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

2023年4月16日

实验要求

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

背景介绍

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)

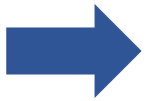
除了停用词，标点符号、数字也可以认为是与分类无关的内容，可将之删去

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

背景介绍

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

例如

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

背景介绍

Term Frequency(Naive Bayes)

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

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

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

对同一text，分母相同，只需要比较分子

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

用频率去逼近概率，即这类文档数量/所有文档数量

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

Term Frequency(Naive Bayes)

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

朴素贝叶斯：
各特征之间相互独立

$$P(text|c) = \prod_{i=1}^{n} P(w_i|c), w_i \in D$$

w_i一共会有m种可能，$d_1 \sim d_m$
显然有$\sum_{?=1}^{m} P(d_?|c) = 1$

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)

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

拉普拉斯平滑：
实际使用时
通常令$P(w_i|c) = \frac{N(w_i\ in\ W_c)+1}{N(W_c)+m}$
既能防止$P(w_i|c) = 0$
又能保持$\sum_{?=1}^{m} P(d_?|c) = 1$

程序中连乘容易趋向于0，于是取对数

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

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

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

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

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

对同一text，分母相同，只需要比较分子

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

即这类文档数量/所有文档数量

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

Bernoulli(Binary) （Optional）

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

即这类文档中出现该词的文档个数/这类文档的总个数

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

拉普拉斯平滑：

实际使用时

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

$$c_{text} = \text{argmax}_{c \in C} \prod_{?=1}^{m} P(d_?|c)^b \left(1 - P(d_?|c)\right)^{1-b} P(c)$$

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

背景介绍

4类文本，从C_1-C_4
文本库中有4个句子，从①到④，分别是C_1-C_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：

	C_1	C_2	C_3	C_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C_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)

$$P(A|C_1), P(B|C_1), \ldots = \frac{2+1}{18+8}, \frac{4+1}{18+8}, \ldots$$

计算P(text|c)

$$P(①|C_1) = P^2(A|C_1)P^4(B|C_1) \ldots P^6(H|C_1)$$

计算P(c)

$$P(C_1) = \frac{1}{4}$$

比较ln P(text|c)P(c)

$$\ln P(①|C_1)\,P(C_1) = 2\ln P(A|C_1) + \cdots + 6\ln P(H|C_1) + \ln P(C_1)$$

找C_1-C_4类中的argmax

背景介绍

4类文本，从C_1-C_4
文本库中有8个句子，从①到⑤都是C_1类的句子，⑥⑦⑧分别是C_2-C_4类的句子
词汇表中有8个词，从A到H

Bernoulli方法：
统计如下：

	C_1	C_2	C_3	C_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C_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C_1为例：
平滑计算P(d|c)

$$P(A|C_1), P(B|C_1), \ldots = \frac{4+1}{5+2}, \frac{3+1}{5+2}, \cdots$$

假设句①中有ABCEFGH这些词
计算P(text|c)

$$P(①|C_1) = P(A|C_1)P(B|C_1)P(C|C_1)\big(1 - P(D|C_1)\big)P(E|C_1)P(F|C_1)P(G|C_1)P(H|C_1)$$

计算P(c)

$$P(C_1) = \frac{5}{8}$$

比较ln P(text|c)P(c)
找1-4类中的argmax

实验数据

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

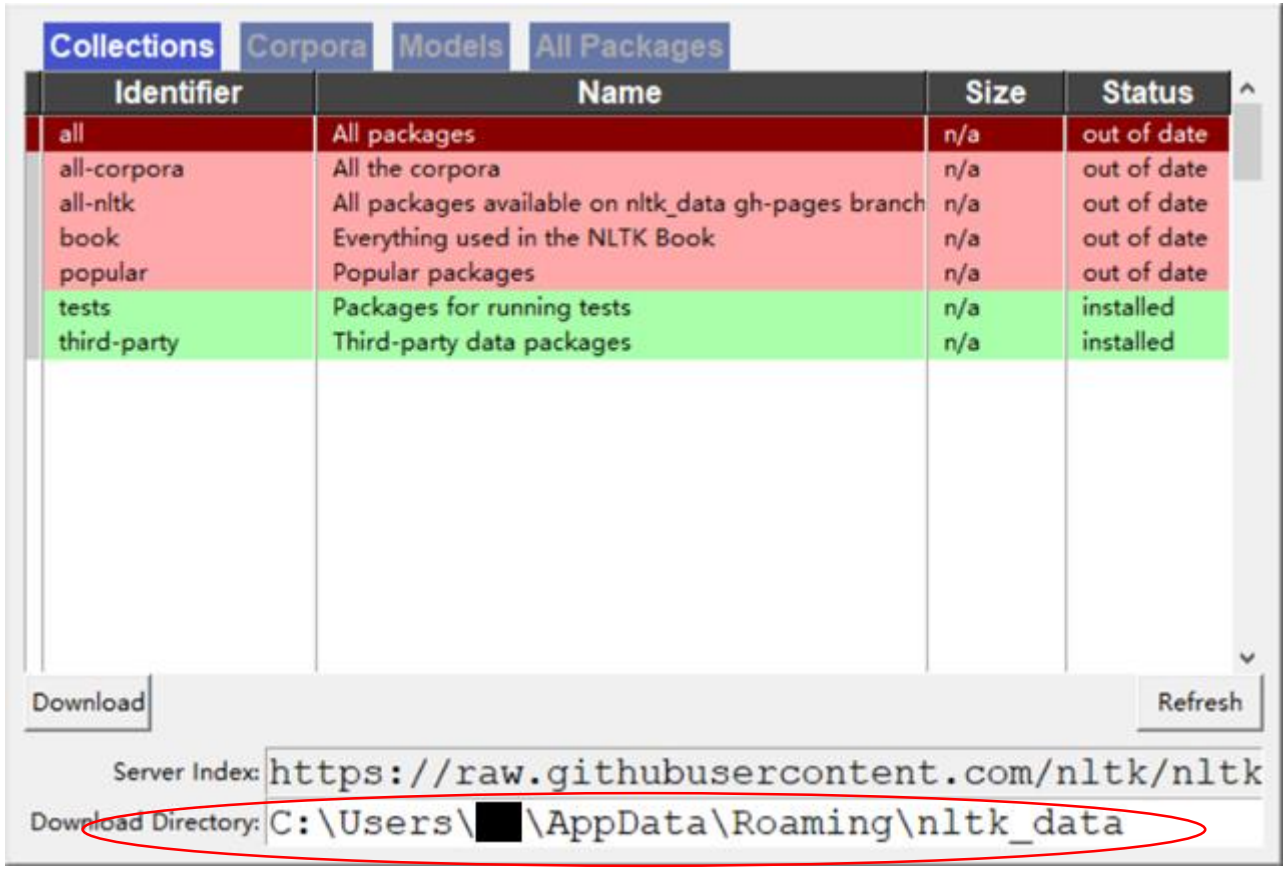
西安交通大学 XI'AN JIAOTONG UNIVERSITY

第三方：nltk

安装：
命令行中，输入pip install nltk

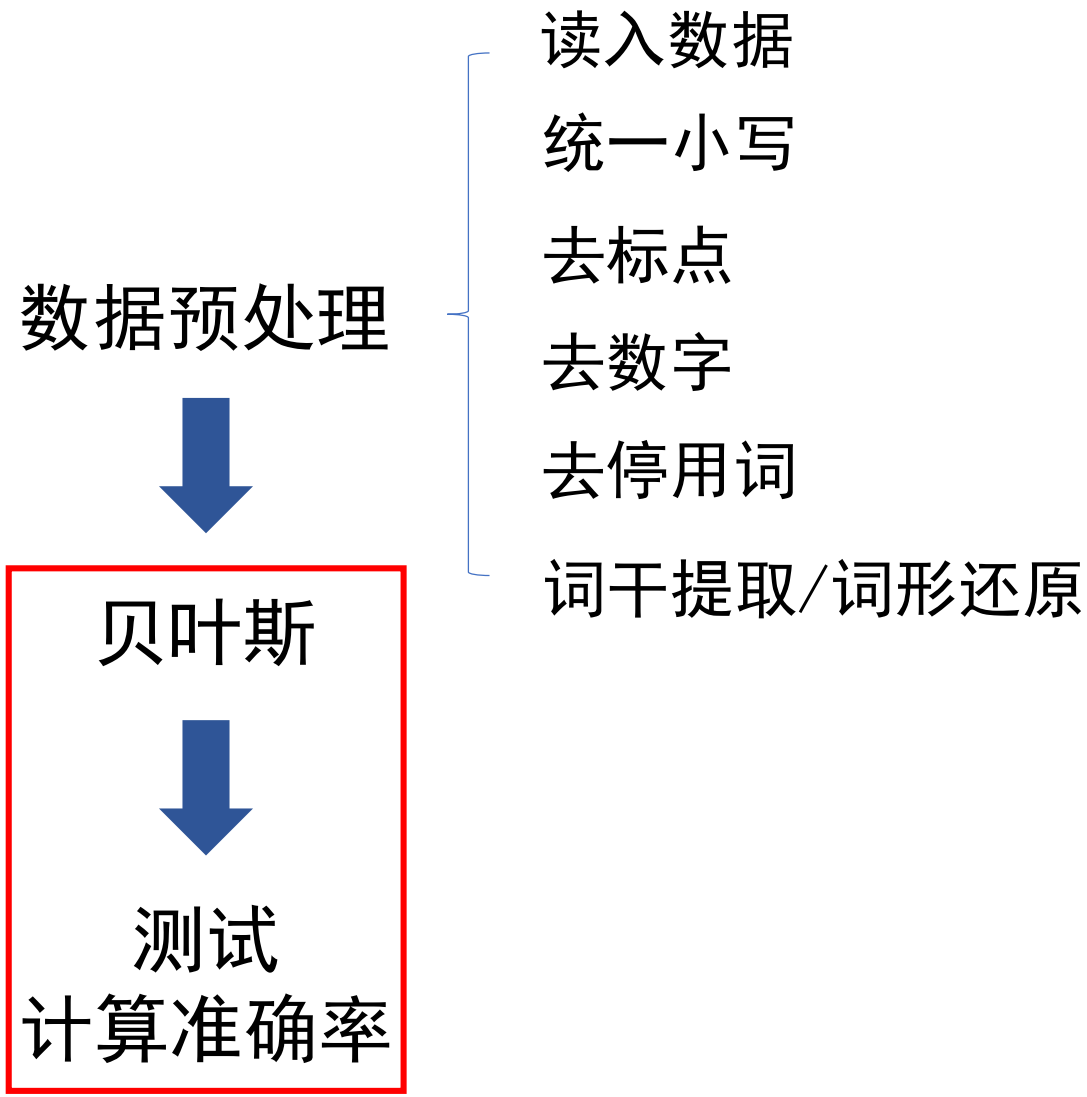
```
import nltk


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

实验流程



代码


```
# 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 load(path, type_word):
    data_x, data_y = [], []
    with open(path, 'r') as f:
        lines = f.readlines()
        length = len(lines)
        for i, line in enumerate(lines):
            tmp = line.split('|')
            data_x.append(preprocess(tmp[1].strip(), type_word))
            data_y.append(tmp[0])
            if i % 1000 == 0:
                print('loading:{}/{}'.format(i, length))

    return data_x, data_y
```

代码


```
def preprocess(sent, type_word):
    # 统一为小写
    sent = sent.lower()
    # 去标点
    remove = str.maketrans('', '', string.punctuation)
    sent = sent.translate(remove)
    # 转化为单词词组
    words = nltk.word_tokenize(sent)
    # 去停用词
    words = [w for w in words if not (w in stopwords)]
    # 去数字
    words = [w for w in words if not w.isdigit()]
    # 还原:词干提取/词形还原
    if type_word == 'stemmer':
        words = [stemmer.stem(w) for w in words]

    elif type_word == 'lemmatizer':
        words = [lemmatizer.lemmatize(w) for w in words]

    return words
```

代码


```
def train_TF(train_x, train_y):
    # 词汇表
    dictionary = words2dic(train_x)

    # n(w_i in w_c) 词-类-词频矩阵(维度:词汇数x类别数)
    words_frequency = np.zeros((len(dictionary),
len(categories)), dtype=int)

    # n(c,text) 每类下的句总数(维度:类别数x1)
    category_sents = np.zeros(len(categories), dtype=int)

    # p(c) (维度:类别数x1)
    p_c = np.zeros(len(categories), dtype=int)

    # n(w_c) 每类下的词总数(维度:类别数x1)
    category_words = np.zeros(len(categories), dtype=int)

    # p(w_i|c) (维度:词汇数x类别数)
    p_stat = np.zeros((len(dictionary), len(categories)))

    return p_stat, dictionary, p_c
```

代码


```
def test(data_x, data_y, p_stat, dictionary, p_c, type_train):
    # 统计预测正确的数目
    count = 0
    # 计算argmax(...)
    if type_train == 'TF':
        for i, words in enumerate(data_x):
            if np.argmax() == categories[data_y[i]]:
                count += 1

    elif type_train == 'Bernoulli':
        pass

    print('Accuracy: {}/{} {}%'.format(count, len(data_y),
round(100*count/len(data_y), 2)))
```

提示

- 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。

运行结果

	TF	Bernoulli（optional）
stemmer	9479/10208 92.86% 2226/2552 87.26%	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%

实验要求

1. 代码补充完整(Bernoulli方法选做)

2. 调整预处理函数，看看部分预处理操作的有/无对结果有什么影响

3. 实验文档内容需要包括：
   实验原理
   代码及对应简要说明
   实验结果(准确率)

Thank You Q & A

张浩：1050852440@qq.com

刘卓：lzpmbw@163.com