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: 2025-04-05

Typst Version: 0.13.1

目录

本文使用了如下 AI 模型：

- DeepSeek-R1：各种提问，各种解释，辅助编写文档
- FittenCode：代码编写辅助 AI

本文所有代码（包括本 PDF 编写使用的 Typst 源码以及模板）以及部分预处理好的数据均公开在 GitHub 仓库 https://github.com/GHe0000/SpamEmailBayesClassifier 中.

由于文件较多，代码量较长，因此本文并未放出全部代码，仅对关键部分进行了简要阐述.

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：

$$\log(P(l_m|d)) = \log(P(l_m)) + \sum_{\omega'_n \in \Omega'} \omega'_n \cdot \log(P(\omega_n|l_m)),\quad \omega_n \in \Omega$$

> **Mark**: 这里也有一个小问题，为什么使用对数进行计算？为什么是 $\omega'_n \cdot \log(P(\omega_n|l_m))$ 的形式？
>
> 实际上词汇表 V 中的每一个词 v_n 构成了特征空间的一组基，张成了一个 n 维的向量空间，且这组基是相互正交的（这里实际上暗含了词之间是相互独立的这一假设）

每一个邮件都可以由特征空间的一个向量表示，且其分量就是对应词的在该文档和语料库中的 TF-IDF 值.

Remark: 从另一个方面来看使用对数进行计算同时也防止了浮点下溢，尤其是长文档.

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：

```
Received: from 163.con ([61.141.165.252])
  by spam-gw.ccert.edu.cn (MIMEDefang) with ESMTP id j7CHJ2B9028021
  for <xing@ccert.edu.cn>; Sun, 14 Aug 2005 10:04:03 +0800 (CST)
Message-ID: <200508130119.j7CHJ2B9028021@spam-gw.ccert.edu.cn>
From: =?GB2312?B?1cW6o8TP?= <jian@163.con>
Subject: =?gb2312?B?uavLvtK1zvEutPq/qreixrGjoQ==?=
To: xing@ccert.edu.cn
Content-Type: text/plain;charset="GB2312"
Date: Sun, 14 Aug 2005 10:17:57 +0800
X-Priority: 2
X-Mailer: Microsoft Outlook Express 5.50.4133.2400

尊敬的贵公司(财务/经理)负责人您好!
        我是深圳金海实业有限公司 (广州。东莞) 等省市有分公司。
    我司有良好的社会关系和实力，因每月进项多出项少现有一部分
    发票可优惠对外代开税率较低，增值税发票为5%其它国税.地税.
    运输.广告等普通发票为1.5%的税点，还可以根据数目大小来衡
    量优惠的多少，希望贵公司.商家等来电商谈欢迎合作。

      本公司郑重承诺所用票据可到税务局验证或抵扣!
    欢迎来电进一步商谈。
    电话：13826556538（24小时服务）
    信箱：szlianfen@163.com
```

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

```
联系人：张海南

顺祝商祺

深圳市金海实业有限公司
```

对于一个邮件文本，我们需要进行如下预处理：

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[3]，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：

```
尊敬 公司 财务 经理 负责人 您好 深圳 金海 实业 有限公司 广州 东莞 省市 分公司 我司 良好
社会关系 实力 每月 进项 多出 项少 现有 一部分 发票 优惠 对外 代开 税率 增值税 发票 国税
地税 运输 广告 普通发票 税点 数目 大小 来衡 优惠 希望 公司 商家 来电 商谈 欢迎 合作 公司
郑重 承诺 所用 票据 税务局 验证 抵扣 欢迎 来电 进一步 商谈 电话 小时 服务 信箱
szlianfen com 联系人 海南 顺祝 商祺 深圳市 金海 实业 有限公司
```

[3]这里停用词列表使用百度停用词表.

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.

2.2 分类器具体实现

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[4]：

前文过程可以抽象为几个纯函数：

- `build_vocab`：传入一组邮件词序列，返回词汇表
- `calc_idf`：计算词汇表中每个词在训练集的 IDF 值
- `doc2vec`：传入一个邮件的词序列，返回每个词汇表中词的 TF-IDF 值，其构成一个特征向量
- `train`：传入训练集，返回分类器参数
- `predict`：传入分类器参数和邮件的词序列，返回其预测结果

2.2.1 构建词汇表

```
def build_vocab(docs, max_size=None):
    df_counter = pipe(
        docs,
        lambda seq: map(lambda doc: {w: 1 for w in set(doc)}, seq),
        lambda seq: reduce(
            lambda a, b: {k: a.get(k, 0) + b.get(k, 0) for k in a.keys() |
b.keys()},
            seq,
            {}
        )
    )

    sorted_vocab = sorted(
        df_counter.keys(),
        key=lambda x: (-df_counter[x], x)
    )

    vocab = list(
        w for i, w in enumerate(sorted_vocab)
        if max_size is None or i < max_size
    )

    return vocab
```

1. 文档频率统计：
   - 每个文档先转换为词集合（去重）

[4]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.

- 用 MapReduce 计算每个词出现的文档总数（每个文档计 1 次）

2. 排序处理：
   - 按文档频率降序排列
   - 同频词按字母升序排列
3. 截断控制：
   - 保留前 max_size 个高频词

2.2.2 计算 IDF 值

```
def calc_idf(docs, vocab):
    doc_count = len(docs)
    df_counter = map_reduce(
        lambda doc: defaultdict(int, {w: 1 for w in set(doc)}),
        lambda a, b: {k: a.get(k, 0) + b.get(k, 0) for k in set(a) | set(b)}
    )(docs)

    idf = np.array([
        np.log((doc_count + 1) / (df_counter.get(w, 0) + 1))  # 采用加 1 平滑
        for w in vocab
    ])
    return idf
```

1. 文档频率统计：
   - 每个文档先转换为词集合（去重）
   - 用 MapReduce 计算每个词在多少文档中出现过（DF 值）

2. IDF 计算：
   - 对词汇表每个词计算 IDF

2.2.3 计算特征向量

```
def doc2vec(doc, vocab, idf):
    tf = pipe(
        doc,
        lambda d: Counter(d),
        lambda cnt: [cnt.get(w, 0) / len(doc) for w in vocab]  # 计算词频
    )
    return np.multiply(tf, idf)
```

1. 词频计算：
   - 使用 Counter 统计文档原始词频
   - 按 vocab 词序生成向量

2. TF-IDF 合成：
   - 对每个词执行元素级乘法：TF * IDF

2.2.4 训练

```
def train_bayes(X_train, y_train, smooth=1.0, vocab_size=None):
    # 构建特征空间
    vocab = build_vocab(X_train, max_size=vocab_size)
    idf = calc_idf(X_train, vocab)

    # 计算类别先验
    classes, counts = np.unique(y_train, return_counts=True)
    prior = counts / counts.sum()

    # 计算条件概率
    def class_probability(c):
        class_docs = [doc for doc, label in zip(X_train, y_train) if label ==
c]
        vecs = [doc2vec(doc, vocab, idf) for doc in class_docs]
        total = sum(vecs)  # 向量相加
        return (total + smooth) / (total.sum() + smooth * len(vocab))

    likelihood = np.array([class_probability(c) for c in classes])
    return {
        'vocab': vocab,
        'idf': idf,
        'classes': classes,
        'prior': prior,
        'likelihood': likelihood
    }
```

2.2.5 预测

```
def predict(model, X_test):
    vocab = model['vocab']
    idf = model['idf']

    def predict_single(doc):
        vec = doc2vec(doc, vocab, idf)
        log_probs = np.log(model['prior']) + np.sum(
            vec * np.log(model['likelihood']), axis=1
        )
        return model['classes'][np.argmax(log_probs)]

    return [predict_single(doc) for doc in X_test]
```

2.3 实验结果及分析

这里我们对每个类别分别选取 3000 个邮件进行训练，并选取 500 个邮件作为测试集进行测试，并分析模型的性能.

这里我们约定，正为垃圾邮件（Spam），负为正常邮件（Ham）：

- TP：预测为正，实际为正的数量
- TN：预测为负，实际为负的数量
- FP：预测为正，实际为负的数量
- FN：预测为负，实际为正的数量

2.3.1 准确率

准确率是分类器预测正确的邮件所占的比例，我们可以计算准确率如下：

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

用 Python 代码可以及其简单地计算准确率（同样地，应用函数式编程思想来简化代码）：

```
accuracy = np.mean(np.array(y_pred) == np.array(y_test))
```

在 500 个测试集邮件中，准确率为 0.952811.

2.3.2 精确率

精确率是分类器预测出的真实的垃圾邮件所占所有预测出的垃圾邮件的比例，计算公式如下：

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

用如下 Python 代码计算：

```
y_true_01 = np.array(y_test) == 'spam'  # 转换为 0-1 标签
y_pred_01 = np.array(y_pred) == 'spam'  # 转换为 0-1 标签
precision = y_true_01[y_pred_01].mean()
```

在 500 个测试集邮件中，精确率为 0.968685.

2.3.3 召回率

召回率是分类器预测出的真实的垃圾邮件占所有真实的垃圾邮件的比例，计算公式如下：

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

用如下 Python 代码计算：

```
recall = y_pred_01[y_true_01].mean()
```

在 500 个测试集邮件中，召回率为 0.935484.

2.3.4 F1 值

F1 值是精确率和召回率的调和平均值，综合了精确率和召回率，计算公式如下：

$$\mathrm{F1} = 2 * \frac{\mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

用如下 Python 代码计算：

```
f1 = (2 * precision * recall / (precision + recall))
```

在 500 个测试集邮件中，F1 值为 0.951795.

2.3.5 混淆矩阵

混淆矩阵是一个二维表，其中每一行表示实际类别，每一列表示预测类别，表格中的每个元素表示实际类别为 i 而预测类别为 j 的邮件数量. 通过分析混淆矩阵，我们可以了解模型预测的准确率、召回率、F1 值等指标.

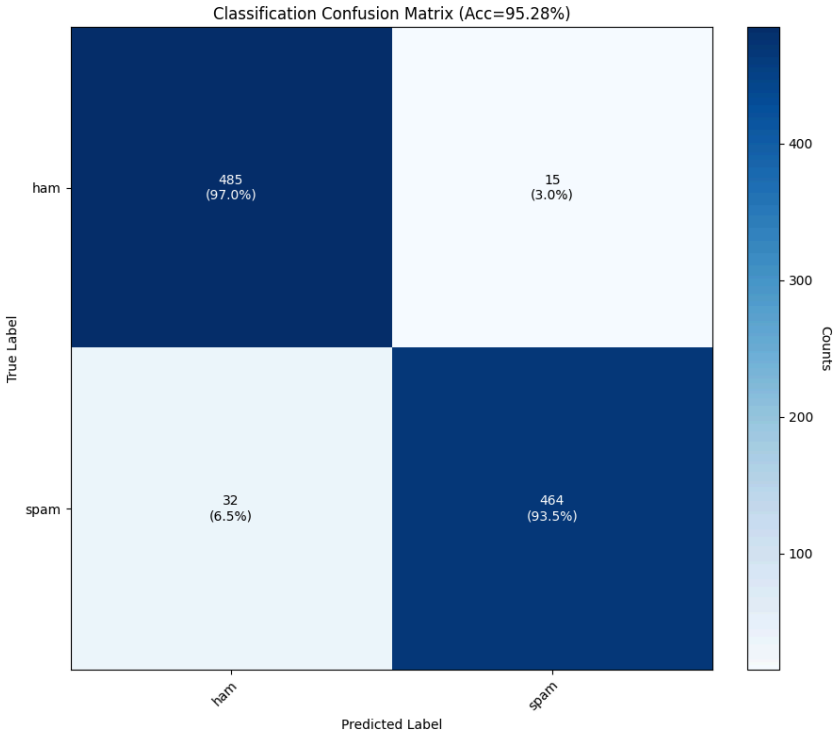

图 2.1　混淆矩阵

2.3.6 每个类别出现概率最大的词

当我们通过训练得到 $Q = \{P(\omega_n|l_m)|\omega_n \in \Omega, l_m \in L\}$ 时，我们可以统计每个类别中出现概率最大的词

ham 一个: 0.0132 mm: 0.0122 知道: 0.0121 gg: 0.0114 没有: 0.0109 觉得: 0.0108 现在: 0.0099 喜欢: 0.0086 他们: 0.0082 一起: 0.0078

spamn 公司: 0.0185 发票: 0.0172 http: 0.0155 www: 0.0132 com: 0.0127 您好: 0.0094 合作: 0.0085 优惠: 0.0082 代开: 0.0080 cn: 0.0080

2.4 最“垃圾邮件”的邮件

3 总结

本文使用 Python 以函数式编程的方式实现了一个简单的 Bayes 分类器，并对其性能进行了评估.

上述模型的实际上仍可以在下面几个方面进行改进：

- **特征空间的选取：**在模型，我们默认每个词之间是相互独立的，这显然不符合常识. 同时，这也使得我们只能截取高频词进行分析. 因此一种改进方法是使用 Word2Vec 模型，通过深度学习的方式将词语映射到特征空间中，此时特征空间中的正交的“特征”完全由机器学习而来，且对于任意词语，均能进行处理.
- **考虑上下文：**在模型中，实际上词语出现的先后顺序对模型分类的结果并无影响，但实际上词语的上下文（如前后词、句子等）在我们的判断中是至关重要的，因此可以使用 n-gram 或其他更复杂的模型来考虑上下文.