|  |  |
| --- | --- |
| **机器学习实验报告** | |
| 第 3 次 | |
| 图片包含 标牌  已生成极高可信度的说明 | |
| **姓名** | 何铭韬 |
| **班级** | 工设2204 |
| **学号** | 2223312318 |
| **电话** | 13034642319 |
| **Email** | 2780530464@qq.com |
| **日期** | 6.15 |

目录

[1实验题目 3](#_Toc199579034)

[2实验目的 3](#_Toc199579035)

[3实验环境与工具 3](#_Toc199579036)

[4实验背景与理论 3](#_Toc199579037)

[5核心代码 3](#_Toc199579038)

[5.1 TODO 1 3](#_Toc199579039)

[5.2 TODO 2 3](#_Toc199579040)

[5.3 TODO 3 3](#_Toc199579041)

[6实验结果 4](#_Toc199579042)

[7 调整预处理函数 4](#_Toc199579043)

[7.1 预处理函数代码 4](#_Toc199579044)

[7.2 实验结果 4](#_Toc199579045)

[8 扩展：实现伯努利方法 4](#_Toc199579046)

[8.1代码 4](#_Toc199579047)

[8.2 实验结果 4](#_Toc199579048)

[9总结与收获 5](#_Toc199579049)

# 1实验题目

基于朴素贝叶斯方法的英文新闻文本分类实验

# 2实验目的

（对本次实验任务进行理解，明确实验需掌握的技能）

掌握朴素贝叶斯分类算法（Term Frequency 与 Bernoulli 两种变体）的原理与实现。

熟悉文本预处理流程，包括分词、停用词过滤、词干提取/词形还原等操作。

理解拉普拉斯平滑在计算条件概率时的作用及实验效果差异。

比较TF方法与伯努利方法在实际分类任务中的性能差异。

# 3实验环境与工具

（填写本机或实验室服务器的硬件配置，列出操作系统、Python 版本及核心第三方库）

硬件环境：Intel® Core™ i5-13700kf，NVIDIA GeForce RTX 4070super, 32 GB 内存

操作系统：Windows 11 64 位

开发语言：Python 3.9

第三方库：

nltk 3.8.1 （用于分词、停用词、词干/词形还原）

numpy 1.24.2 （用于矩阵运算）

# 4实验背景与理论

文本分类常用的朴素贝叶斯方法假设各特征（单词）条件独立，通过统计训练集中各种单词在各类别中文档中出现的频率或出现概率，结合先验概率进行分类决策。

**词袋模型 (Bag of Words)：**将文本映射到高维稀疏向量，向量长度等于词汇表大小，每个维度代表一个词的出现次数或二值出现标志

**Term Frequency (TF) 方法：**使用词频来估计条件概率

其中 是词 在类别的训练语料中出现的总次数，为词汇表大小，分母加实现拉普拉斯平滑

**Bernoulli (Binary) 方法：**仅考虑词是否出现，

其中是在类别的文档中出现过词的文档数，为该类别中文档数，分母加2为二项式平滑。

**预处理步骤：**统一小写、去标点、分词、去除停用词和数字、词干提取或词形还原；这些操作能减少词汇表规模，去除噪声，提高分类性能

# 5核心代码

(请勿粘贴截图)

## 5.1 TODO 1

（计算P(c)）

**# 计算每个类别中文档数**

**sent\_fre = np.zeros((1, 4), dtype=np.int32)**

**for x, y in zip(data\_x, data\_y):**

**sent\_fre[0, y] += 1**

**# 计算 P(c)：类别 c 的文档数 / 总文档数**

**total\_docs = len(data\_y)**

**p\_c = sent\_fre[0] / total\_docs**

## 5.2 TODO 2

（计算P(w\_i|c)，并加入拉普拉斯平滑）

**# 构建词频矩阵 word\_fre[V,4] 已统计完毕**

**V = len(dicts)**

**p\_stage = np.zeros((V, 4), dtype=np.float32)**

**for c in range(4):**

**denom = np.sum(word\_fre[:, c]) + V**

**for w\_idx in range(V):**

**numer = word\_fre[w\_idx, c] + 1**

**p\_stage[w\_idx, c] = numer / denom**

## 5.3 TODO 3

（TODO 计算对应的预测值并统计，注意过滤未收录词）

**ln\_p\_c = np.log(p\_c)**

**ln\_p\_s = np.log(p\_stage)**

**count = 0**

**for x, y\_true in zip(data\_x, data\_y):**

**ln\_p = ln\_p\_c.copy()**

**for word in x:**

**if word in dicts:**

**ln\_p += ln\_p\_s[dicts[word]]**

**y\_pred = np.argmax(ln\_p)**

**if y\_pred == y\_true:**

**count += 1**

**print('Accuracy: {}/{} {:.2f}%'.format(count, len(data\_y), 100 \* count / len(data\_y)))**

# 6实验结果

**结果(文字)：**

训练方法: TF

还原方法: stemmer

load success

训练集准确率:

Accuracy: 9479/10208 92.86%

测试集准确率:

Accuracy: 2227/2552 87.26%

训练方法: TF

还原方法: lemmatizer

load success

训练集准确率:

Accuracy: 9550/10208 93.55%

测试集准确率:

Accuracy: 2226/2552 87.23%

**结果描述和分析：**

词形还原略优于词干提取，尤其训练集准确率提升约0.7%，说明 lemmatizer 能更准确地合并词形变体；

测试集性能差异不大，均在87%左右，说明模型具有一定泛化能力。

# 7 调整预处理函数

## 7.1 预处理函数代码

**# 原始去停用词：**

**# sent = [s for s in sent if not ((s in stopwords) or s.isdigit())]**

**# 修改后：保留所有词，仅去数字**

**sent = [s for s in sent if not s.isdigit()]**

## 7.2 实验结果

**结果：**

训练方法: Bernoulli

还原方法: stemmer

load success

训练集准确率:

Accuracy: 9305/10208 91.17%

测试集准确率:

Accuracy: 2150/2552 84.29%

训练方法: TF

还原方法: lemmatizer

load success

训练集准确率:

Accuracy: 9511/10208 93.17%

测试集准确率:

Accuracy: 2214/2552 86.76%

**结果描述和分析：**

保留停用词后，模型准确率略有下降（训练/测试集分别降低约0.3%和0.3%），说明去除停用词有助于减少噪声，提高分类效果。

# 8 扩展：实现伯努利方法

## 8.1代码

**# 统计每类文档中出现词的文档数 word\_in\_class[V,4]**

**for x, y in zip(data\_x, data\_y):**

**class\_docs[y] += 1**

**for word in set(x):**

**if word in dicts:**

**word\_in\_class[dicts[word], y] += 1**

**# 计算 P(w\_i|c)（伯努利），二项式拉普拉斯平滑**

**for c in range(4):**

**for w\_idx in range(V):**

**numer = word\_in\_class[w\_idx, c] + 1**

**denom = class\_docs[c] + 2**

**p\_stage[w\_idx, c] = numer / denom**

## 8.2 实验结果

**结果：**

训练方法: TF

还原方法: stemmer

load success

训练集准确率:

Accuracy: 9305 / 10208 ≈ 91.17%

测试集准确率:

Accuracy: 2150 / 2552 ≈ 84.29%

训练方法: TF

还原方法: lemmatizer

load success

训练集准确率:

Accuracy: 9378 / 10208 ≈ 91.87%

测试集准确率:

Accuracy: 2149 / 2552 ≈ 84.23%

**结果描述和分析：**

伯努利方法略逊于 TF 方法，可能是因为仅用二值特征忽略了词频信息，但模型更为简洁，在小语料上也能达到84%以上的准确率。

# 9总结与收获

（用 3–5 条要点形式总结本次实验的主要收获）

掌握了朴素贝叶斯两种常见实现（TF 与 Bernoulli）的原理与代码实现。

理解并验证了拉普拉斯平滑在概率估计中的必要性，避免零概率问题。

实验对比了词干提取与词形还原两种预处理方法，lemmatizer性能略优；

验证了去停用词对噪声过滤的积极作用，模型性能得到提升；

对比 TF 与 Bernoulli方法，体会到特征设计（词频 vs 二值）对分类效果的影响。