



基于贝叶斯方法的文本分类

授课教师：庞善民

助教：张浩、刘卓

2022年10月22日



分别用Term Frequency与Bernoulli方法
实现基于贝叶斯方法的文本分类算法
在给定的数据集上进行训练与测试

词袋模型 Bag of Words(BoW)

将文本转化为向量

如将下边的两个句子当作文本库：

Each state has its own laws.

Every country has its own culture.

词汇表(不计标点符号)：

each state has its own laws

every country culture

则两个句子分别转化成了如下向量：

(1, 1, 1, 1, 1, 1, 0, 0, 0)

(0, 0, 1, 1, 1, 0, 1, 1, 1)

向量定长，长度与词汇表长度一致

词	句1	句2
each	1	0
state	1	0
has	1	1
its	1	1
own	1	1
laws	1	0
every	0	1
country	0	1
culture	0	1

词频：词在句中出现的次数

优点：
简单方便

缺点：
一段文本只会用到词汇表中的一部分词，对大文本库，通过这种方法获得的向量会很稀疏(即包含很多0)

文本上下文之间的关联(即文本中单词的顺序)信息被抹除了

对中文文本需要引入额外的分词工具进行词组切分

停用词

即在文本中极为常见或无实际意义，无法起到分类作用的词

例如：so, and, or, the, a, ...

构建文本向量时，通常要将这些停用词删掉，不放入词汇表中，以减少向量的维度（臃肿程度）

除了停用词，标点符号、数字也可以认为是与分类无关的内容，可将之删去

停用词

如将下边的两个句子当作文本库：

A swimmer likes swimming,
thus he swims.

He is a swimmer.



词汇表：

swimmer likes swimming he swims

不含a、thus、is

例如

词干提取
(Stemming)

所得未必是真实的单词
计算复杂度较低、速度较快

词	提取	还原
swimming	swim	swimming
swims	swim	swim
thus	thu	thus
likes	like	like

词形还原
(Lemmatization)

所得必然是真实的单词
计算复杂度较高、速度较慢

Term Frequency (朴素贝叶斯)

文档类别的集合为C，共计k类： $C = \{c_1, c_2, \dots, c_k\}$

训练集的词汇表为D，共计m词： $D = \{d_1, d_2, \dots, d_m\}$

待分类的一个文档内容为text： $text = \{w_1, w_2, \dots, w_n\}$

目标：text所属的类别 c_{text}



假设已去停用词
并还原
且都在D范围内

$$c_{text} = \operatorname{argmax}_{c \in C} P(c|text)$$

$$P(c|text) = \frac{P(c, text)}{P(text)} = \frac{P(text|c)P(c)}{P(text)}$$

对同一text，分母相同，
只需要比较分子

$$c_{text} = \operatorname{argmax}_{c \in C} P(text|c)P(c)$$

即这类文档数量在所有
文档数量中的占比

$$P(c) = \frac{N(c, text)}{N(text)}$$

Term Frequency (朴素贝叶斯)

朴素贝叶斯：
各特征之间相互独立

$$P(\text{text}|c) = P(w_1, w_2, \dots, w_n|c)$$

$$P(\text{text}|c) = \prod_{i=1}^n P(w_i|c), w_i \in D \rightarrow w_i \text{ 一共会有 } m \text{ 种可能, } d_1 \sim d_m$$

显然有 $\sum_{j=1}^m P(d_j|c) = 1$

即词在这类文档中出现的次数(词频)/这类文档的总词数(含重复)

$$P(w_i|c) = \frac{N(w_i \text{ in } W_c)}{N(W_c)} \rightarrow$$

拉普拉斯平滑：

实际使用时

$$\text{通常令 } P(w_i|c) = \frac{N(w_i \text{ in } W_c) + 1}{N(W_c) + m}$$

既能防止 $P(w_i|c) = 0$

又能保持 $\sum_{j=1}^m P(d_j|c) = 1$

程序中连乘容易趋向于0，于是取对数

$$c_{\text{text}} = \operatorname{argmax}_{c \in C} \prod_{i=1}^n P(w_i|c) P(c)$$

$$c_{\text{text}} = \operatorname{argmax}_{c \in C} \left[\ln P(c) + \sum_{i=1}^n \ln P(w_i|c) \right]$$

Bernoulli (Binary) (Optional)

文档类别的集合为 C ，共计 k 类： $C = \{c_1, c_2, \dots, c_k\}$

训练集的词汇表为 D ，共计 m 词： $D = \{d_1, d_2, \dots, d_m\}$

待分类的一个文档内容为 $text$ ： $text = \{w_1, w_2, \dots, w_n\}$

目标： $text$ 所属的类别 c_{text}



假设已去停用词
并还原
且都在 D 范围内

$$c_{text} = \operatorname{argmax}_{c \in C} P(c|text)$$

$$P(c|text) = \frac{P(c, text)}{P(text)} = \frac{P(text|c)P(c)}{P(text)}$$

对同一 $text$ ，分母相同，
只需要比较分子

$$c_{text} = \operatorname{argmax}_{c \in C} P(text|c)P(c)$$

即这类文档数量在所有
文档数量中的占比

$$P(c) = \frac{N(c, text)}{N(text)}$$

Bernoulli (Binary) (Optional)

$$P(text|c) = \prod_{i=1}^m P(d_i|c)^b (1 - P(d_i|c))^{1-b}, d_i \in D, b = \begin{cases} 1 & \text{if } d_i \in text \\ 0 & \text{else} \end{cases}$$

即这类文档中出现该词的文档个数/这类文档的总个数

$$P(d_i|c) = \frac{N(C_{d_i})}{N(C)}$$



拉普拉斯平滑：
实际使用时

$$\text{通常令 } P(d_i|c) = \frac{N(C_{d_i})+1}{N(C)+2}$$

$$c_{text} = \operatorname{argmax}_{c \in C} \prod_{i=1}^m P(d_i|c)^b (1 - P(d_i|c))^{1-b} P(c)$$

$$c_{text} = \operatorname{argmax}_{c \in C} \left[\ln P(c) + \sum_{i=1}^m \ln P(d_i|c)^b (1 - P(d_i|c))^{1-b} \right]$$

4类文本，从1-4

文本库中有4个句子，从①到④，分别是1-4类的句子

词汇表中有8个词，从A到H

TF方法：

统计词频如下：

	1	2	3	4
A	2	1	5	1
B	4	3	3	0
C	0	3	4	0
D	3	0	4	0
E	2	1	2	3
F	1	5	0	3
G	0	4	2	2
H	6	0	1	1

如黑框这一列，表示1类中的所有句子(也就是句①)，一共有18个词，其中A出现了2次，B出现了4次...

以句①与类1为例：

平滑计算 $P(w|c)$

$$P(A|1), P(B|1), \dots = \frac{2+1}{18+8}, \frac{4+1}{18+8}, \dots$$

计算 $P(\text{text}|c)$

$$P(\textcircled{1}|1) = P^2(A|1)P^4(B|1) \dots P^6(H|1)$$

计算 $P(c)$

$$P(1) = \frac{1}{4}$$

比较 $\ln P(\text{text}|c)P(c)$

$$\ln P(\textcircled{1}|1)P(1) = 2 \ln P(A|1) + \dots + 6 \ln P(H|1) + \ln P(1)$$

找1-4类中的 argmax

4类文本，从1-4

文本库中有8个句子，从①到⑤都是1类的句子，⑥⑦⑧分别是2-4类的句子

词汇表中有8个词，从A到H

Bernoulli方法：

统计如下：

	1	2	3	4
A	4	1	0	1
B	3	0	1	1
C	2	0	1	0
D	0	1	0	1
E	1	0	1	1
F	5	1	0	1
G	3	1	1	0
H	2	0	1	1

如黑框这一列，表示1类中的所有句子(共5个)，有4个出现了A，3个出现了B...

以句①与类1为例：

平滑计算 $P(d|c)$

$$P(A|1), P(B|1), \dots = \frac{4+1}{5+2}, \frac{3+1}{5+2}, \dots$$

假设句①中有ABCEFGH这些词

计算 $P(\text{text}|c)$

$$P(\textcircled{1}|1) = P(A|1)P(B|1)P(C|1)(1 - P(D|1))P(E|1)P(F|1)P(G|1)P(H|1)$$

计算 $P(c)$

$$P(1) = \frac{5}{8}$$

比较 $\ln P(\text{text}|c)P(c)$

找1-4类中的 argmax

数据集：AgNews

news_category_train_mini.csv
news_category_test_mini.csv

训练及测试用数据(英文)

4类新闻：World Sci/Tech Sports Business

每行的格式形如：类别名|句子内容

nltk_data/:

nltk备用安装包

» 用到的库



第三方: nltk

安装:

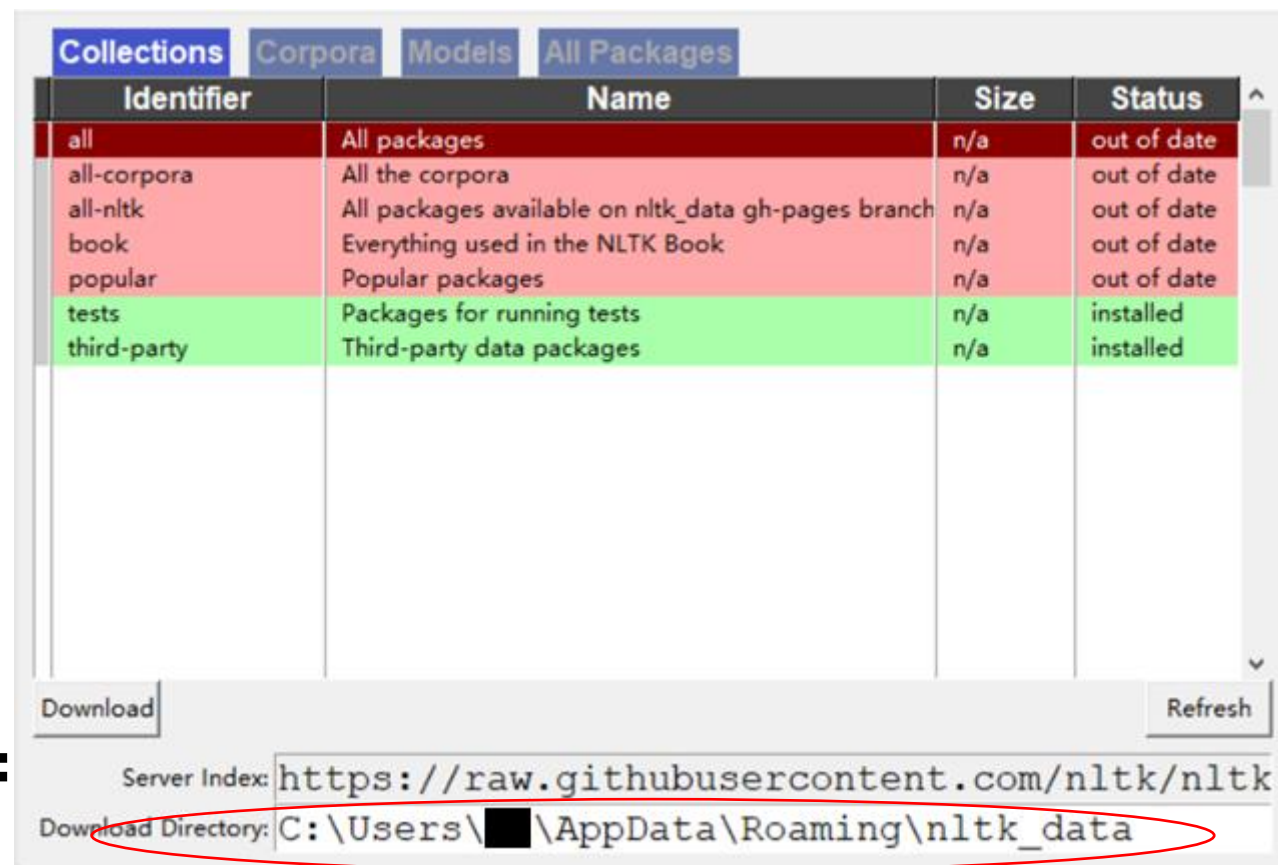
命令行中, 输入 `pip install nltk`

```
import nltk
```

```
nltk.download('punkt')  
nltk.download('stopwords')  
nltk.download('wordnet')  
nltk.download('omw-1.4')
```

如果无法下载, 则在python中输入:
`nltk.download()`

将nltk_data文件包下的tokenizers与corpora文件夹放入弹出的窗口目录下



数据预处理



贝叶斯



测试
计算准确率

统一小写

去标点

去数字

去停用词

词干提取/词形还原



```
# 种类
categories = {'World': 0, 'Sci/Tech': 1, 'Sports': 2, 'Business': 3}
# 还原方法
type_word = ['stemmer', 'lemmatizer'][0]
# 训练方法
type_train = ['TF', 'Bernoulli'][0]
```

» 代码



```
def train_TF(train_x, train_y):  
    dictionary = words2dic(train_x)  
    # 词-类-词频矩阵(维度:词典长度x类别数)  
    words_frequency = np.zeros((len(dictionary), len(categories)), dtype=int)  
    for i, words in enumerate(train_x):  
        category_id = categories[train_y[i]]  
        for word in words:  
            word_id = dictionary[word]  
            words_frequency[word_id][category_id] += 1  
  
    # 每类下的句总数(维度:类别数x1)  
    category_sents = np.zeros(len(categories))  
    for category in train_y:  
        category_id = categories[category]  
        category_sents[category_id] += 1  
    p_c = category_sents / len(train_y)  
  
    # 每类下的词总数(维度:类别数x1)  
    category_words = np.sum(words_frequency, axis=0)  
  
    # 统计矩阵(维度:词典长度x类别数)  
    p_stat = np.zeros((len(dictionary), len(categories)))  
    # TODO: 计算TF方法下的 $P(w_i|c)$ , 即这里的p_stat
```

词典	类别				
	1	2	3	4	
	A	2	1	5	1
	B	4	3	3	0
	C	0	3	4	0
	D	3	0	4	0
	E	2	1	2	3
	F	1	5	0	3
	G	0	4	2	2
	H	6	0	1	1

» 代码 (optional)



```
def train_Bernoulli(train_x, train_y):
    dictionary = words2dic(train_x)
    # 词-类-句总数矩阵(维度:词典长度x类别数)
    sents_count = np.zeros((len(dictionary), len(categories)), dtype=int)
    for i, words in enumerate(train_x):
        category_id = categories[train_y[i]]
        for word in set(words):
            word_id = dictionary[word]
            sents_count[word_id][category_id] += 1

    # 每类下的句总数(维度:类别数x1)
    category_sents = np.zeros(len(categories))
    for category in train_y:
        category_id = categories[category]
        category_sents[category_id] += 1
    p_c = category_sents / len(train_y)

    # 统计矩阵(维度:词典长度x类别数)
    p_stat = np.zeros((len(dictionary), len(categories)))
    # TODO: 计算Bernoulli方法下的 $P(d_?|c)$ , 即这里的p_stat
```

		类别			
		1	2	3	4
词典	A	4	1	0	1
	B	3	0	1	1
	C	2	0	1	0
	D	0	1	0	1
	E	1	0	1	1
	F	5	1	0	1
	G	3	1	1	0
	H	2	0	1	1

```
def test_TF(data_x, data_y, p_stat, dictionary, p_c):  
    """  
    批量数据测试, 计算准确率  
    """  
    # 计算  $\ln P(C)$   
    p_c = np.log(p_c)  
  
    # 分类正确的个数  
    count = 0  
    # 计算P, 即argmax之后的内容  
    for i, words in enumerate(data_x):  
        p = np.zeros(len(categories))  
        # TODO: 计算TF方法下的  $\ln P(\text{text}|c)P(c)$ , 即这里的p  
  
        if np.argmax(p) == categories[data_y[i]]:  
            count += 1  
  
    print('Accuracy: {}/{} {}'.format(count, len(data_y), round(100*count/len(data_y), 2)))
```

- + 未登录词：即只在测试集中出现过，而没有在训练集中出现过的词。可以直接跳过这个词，当它不存在。因为已经训练得到的贝叶斯模型中不含与它相关的知识，这个词对分类没有帮助。
- + 数据集相对比较小，两种还原方法的速度差异并不明显，但在大数据集上的速度差异比较明显。
- + 计算过程中尽量多用矩阵操作，速度较快。



	TF	Bernoulli (optional)
stemmer	9479/10208 92.86% 2226/2552 87.23%	9484/10208 92.91% 2227/2552 87.26%
lemmatizer	9550/10208 93.55% 2226/2552 87.23%	9551/10208 93.56% 2217/2552 86.87%

1. 填空，让程序在2种方法下都可以得出准确率80%以上的结果(Bernoulli选做)
 2. 调整preprocess文本预处理函数，看看部分预处理操作的有/无会对结果造成什么影响
 3. 将上述内容写入实验文档，并附上对应情况下的所得的准确率结果
 4. 思源空间提交：
整个程序的压缩包(可运行)+实验文档(学号_姓名_实验3.pdf)
- + 作业2即将截止，及时提交，互相转告



Thank You Q & A

张浩: 1050852440@qq.com

刘卓: 1zpmbw@163.com