件，向量中的元素为 0 或 1，1 代表该邮件出现了该位置的词，0 则相反。当使用 40000 条测试样本时，词集大小为 140000 左右，而大部分邮件分词后得到的词语数量往往在几十至几百，这导致了每个向量的大部分元素的值都为 0（即数据是十分稀疏的），这使得矩阵的创建是十分浪费空间的。

根据矩阵数据的稀疏属性，我们可以使用稀疏矩阵（英语：sparse matrix）来压缩矩阵所需的内存空间。最简单的稀疏矩阵往往用一个 (x,y,v) 元组代表矩阵中的一个元素，x、y 代表元素的位置，v 代表元素的值。本设计使用 scipy（一个 Python语言的科学计算库）的 sparse\_matrix模块来创建稀疏矩阵。

## 4.2 拉普拉斯平滑

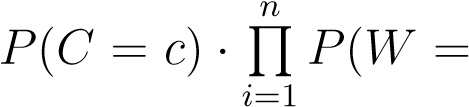
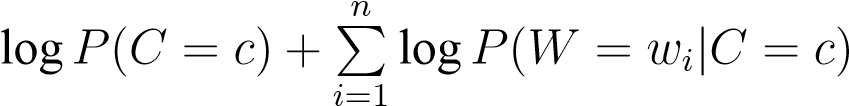
在上文的用例中，我们得到了一个大小为 140000 的词集。然而，在这 140000 个词语中，并不是所有词都在垃圾邮件或合法邮件中出现过。在上文的用例中“，爱屋及乌”“、强行”等词从未在垃圾邮件中出现过，而“买家量”“、企业链”等词则从未在合法邮件中出现过。这导致对于这些词语，psi = P(W = wi|C = cspam) 或 phi = P(W = wi|C = cham) 的计算结果为 0，使得具有该词语的邮件属于垃圾邮件的概率，或合法邮件的概率

计算为 0，从而导致无法对该邮件进行分类。

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，该方法被叫做拉普拉斯平滑[13]。可以在计算概率矩阵 p⃗s 和 ph⃗ ，统计词语在垃圾/合法邮件中出现的次数时，将初值设置为 1，即假定所有词语都在垃圾/合法邮件中至少出现过一次，当然，在计算概率时，分母也要加上 n，即词集的长度。研究表明，在训练样本很大时，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，同时也可以方便有效的避免零概率问题。

## 4.3 小数向下溢出

大多数词语的出现次数不不多，所以词语的垃圾邮件概率 psi 和合法邮件概率 phi 往往是一个十分小的小数。而上文中我们使用的计算邮件为垃圾/合法邮件概率的公式中使用了连乘，而连乘算式中的因子都是十分小的小数，导致公式得到的结果往往是一个非常小的小数，小数点后的位数能达到几十甚至几百。而我们知道计算机中浮点数往往有具体的精度，过小的小数会导致算数下溢，从而导致公式的计算结果不正确。

根据朴素贝斯算法是比较各个分类概率的大小来决定最终分类的特性，所以，只要确保分类概率之间的大小关系不被改变，我们可以对公式进行一些变形。最简单的方法就是对公式进行一次 log 运算，所以，公式wi|C = c) 变形为，从而解决了小数向下溢出的问题，同时也没有改变最终得到的各个分类的概率之间的大小关系。

# 第 5 章 系统测试与评估

## 5.1 测试环境

本次实验的环境如表 5-1所示：

表 5-1: 实验环境

|  |  |
| --- | --- |
| 操作系统 | Arch Linux |
| CPU | AMD Ryzen 5 2500U |
| 内存 | 6.8GiB |
| 系统内核 | Linux 5.1.5 |
| Python 版本 | 3.7.3 |

## 5.2 性能评估指标

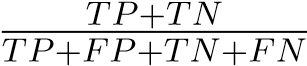
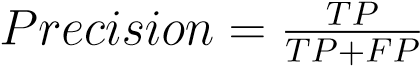
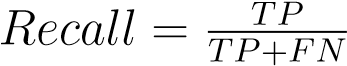
在正式开始实验测试前，需要先介绍一下衡量算法性能的评估指标：准确率（Accuracy）、精确率（Precision）和召回率（Recall）。以垃圾邮件为正例，我们假设：

真正例 (TP)：垃圾邮件被分类为垃圾邮件的数量

假正例 (FP)：合法邮件被分类为垃圾邮件的数量

假负例 (FN)：垃圾邮件被分类为合法邮件的数量

真负例 (TN)：合法邮件被分类为合法邮件的数量

准确率代表分类后的邮件中被正确分类的邮件所占的百分比，为Accuracy =。精确率代表被分类为垃圾邮件的邮件中正确分类的邮件所占的百分比，为。召回率指实际垃圾邮件中被正确分类的邮件所占的百分比，为。

## 5.3 实验结果

本文采取交叉验证的方法，从数据集中取出小部分在分类阶段测试使用，剩下的大部分作为训练阶段的训练样本集，用来测试分类器的精确性。本文依次采用 10、50、100、500、1000、5000、10000、20000、40000 封训练样本，统一采用 20000 封测试样本。同时，还以是否开启停用词过滤为变量进行了两次实验，以验证停用词对算法性能的影响。每组控制变量下运行多次测试（本文采用了十次），取准确率、精确率、召回率的平均值作为最终结果，以减轻偶然的误差对实验结果的影响。实验结果如表 5-2和表 5-3所示。

表 5-2: 开启停用词过滤时不同训练集大小的算法性能

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 训练集大小 | 词集大小平均值 | 测试集大小 | 准确率平均值 | 精确率平均值 | 召回率平均值 |
| 10 | 908 | 20000 | 0.887855 | 0.885696 | 0.955441 |
| 50 | 3306 | 20000 | 0.924845 | 0.933098 | 0.956466 |
| 100 | 5178 | 20000 | 0.947379 | 0.945317 | 0.975438 |
| 500 | 14965 | 20000 | 0.954719 | 0.955003 | 0.976015 |
| 1000 | 23275 | 20000 | 0.955300 | 0.957949 | 0.974849 |
| 5000 | 54883 | 20000 | 0.952505 | 0.967649 | 0.961841 |
| 10000 | 76204 | 20000 | 0.965700 | 0.971803 | 0.975324 |
| 20000 | 104593 | 20000 | 0.968320 | 0.979485 | 0.976428 |
| 40000 | 141830 | 20000 | 0.971235 | 0.978030 | 0.977662 |

表 5-3: 关闭停用词过滤时不同训练集大小的算法性能

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 训练集大小 | 词集大小平均值 | 测试集大小 | 准确率平均值 | 精确率平均值 | 召回率平均值 |
| 10 | 1054 | 20000 | 0.869190 | 0.900778 | 0.915103 |
| 50 | 3701 | 20000 | 0.938640 | 0.947740 | 0.958458 |
| 100 | 6043 | 20000 | 0.941490 | 0.954756 | 0.955379 |
| 500 | 15460 | 20000 | 0.947824 | 0.960926 | 0.958371 |
| 1000 | 23826 | 20000 | 0.946430 | 0.966173 | 0.955392 |
| 5000 | 54093 | 20000 | 0.954619 | 0.967474 | 0.962501 |
| 10000 | 76413 | 20000 | 0.954750 | 0.968983 | 0.961193 |
| 20000 | 105827 | 20000 | 0.957215 | 0.970641 | 0.963235 |
| 40000 | 142781 | 20000 | 0.960180 | 0.972995 | 0.965485 |

将表 5-2和表 5-3的数据转换为折线图的形式，可以更直观地分析数据之间的关系。

0

0

.

5

1

1

.

5

2

2

.

5

3

3

.

5

4

·

10

4

0

.

86

0

.

88

0

.

9

0

.

92

0

.

94

0

.

96

0

.

98

训练集大小（封）

准确率平均值

开启

停用

词过滤

-

准确

值

率平均

词过滤

关闭

停用

-

准确

率平均

值

图 5-6: 准确率变化折线图

0

0

.

5

1

1

.

5

2

2

.

5

3

3

.

5

4

·

10

4

0

.

88

0

.

9

0

.

92

0

.

94

0

.

96

0

.

98

训练集大小（封）

精确率平均值

开启

停用

词过滤

-

精确

值

率平均

关闭

停用

词过滤

-

率平均

精确

值

图 5-7: 精确率变化折线图

0

0

.

5

1

1

.

5

2

2

.

5

3

3

.

5

4

·

10

4

0

.

91

0

.

92

0

.

93

0

.

94

0

.

95

0

.

96

0

.

97

0

.

98

训练集大小（封）

准确率平均值

开启

停用

词过滤

-

召回

率平均

值

词过滤

停用

关闭

-

值

率平均

召回

图 5-8: 召回率变化折线图

从图 5-6、图 5-7和图 5-8中我们可以看出，随着训练集大小（即训练样本的数量）的增加，准确率、精准率和召回率的平均值都处于上升趋势，并逐渐趋于稳定。并且，比较开启停用词过滤和关闭停用词过滤之间的数值差别，我们可以发现，两者最终的准确率平均值、精确率平均值和召回率平均值都稳定在一个十分优秀的数值，但前者的数值往往总是大于后者的数值。同时结合表 5-2和表 5-3我们可以发现，关闭停用词过滤后的词集大小往往要大于开启停用词过滤后的词集大小，这会给系统带来些许时空上的开销。所以，停用词过滤是十分值得开启的。

当开启了停用词过滤，训练集大小为 40000 时，准确率、精准率、召回率三者的平均值都到达了最高点，它们的值分别为 0.971235、0.978030 和 0.977662，可见该垃圾邮件过滤系统是十分有效的。0.971235 的准确率代表在 20000 封测试邮件中有 575 封被错误分类（包括垃圾邮件被分类为合法邮件、合法邮件被分类为垃圾邮件），0.978030 的精确率代表在被分类为垃圾邮件的样本中有 2.197% 其实是合法邮件，而 0.977662 的召回率代表有 2.233% 的垃圾邮件被分类为合法邮件。由此可见本文的邮件过滤系统有较高的准确率、精确率和召回率，性能稳定，过滤效果较好，证明了算法的有效性。

# 总结

本文基于朴素贝叶斯算法设计并实现了一种垃圾邮件过滤算法。并根据分类结果的准确率、精确率和召回率评估该算法，经过大量实验证明该垃圾邮件过滤系统对中文垃圾邮件有着良好的过滤效果，可以在一定程度上提高垃圾邮件过滤的分类准确率，可以泛应用在未来的垃圾邮件过滤系统中。该算法主要适用于属性向量之间的混叠性比较高的数据集，如果数据集之间的混叠性比较弱，就无法体现该算法的优势。

同时，本设计仍然存在不足之处。例如没有考虑合法邮件被分类垃圾邮件给用户造成的损失要远远大于垃圾邮件被分类为合法邮件这一问题，也没考虑到错误分类的邮件加入数据集造成的脏数据问题。为了使邮件过滤系统更加人性化，还需要做出进一步的努力。下一步还可以考虑结合多种过滤方法，从而进一步的提高过滤效果。

# 参考文献

1. Darya Gudkova, Maria Vergelis, Tatyana Shcherbakova, Nadezhda Demidova.Spam and phishing in 2017
2. 王 亚 军. 反 垃 圾 邮 件 技 术 分 析 [J]. 福 建 电 脑,2016,32(07):97-98.

[EB/OL].https://securelist.com/spam-and-phishing-in-2017/83833/, 2018-02-15.

1. 张培, 纪鸿旭, 李璐. 基于朴素贝叶斯的中文垃圾邮件过滤 [J]. 信息与电脑 (理论版),2017(07):79-81.
2. HAIBO H,GARCIA E A. Learning form imbalanced data[J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(9): 1263-1284.
3. WUX,KUMARV,ROSSQUINLANJ,etal.Top10algorithmsindatamining[J].Knowledge and Information Systems, 2008, 14(1): 1-37.
4. 曹翠玲, 王媛媛, 袁野, 赵国冬. 用于垃圾邮件的贝叶斯过滤算法研究 [J]. 网络与信息安全学报,2017,3(03):64-70.
5. 杨雷, 曹翠玲, 孙建国, 张立国. 改进的朴素贝叶斯算法在垃圾邮件过滤中的研究 [J]. 通信学报,2017,38(04):140-148.
6. 马小龙. 一种改进的贝叶斯算法在垃圾邮件过滤中的研究 [J]. 计算机应用研究, 2012,

29(3):1091-1094.

1. 张雯捷. 贝叶斯定理及其应用浅析 [J]. 课程教育研究,2019(07):132-133.
2. 王阳, 周云才. 朴素贝叶斯分类算法的设计与分析 [J]. 电脑知识与技术,2019,15(11):206-

208.

1. Xu Sun, Houfeng Wang, Wenjie Li. Fast Online Training with Frequency-Adaptive Learning Rates for Chinese Word Segmentation and New Word Detection. Proceedings of ACL. 253– 262. 2012
2. 化柏林. 知识抽取中的停用词处理技术 [J]. 现代图书情报技术, 2007, 2(8): 48-51.
3. 漆原, 乔宇. 针对朴素贝叶斯文本分类方法的改进 [J]. 电子科学技术,2017,04(05):114-

116+129.

# 致谢

通过一阶段的努力，我的毕业设计《基于朴素贝叶斯的垃圾邮件过滤算法》终于完成了，这意味着大学生活即将结束。首先，我要感谢我的导师徐晶老师，他严谨细致、一丝不苟的作风一直是我学习中的榜样，给我起到了指明灯的作用。尽管事务繁忙，徐老师仍然在我彷徨和困惑的时候给与了及时的帮助和指导。在本论文的写作过程中，徐老师倾注了大量的心血，从开题报告到毕业论文的撰写，从写作提纲到一遍又一遍的指出稿中的具体问题，严格把关，循循善诱，在此我表示衷心感谢。同时，感谢所有任课老师和所有同学在这四年来给自己的指导和帮助，是他们教会了我专业知识，教会了我如何学习，教会了我如何做人。正是由于他们，我才能在各方面取得显著的进步，感谢计算机学院的所有同学，给我创造了一个团结进取，充满温暖，充满爱的大集体，使我快乐而且充实地渡过了人生中最美好的大学时光。

感谢我的同窗们，正是有了互相帮助一起学习的日子，才让我的大学多姿多彩。

# 附录一

**#程序依赖**

**Click==7.0**

**cycler==0.10.0**

**Flask==1.0.2**

**Flask-Cors==3.0.7**

**gevent==1.4.0**

**greenlet==0.4.15**

**gunicorn==19.9.0**

**imageio==2.5.0**

**itsdangerous==1.1.0**

**Jinja2==2.10.1**

**kiwisolver==1.1.0**

**MarkupSafe==1.1.1**

**matplotlib==3.0.3**

**numpy==1.16.3**

**pandas==0.24.2**

**Pillow==6.0.0**

**pkuseg==0.0.21**

**pyparsing==2.4.0**

**python-dateutil==2.8.0**

**pytz==2019.1**

**scipy==1.2.1**

**six==1.12.0**

**tqdm==4.31.1**

**Werkzeug==0.15.2**

**wordcloud==1.5.0**

**"""**

**预处理模块**

**多进程分词 trec06 数据集**

**"""**

**import glob**

**import pkuseg**

**import re**

**import pandas**

**import threading**

**from queue import Queue**

**import os**

**import click**

**from mail import \***

**from tqdm import tqdm**

**segmentation = pkuseg.pkuseg()**

**class Worker(threading.Thread):**

**"""**

**消费者，分词数据集，并存入 MySQL**

**"""**

**def \_\_init\_\_(self, queue, stop\_words, spam\_dir, ham\_dir, thread\_num):**

**threading.Thread.\_\_init\_\_(self)**

**self.queue = queue**

**self.stop\_words = stop\_words**

**self.spam\_dir = spam\_dir**

**self.ham\_dir = ham\_dir**

**self.thread\_num = thread\_num**

**def run(self) -> None:**

**print('starting worker thread', self.thread\_num)**

**while True:**

**path, spam = self.queue.get()**

**if path is None and spam is None:**

**break**

**try:**

**output\_filename = '{}{}.txt'.format(path.split('/')[-2], path.split('/')[-1])**

**output\_file\_path = os.path.join(self.spam\_dir if spam else self.ham\_dir, output\_filename)**

**if os.path.exists(output\_file\_path):**

**continue**

**mail\_content = get\_mail\_content(path)**

**# 跳过内容为空的邮件**

**if len(mail\_content) == 0:**

**continue**

**word\_list = create\_string\_word\_list(mail\_content, self.stop\_words)**

**# 跳过去掉禁用词后内容为空的邮件**

**if len(word\_list) == 0:**

**continue**

**output\_text = ','.join(word\_list)**

**with open(output\_file\_path, 'w', encoding='utf-8') as f:**

**f.write(output\_text)**

**except Exception as e:**

**print(path, e)**

**class Producer(threading.Thread):**

**"""**

**生产者，读取 full/index 文件并发送给消费者**

**"""**

**def \_\_init\_\_(self, queue, index, worker\_num):**

**threading.Thread.\_\_init\_\_(self)**

**self.queue = queue**

**self.index = index**

**self.worker\_num = worker\_num**

**def run(self) -> None:**

**print("starting producer thread, total:", len(self.index))**

**pbar = tqdm(total=len(self.index)) # CLI 进度条**

**pcount = 0 # 进度条统计**

**for i, row in self.index.iterrows():**

**self.queue.put((row.path, row.spam))**

**pcount += 1**

**if pcount == 100:**

**pbar.update(pcount)**

**pcount = 0**

**pbar.update(pcount)**

**pbar.close()**

**print('done')**

**for i in range(self.worker\_num):**

**self.queue.put((None, None))**

**def load\_stop\_words(file\_glob):**

**"""**

**加载停用词列表**

**"""**

**stop\_words = set()**

**for path in glob.glob(file\_glob):**

**with open(path, 'r') as f:**

**lines = f.readlines()**

**stop\_words |= set([i.strip() for i in lines])**

**return stop\_words**

**def create\_string\_word\_list(string, stop\_words, seg=segmentation):**

**"""**

**根据文本内容，创建这条文本的词表**

**:param string: 字符串**

**:param stop\_words: 停用词列表**

**:param seg: 分词器，如果为 None 则用全局实例**

**:return**

**"""**

**# 去除非中文字符**

**string = re.findall(u'[\u4E00-\u9FD5]', string)**

**string = ''.join(string)**

**word\_list = []**

**# pkuseg 分词**

**seg\_list = seg.cut(string)**

**for word in seg\_list:**

**if word != '' and len(word) != 0 and word not in stop\_words:**

**word\_list.append(word)**

**return word\_list**

**def get\_index(trec06c\_path):**

**index = pandas.read\_csv(os.path.join(trec06c\_path, 'full/index'), sep=' ', names=['spam', 'path'])**

**index.path = index.path.apply(lambda x: os.path.join(trec06c\_path, x[1:]))**

**index.spam = index.spam.apply(lambda x: True if x == 'spam' else False)**

**return index**

**def make\_data\_dir(target\_dir):**

**spam\_dir, ham\_dir = os.path.join(target\_dir, 'spam'), os.path.join(target\_dir, 'ham')**

**if not os.path.exists(spam\_dir):**

**os.makedirs(spam\_dir)**

**os.makedirs(ham\_dir)**

**return spam\_dir, ham\_dir**

**@click.command()**

**@click.option('-t', 'trec06c\_path', default='./trec06c', help='trec06 数据集根路径，default ./trec06c')**

**@click.option('-s', 'stopwords\_glob', default='./stopwords/\*.txt', help='禁用词表 GLOB，default ./stopwords/\*.txt')**

**@click.option('-T', 'target\_dir', default='./data', help='输出文件目标目录，default ./data')**

**@click.option('-n', 'worker\_num', default=4, help='消费者进程数，default 4')**

**def run(trec06c\_path, stopwords\_glob, target\_dir, worker\_num):**

**queue = Queue(worker\_num \* 4)**

**if len(stopwords\_glob) == 0:**

**stop\_words = set()**

**else:**

**stop\_words = load\_stop\_words(stopwords\_glob)**

**print(stop\_words)**

**print('禁用词表加载成功，长度', len(stop\_words))**

**index = get\_index(trec06c\_path)**

**spam\_dir, ham\_dir = make\_data\_dir(target\_dir)**

**threads = list()**

**producer = Producer(queue, index, worker\_num)**

**producer.start()**

**threads.append(producer)**

**for i in range(worker\_num):**

**worker = Worker(queue, stop\_words, spam\_dir, ham\_dir, i)**

**worker.start()**

**threads.append(worker)**

**for thread in threads:**

**thread.join()**

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

**run()**

**"""**

**train.py**

**训练模块**

**"""**

**import os**

**import random**

**import numpy**

**import click**

**import gc**

**from scipy.sparse import lil\_matrix, coo\_matrix**

**from tqdm import tqdm**

**from datetime import datetime**

**import pickle**

**class Model:**

**def \_\_init\_\_(self, word\_set, Pspam, Pham, PS, PH, created\_at):**

**self.word\_set = word\_set**

**self.Pspam = Pspam**

**self.Pham = Pham**

**self.PS = PS**

**self.PH = PH**

**self.created\_at: datetime = created\_at**

**def load\_dataset(data\_dir, ratio=1.0, dataset\_pickle\_filepath=''):**

**"""**

**读取数据集，随机打乱**

**:param data\_dir: 经过 preprocess.py 处理过的输出目录路径**

**:param ratio: 读取百分比**

**:param dataset\_pickle\_filepath: 数据集 pickle 文件路径，留空则不会生成/读取**

**:return dataset: 示例 [(1, ['哈哈', '呵呵'])...]，1 代表 spam 0 代表 ham**

**"""**

**dataset = list()**

**# 可选使用 pickle 文件加快数据集读取速度**

**if len(dataset\_pickle\_filepath) != 0 and os.path.exists(dataset\_pickle\_filepath):**

**print('发现数据集 pickle 文件')**

**with open(dataset\_pickle\_filepath, 'rb') as f:**

**dataset = pickle.load(f)**

**else:**

**labels = os.listdir(data\_dir)**

**for label in labels:**

**label\_dir = os.path.join(data\_dir, label)**

**filenames = os.listdir(label\_dir)**

**print('开始读取{}'.format(label))**

**# CLI 进度条**

**pbar = tqdm(total=len(filenames))**

**# 进度条计数，满 100 更新一次进度条**

**pcount = 0**

**for filename in filenames:**

**file\_path = os.path.join(label\_dir, filename)**

**with open(file\_path, 'r', encoding='utf-8') as f:**

**raw\_text = f.read()**

**words = raw\_text.split(',')**

**dataset.append((1 if label == 'spam' else 0, words))**

**pcount += 1**

**if pcount == 100:**

**pbar.update(pcount)**

**pcount = 0**

**pbar.update(pcount)**

**pbar.close()**

**if len(dataset\_pickle\_filepath) != 0:**

**with open(dataset\_pickle\_filepath, 'wb') as f:**

**pickle.dump(dataset, f)**

**print('生成 pickle 文件成功：', dataset\_pickle\_filepath)**

**random.shuffle(dataset)**

**dataset = dataset[:int(len(dataset) \* ratio)]**

**return dataset**

**def create\_word\_set(dataset):**

**"""**

**创建不含重复词的词集**

**:param dataset:**

**:return:**

**"""**

**word\_set = set()**

**for label, words in dataset:**

**word\_set |= set(words)**

**return list(word\_set)**

**def words2vector(word\_set, words, word\_dict=None):**

**if word\_dict is None:**

**word\_dict = {word: i for i, word in enumerate(word\_set)}**

**vector = numpy.zeros(len(word\_set), dtype=numpy.uint8)**

**for word in words:**

**if word in word\_set:**

**vector[word\_dict[word]] = 1**

**return vector**

**def dataset2vectors(word\_set, dataset):**

**"""**

**将数据集转化为词向量**

**:param word\_set: 由 create\_word\_set 得到的不含重复词的词集**

**:param dataset: 数据集**

**:return:**

**vectors:**

**labels:**

**"""**

**# word\_dict 的对应关系为 词 -> 该词的位置**

**word\_dict = {word: i for i, word in enumerate(word\_set)}**

**gc.collect()**

**# vector 的总数（vectors 的长度）等于邮件集的数量**

**# vector 的长度为词集的长度**

**# vector 的元素值有两种：1 代表词集中该位置的词在本邮件中出现了，0 则相反**

**# 使用稀疏矩阵解决内存不足的问题**

**vectors = lil\_matrix((len(dataset), len(word\_set)), dtype=numpy.uint8)**

**# vectors = numpy.zeros((len(dataset), len(word\_set)), dtype='float64')**

**# labels 的长度为邮件集的数量**

**# labels 的元素值有两种：1 代表该位置的邮件为垃圾邮件，0 则为正常邮件**

**labels = numpy.zeros(len(dataset), dtype=numpy.uint8)**

**# CLI 进度条**

**pbar = tqdm(total=len(dataset))**

**# 进度条计数，满 100 更新一次进度条**

**pcount = 0**

**for i, (label, words) in enumerate(dataset):**

**for word in words:**

**if word in word\_dict:**

**vectors[i, word\_dict[word]] = 1**

**labels[i] = label**

**pcount += 1**

**if pcount == 100:**

**pbar.update(pcount)**

**pcount = 0**

**pbar.update(pcount)**

**pbar.close()**

**return vectors, labels**

**def train\_nb(vectors, labels, word\_set=None):**

**"""**

**训练模型**

**:param vectors:**

**:param labels:**

**:return:**

**Pspam: P(是垃圾邮件)**

**Pham: P(是正常邮件)**

**PS: P(是垃圾邮件|出现该词) 的矩阵**

**PH: P(是正常邮件|出现该词) 的矩阵**

**"""**

**Pspam = sum(labels) / labels.shape[0] # 垃圾邮件概率**

**Pham = 1 - Pspam # 非垃圾邮件概率**

**gc.collect()**

**# 这里要用拉普拉斯平滑**

**SN = numpy.ones(vectors.shape[1], dtype=numpy.float64)**

**HN = numpy.ones(vectors.shape[1], dtype=numpy.float64)**

**coo\_vectors = coo\_matrix(vectors)**

**# CLI 进度条**

**pbar = tqdm(total=sum(coo\_vectors.data))**

**# 进度条计数，满 100 更新一次进度条**

**pcount = 0**

**for i, j, v in zip(coo\_vectors.row, coo\_vectors.col, coo\_vectors.data):**

**# 如果是垃圾邮件**

**if labels[i] == 1:**

**SN[j] += v**

**else:**

**HN[j] += v**

**pcount += 1**

**if pcount == 100:**

**pbar.update(pcount)**

**pcount = 0**

**pbar.update(pcount)**

**pbar.close()**

**PS = SN / sum(SN)**

**PH = HN / sum(HN)**

**return Pspam, Pham, PS, PH**

**def predict\_nb(vectors, Pspam, Pham, PS, PH, show\_progress\_bar=True):**

**"""**

**判断垃圾/正常邮件**

**:param vectors: 需要判断的邮件集的词向量矩阵**

**:param Pspam: P(是垃圾邮件)**

**:param Pham: P(是正常邮件)**

**:param PS: P(是垃圾邮件|出现该词) 的矩阵**

**:param PH: P(是正常邮件|出现该词) 的矩阵**

**:return predict\_vector: 预测结果矩阵，1 代表该位置的邮件为垃圾邮件，0 则正常**

**"""**

**# log 防止小数溢出**

**Pspam, Pham = numpy.math.log(Pspam), numpy.math.log(Pham)**

**PS, PH = numpy.log(PS), numpy.log(PH)**

**predict\_vector = numpy.zeros(vectors.shape[0], dtype=numpy.uint8)**

**probalities = list()**

**if show\_progress\_bar:**

**# CLI 进度条**

**pbar = tqdm(total=vectors.shape[0])**

**# 进度条计数，满 100 更新一次进度条**

**pcount = 0**

**for i, vector in enumerate(vectors):**

**pspam = Pspam + sum(vector \* PS)**

**pham = Pham + sum(vector \* PH)**

**if pspam > pham:**

**predict\_vector[i] = 1**

**if show\_progress\_bar:**

**pcount += 1**

**if pcount == 100:**

**pbar.update(pcount)**

**pcount = 0**

**probalities.append((pspam, pham))**

**if show\_progress\_bar:**

**pbar.update(pcount)**

**pbar.close()**

**# predict\_vector = [1 if (Pspam + sum(vector \* PS)) >= (Pham + sum(vector \* PH)) else 0 for vector in vectors]**

**return predict\_vector, probalities**

**def split\_vectors\_by\_ratio(vectors, labels, train\_size, test\_size):**

**"""**

**按照一定的比例切分向量**

**:param vectors:**

**:param labels:**

**:param train\_size: 训练集大小**

**:param test\_size: 测试集大小**

**:return:**

**"""**

**train\_num = int(vectors.shape[0] \* train\_size)**

**train\_vectors = vectors[:train\_size]**

**train\_labels = labels[:train\_size]**

**test\_vectors = vectors[train\_size:train\_size + test\_size]**

**test\_labels = labels[train\_size:train\_size + test\_size]**

**return train\_vectors, train\_labels, test\_vectors, test\_labels**

**def calc\_accuracy(predicts, labels):**

**"""**

**计算准确率**

**:param predicts:**

**:param labels:**

**:return:**

**"""**

**return numpy.mean(predicts == labels)**

**def calc\_precision(predicts, labels):**

**"""**

**计算精确率**

**:param predicts:**

**:param labels:**

**:return:**

**"""**

**tp = 0**

**for i in range(predicts.shape[0]):**

**if predicts[i] == labels[i] == 1:**

**tp += 1**

**return tp / sum(predicts)**

**def calc\_recall(predicts, labels):**

**"""**

**计算召回率**

**:param predicts:**

**:param labels:**

**:return:**

**"""**

**tp = 0**

**for i in range(predicts.shape[0]):**

**if predicts[i] == labels[i] == 1:**

**tp += 1**

**return tp / sum(labels)**

**def run(train\_size, test\_size, dataset\_pickle\_filepath, output\_pickle\_filepath):**

**dataset = load\_dataset('./data', 1, dataset\_pickle\_filepath)**

**dataset = dataset[:train\_size + test\_size]**

**train\_dataset = dataset[:train\_size]**

**test\_dataset = dataset[train\_size:]**

**print('数据集读取成功，总数 {}'.format(len(dataset)))**

**word\_set = create\_word\_set(train\_dataset)**

**print('词集创建成功，长度', len(word\_set), '，开始创建词向量')**

**train\_vectors, train\_labels = dataset2vectors(word\_set, train\_dataset)**

**test\_vectors, test\_labels = dataset2vectors(word\_set, test\_dataset)**

**print('词向量创建成功')**

**# train\_vectors, train\_labels, test\_vectors, test\_labels = split\_vectors\_by\_ratio(vectors, labels, train\_size, test\_size)**

**print('数据集切分成功，训练集长度 {}×{}，测试集长度 {}×{}'.format(train\_vectors.shape[0], train\_vectors.shape[1],**

**test\_vectors.shape[0], test\_vectors.shape[1]))**

**print('开始训练，训练集长度 {}×{}'.format(train\_vectors.shape[0], train\_vectors.shape[1]))**

**Pspam, Pham, PS, PH = train\_nb(train\_vectors, train\_labels, word\_set)**

**print('开始测试，测试集长度 {}×{}'.format(test\_vectors.shape[0], test\_vectors.shape[1]))**

**predict\_vector, probabilities = predict\_nb(test\_vectors, Pspam, Pham, PS, PH)**

**accuracy = calc\_accuracy(predict\_vector, test\_labels)**

**precision = calc\_precision(predict\_vector, test\_labels)**

**recall = calc\_recall(predict\_vector, test\_labels)**

**print('测试完成，准确率 {}，精确率 {}，召回率 {}%'.format(accuracy, precision, recall))**

**# 防止 \_\_main\_\_.Model 问题**

**# https://stackoverflow.com/questions/40287657/load-pickled-object-in-different-file-attribute-error**

**from train import Model**

**output = Model(word\_set, Pspam, Pham, PS, PH, datetime.now())**

**with open(output\_pickle\_filepath, 'wb') as f:**

**pickle.dump(output, f)**

**return word\_set, accuracy, precision, recall**

**@click.command()**

**@click.option('-t', 'train\_size', default=40000, help='训练集集大小，default 40000')**

**@click.option('-T', 'test\_size', default=20000, help='测试集大小, default 20000')**

**@click.option('-d', 'dataset\_pickle\_filepath', default='',**

**help='dataset pickle 文件路径，程序在读取数据集时会自动生成/读取 pickle 文件以加快读取速度，default \'\'')**

**@click.option('-o', 'output\_pickle\_filepath', default='model.pickle',**

**help='输出 pickle 路径，{word\_set, Pspam, Pham, PS, PH}, default \'model.pickle\'')**

**# @click.option('-f', 'float\_precision', default=16, help='浮点数精度，available16 32 64，default 16')**

**def main(train\_size, test\_size, dataset\_pickle\_filepath, output\_pickle\_filepath):**

**run(train\_size, test\_size, dataset\_pickle\_filepath, output\_pickle\_filepath)**

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

**main()**

**"""**

**web.py**

**分类模块，Web 后端**

**"""**

**from flask import Flask, request, jsonify**

**from flask\_cors import CORS**

**from imageio import imread**

**from wordcloud import WordCloud**

**from train import \***

**from preprocess import \***

**from web\_config import config**

**app = Flask(\_\_name\_\_, static\_folder='./static')**

**CORS(app) # allow CORS for all domains on all routes**

**model: Model**

**chinese\_font\_path: str**

**stop\_words: set**

**def load\_model(model\_pickle\_filepath):**

**with open(model\_pickle\_filepath, 'rb') as f:**

**return pickle.load(f)**

**def setup\_app(app: Flask):**

**"""**

**进行一些初始化工作**

**:param app:**

**:return:**

**"""**

**app.debug = config['debug']**

**global model, chinese\_font\_path, stop\_words**

**model = load\_model(config['model\_pickle\_filepath'])**

**model.word\_dict = {word: i for i, word in enumerate(model.word\_set)}**

**chinese\_font\_path = config['zh\_font\_filepath']**

**stop\_words = load\_stop\_words(config['stopwords\_file\_glob'])**

**setup\_app(app)**

**def api\_response(success=True, data={}, code=0, message=''):**

**return {**

**'success': success,**

**'data': data,**

**'code': code,**

**'message': message**

**}**

**@app.route('/')**

**def index():**

**return 'Hello World'**

**@app.route('/wordcloud/spam')**

**def spam\_word\_cloud():**

**"""**

**垃圾邮件词云**

**:return:**

**"""**

**created\_at = model.created\_at**

**filename = 'spam\_wordcloud{}{}{}{}{}.png'.format(created\_at.year, created\_at.month, created\_at.day, created\_at.hour,**

**created\_at.minute)**

**filepath = os.path.join('./static/images', filename)**

**if not os.path.exists(filepath):**

**word\_set = model.word\_set**

**PS = model.PS**

**word\_dict = {word: ps for word, ps in zip(word\_set, PS)}**

**image\_mask = imread('./static/images/MailIcon.jpg')**

**wc: WordCloud = WordCloud(font\_path=chinese\_font\_path, mask=image\_mask,**

**max\_words=2000).generate\_from\_frequencies(word\_dict)**

**wc.to\_file(filepath)**

**return app.send\_static\_file(os.path.join('images', filename))**

**@app.route('/wordcloud/ham')**

**def ham\_word\_cloud():**

**"""**

**正常邮件词云**

**:return:**

**"""**

**created\_at = model.created\_at**

**filename = 'ham\_wordcloud{}{}{}{}{}.png'.format(created\_at.year, created\_at.month, created\_at.day, created\_at.hour,**

**created\_at.minute)**

**filepath = os.path.join('./static/images', filename)**

**if not os.path.exists(filepath):**

**word\_set = model.word\_set**

**PH = model.PH**

**word\_dict = {word: ps for word, ps in zip(word\_set, PH)}**

**image\_mask = imread('./static/images/MailIcon.jpg')**

**wc = WordCloud(font\_path=chinese\_font\_path, mask=image\_mask, max\_words=2000).generate\_from\_frequencies(**

**word\_dict)**

**wc.to\_file(filepath)**

**return app.send\_static\_file(os.path.join('images', filename))**

**@app.route('/identify', methods=['POST'])**

**def identify\_mail():**

**"""**

**判断邮件为垃圾邮件还是正常邮件**

**POST /identify**

**{**

**"content": "邮件正文"**

**}**

**:return:**

**{**

**"success": true,**

**"data": {**

**"type": "spam", // 或者 ham**

**"warnMessage": "结果可能不正确：..." // 当分词得到的词过少时会携带警告信息**

**}**

**message: ''**

**}**

**"""**

**data = request.get\_json()**

**content = data['content']**

**if len(content) == 0:**

**return jsonify(api\_response(success=False, message='内容为空'))**

**words = create\_string\_word\_list(content, stop\_words)**

**vector = words2vector(model.word\_set, words, model.word\_dict)**

**vectors = numpy.zeros((1, vector.shape[0]), dtype=numpy.uint8)**

**vectors[0] = vector**

**results, probabilities = predict\_nb(vectors, model.Pspam, model.Pham, model.PS, model.PH, show\_progress\_bar=False)**

**pspam, pham = probabilities[0]**

**ret\_data = {'type': 'spam' if results[0] == 1 else 'ham', "pspam": pspam, "pham": pham}**

**if len(set(words)) < 20:**

**ret\_data['warnMessage'] = '结果可能不正确：分词并去重后仅得到{}个词'.format(len(set(words)))**

**return jsonify(api\_response(data=ret\_data))**

**@app.route('/stat')**

**def stat():**

**return jsonify(api\_response(data={**

**"wordSetCount": len(model.word\_set),**

**"stopWordsCount": len(stop\_words),**

**"Pspam": model.Pspam,**

**"Pham": model.Pham,**

**}))**

**@app.route('/stat/wordset')**

**def word\_set():**

**return jsonify(model.word\_set)**

**@app.route('/stat/stopwords')**

**def stopwords():**

**return jsonify(list(stop\_words))**

**@app.route('/stat/PS')**

**def PS():**

**return jsonify(model.PS.tolist())**

**@app.route('/stat/PH')**

**def PH():**

**return jsonify(model.PH.tolist())**

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

**app.run(host=config['host'], port=config['port'], debug=config['debug])**