概率论与数理统计

扩展任务 8 试验报告

GHe

GitHub：https://github.com/GHe0000/SpamEmailBayesClassifier

Build: 2025-04-04

Typst Version: 0.13.1

目录

1 程序设计和原理阐述

1.1 词频-逆文档频率（TF-IDF）

TF-IDF 是一种统计方法，旨在评估某个词对于一个文档的重要程度，其核心思想是：一个词在文档中出现的频率越高（TF），同时在语料库中的常见程度越低（IDF），则认为这个词越能代表文档，其重要性越高.

Definition 1.1.1 (词频(TF)): 词频衡量了某个词在文档中出现的频率，词频越大，词对当前文档的重要性可能越大.

$$\mathrm{TF}(t,d)=\frac{\text{词 t 在文档 d 中出现的次数}}{\text{文档 d 的总词数}}$$

Definition 1.1.2 (逆文档频率(IDF)): 逆文档频率衡量了某个词在整个语料库中的普遍性或稀缺性，逆文档频率越高，词在整个语料库中越稀有，则认为这个词越能代表文档，其重要性越高.

$$\mathrm{IDF}(t,D)=\log\left(\frac{\text{语料库的文档总数 N}+1}{\text{包含词 t 的文档数}+1}\right)$$

注意在上述 IDF 的计算中，并没有使用完全标准的 IDF 计算公式，而是经过了一个所谓的“+1”平滑，这是为了避免文档频率为 0 的“除零”问题，从而增强了数值的稳定性. 同时这种处理本身也类似于 Laplace 平滑，对 IDF 值的影响不大，同时也提升了模型的稳健性.

Remark: 所谓的 *“+1”* 平滑，可以看成是 *Bayes* 学派中的**先验分布**. 当我们通过数据得到频率后，相当于“更新”了这个先验，得到后验. 当数据量足够大时，先验的取值并不会影响最终的结果.

所谓的 TF-IDF，就是将 TF 和 IDF 两个指标综合起来，作为衡量词对于文档的重要性的最终指标. 其计算公式如下：

$$\text{TF-IDF}(t,d,D)=\mathrm{TF}(t,d)\cdot\mathrm{IDF}(t,D)$$

Mark: 这里有一个有意思的小问题：为何 TF 直接是频率和 IDF 经过了 log 变换？

TF 不需要 log 变换很好理解，因为其反映的是词在文档中的**局部信息**，其信息量关于词频是**线性**的. 若某个词出现的次数是另一个词的两倍，则其包含的信息也认为是另一个词的两倍.

但 IDF 不同，一个词出现在文档中越少，则其携带的信息越多，且这种信息的增长量是**非线性**的. 例如一个词在 1 篇文档中出现与在 10 篇文档中出现，其重要性差异远大于在 100 篇文档中出现与在 1000 篇文档中出现的差异. 而 log 正好能将这种非线性的增长量转化为线性的增长量.

从信息论的角度看，一个事件 x 的**自信息**定义为：

$$I(x) = -\log(P(x))$$

其核心思想是：事件发生的概率越低，其发生时携带的信息量越大. 这里 log 将概率的乘法关系转化为信息量的加法关系（例如，独立事件联合概率的信息量为各事件信息量之和）.

因此实际上，IDF 就是“某个词在文档中出现”这一个事件带来的**信息量**.[1]

这里我们将 TF 和 IDF 两个指标相乘，得到 TF-IDF 值，实际上可以将 IDF 看成是 TF 的权重. 这样，既通过 IDF 抑制了常见词，也通过 TF 强调了文档中的重要词. 因此 TF-IDF 常常作为衡量一个词对于一个文档的重要性的最终指标.

1.2 Bayes 分类器具体算法

现在假设我们有某个邮件 d，其包含了 w 个词：$d = \{v_1, v_2, ..., v_w\}$. 然后我们有训练集 $D = \{d_1, d_2, ..., d_m\}$，其中 d_m 是训练集中的邮件. 每一个邮件 d_m 都对应一个标签 l_m，表示该邮件是否是垃圾邮件（Spam）或正常邮件（Ham）.

1.2.1 训练过程

- **生成词汇表**：从训练数据集 D 中生成词汇表 $V = \{v_1, v_2, ..., v_n\}$，其中包含了最高频出现的前 n 个词（提前去除停用词），词汇表 V 即 Bayes 分类器的特征空间.
- **计算先验**：对于每个标签 $l_m \in L$，计算先验概率 $P(l_m)$. 这里我们用频率来估计先验概率，即 $P(l_m) = m_l/m$，其中 m_l 是训练集中标签为 l_m 的邮件的数量，m 是训练集中邮件的总数.
- **计算 TF-IDF**：对于每个标签 $l_m \in L$，提取出训练集中所有为 l_m 的邮件 D_{l_m}，$\forall d \in D_{l_m}$，统计其词汇表中每个词的 TF-IDF 总和，得到 Ω，其中：

$$\forall \omega_n \in \Omega, \quad \omega_n = \sum_{d \in D} \text{TF-IDF}(v_n, d, D), \quad v_n \in V, d \in D$$

[1] 这里在 Shannon 的论文 The Mathematical Theory of Communication 中对于信息均默认取以 2 为底的对数，但实际上 log 的底数是可以任取的，这里取 10 为底.

TF-IDF 根据下面式子进行计算：

$$\text{TF-IDF}(v_n, d, D) = \frac{c_n}{c_d} \cdot \log\left(\frac{m+1}{m_n+1}\right)$$

c_n 是邮件 d 中词 v_n 出现的次数，c_d 是邮件 d 的总词数，m 是训练集 D 中邮件的总数，m_n 是训练集 D 中包含词 v_n 的邮件数.

- **计算条件概率**：通过如下式子计算当标签为 l_m 时，词 v_n 对应的 TF-IDF 值出现的概率（同样使用 Laplace 平滑）：

$$P(\omega_n|l_m) = \frac{\omega_n + s}{\sum_{\omega_n \in \Omega} \omega_n + s\ |V|}$$

其中 s 是 Laplace 平滑的超参数（在代码中设为 1），$|V|$ 是词汇表 V 的大小（即 n）.

上述过程中得到的词汇表 V、先验概率 $P = \{P(l_m)|l_m \in L\}$、条件概率 Q，以及词汇的 IDF 值 I 即分类器的全部参数. 其中:

$$Q = \{P(\omega_n|l_m)|\omega_n \in \Omega, l_m \in L\}$$

$$I = \{\text{IDF}(v_n, D)|v_n \in V\}$$

1.2.2 预测过程

现在我们假设有了已经训练好的参数 V，P，Q，I，以及待预测的邮件 $d = \{v_1, v_2, ..., v_w\}$，则预测过程如下：

- **计算 TF-IDF**：对于邮件 d，计算其词汇表中每个词的 TF-IDF 值，得到 Ω：

$$\forall \omega_n \in \Omega, \quad \omega_n = \text{TF-IDF}(v_n, d, I) = \frac{c_n}{c_d} \cdot i_{v_n}, \quad v_n \in V, i_n \in I$$

其中 c_n 是邮件 d 中词 v_n 出现的次数，c_d 是邮件 d 的总词数，i_n 是词 v_n 在训练集 D 中的 IDF 值.

Remark: 这里计算 *TF-IDF* 时认为带预测的邮件和训练集中的邮件类似，因此可以使用词汇在训练集中的 *IDF*（也就是保存的 I）来计算 *TF-IDF* 值.

- **计算类别概率**：通过如下式子计算邮件 d 属于标签 l_m 的概率：

$$\log(P(l_m|d)) = \log(P(l_m)) + \sum_{\omega_n \in \Omega} \log(P(\omega_n|l_m))$$

选取概率最大的标签作为预测结果.

2 程序实现和结果分析

2.1 数据准备以及数据清洗

这里我们使用 2006 TREC Public Spam Corpora[2] 中的 trec06c 中文垃圾邮件数据集作为实验数据集.

数据集提供了一个 label 文件，标记了邮件的类别（Spam 还是 Ham），以及对应的邮件文本的相对路径. 邮件文本的一个例子如下：

```
Test
```

这里我们首先
1. 定位邮件头部结束标志，仅保留邮件正文内容
2. 移除编码字符串和 HTML 标签
3. 将所有的标点、特殊符号替换为空格
4. 使用 `jieba` 库进行分词
5. 过滤去除所有长度等于 1 的词和停用词，并去除纯数字和电子邮件地址等无意义词

上述过程对应的代码如下：

```
def preprocess(raw_text, stopwords):
    header_end = re.search(r'\n\s*\n', raw_text)
    body = raw_text[header_end.end():] if header_end else raw_text
    cleaned = re.sub(r'=\?gb2312\?B\?.*?\?=', ' ', body)
    cleaned = re.sub(r'<.*?>', ' ', cleaned)
    cleaned = re.sub(r'[^\u4e00-\u9fa5a-zA-Z0-9]', ' ', cleaned)
    words = jieba.lcut(cleaned)
    filtered = [
        word.lower() for word in words
        if len(word) > 1
        and word not in stopwords
        and not re.match(r'^\d+$', word)
        and not re.match(r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
word)
    ]
    return ' '.join(filtered)
```

这样我们就将一个邮件文本转换为一个词序列，例如上述例子处理后的结果为：

```

```

对于 label 文件中的每一行，我们都按照路径找到对应的邮件文本，转化为词序列后存到对应类别下的文件夹中，这样我们就得到了经过处理的数据.

[2]https://plg.uwaterloo.ca/~gvcormac/treccorpus06/

2.2 分类器算法实现

这里我们采用函数式编程的思想，将分类器分为多个纯函数的组合[3]：

2.3 实验结果及分析

3 总结

[3]一般的处理是将分类器抽象为一个对象，这也是实际中更常见的处理，但为了好玩，这里采用函数式编程的思想来实现分类器.