基于朴素贝叶斯算法的垃圾邮件分类器

梁鑫嵘 200110619
计算机科学与技术学院

1 理论基础

1.1 贝叶斯公式

贝叶斯公式刻画了先验概率和后验概率之间的关系。

设$A_1, A_2, \ldots, A_n$构成完备事件组，且对于$\forall A_i, P(A_i) > 0$。B为样本空间的任意事件，$P(B) > 0$，则有：[1]

$$P(A_k|B) = \frac{P(A_k)P(B|A_k)}{\sum_{i=1}^{n} P(A_i)P(B|A_i)}$$

1.2 贝叶斯公式在问题中的应用

对于贝叶斯公式我们可以有如下的理解[2]：

$$P(\text{类别}|\text{特征}) = \frac{P(\text{特征}|\text{类别})P(\text{类别})}{P(\text{特征})}$$

当我们求得P(类别|特征)，再将本样本归于概率最大的那一类，实际就完成了一次贝叶斯分类过程。

但是上述对贝叶斯公式的理解还需要加上一个不可缺少的条件，即各变量之间相互独立，以满足“$A_1, A_2, \ldots, A_n$为完备事件组”这一条件。所以在实际运用中，我们还需要假设各个特征之间是独立的。

然而实际上，能够统计到的变量之间几乎不可能是互相独立的。例如，设事件$A =$ 电子邮件中出现“http”,$B =$ 电子邮件中包含完整链接，当A, B都发生的时候电子邮件很可能是一封垃圾广告邮件。但是，显然$B \subseteq A$，也就是说A, B不互相独立。我们的程序在处理的过程中很难依据内容判断词与词之间的复杂的语义、逻辑关系，即无法判断两个词的出现事件之间是否互相独立。此外，由于人类语言的复杂性，词语之间基本都存在互相关联的情况，无法直接地抽象为互斥事件，所以利用朴素贝叶斯算法进行基于文本的垃圾邮件判断存在一定的理论误差。

在不大规模使用的情况下，朴素贝叶斯算法仍然能在简单的邮件分类作业上表现良好。

假设A_i为出现词汇表中的第i个词，B为此邮件为垃圾邮件，
$\because A_i, A_j$互相独立$(i \neq j)$，则：

$$
\begin{aligned}
P(B|A_1,A_2,\cdots,A_n) &= \frac{P(A_1|B)P(A_2|B)\cdots P(A_n|B)P(B)}{P(A_1)P(A_2)\cdots P(A_n)} \\
&= \frac{P(B)\Pi_{i=1}^{n}P(A_i|B)}{\Pi_{i=1}^{n}P(A_i)}
\end{aligned}
$$

2 实践

2.1 要点问题分析

2.1.1 如何用数据表示词语

在程序处理的过程中，我们只需要进行数字的计算，但是我们的日常语言是使用词语等拼接起来的，为了进行这其中的转换，使得程序能够用数字的方式正确“理解”词语，我们需要一种用数字正确表示词语的方法。

常用的方法有独热码、词向量等，这里使用独热码表示。

在独热码表示法中，需要使用的数据在向量中对应位置为1，其他数据在向量中的位置为0。对应在这个分类问题中，词汇表为训练数据中所有出现的词语，向量长度为词汇表长度，每一个词语对应的独热码为$[0,0,\ldots,1,\ldots,0]$。

2.1.2 正确处理“0概率”情况

观察贝叶斯公式，分母为$\Pi_{i=1}^{n}P(A_i)$，但是如果分母为0，计算出的概率是无意义的。为了解决这样的问题，我们需要保证分母不为0，为此我们需要使用拉普拉斯平滑法处理分母数据。

拉普拉斯平滑，又称为加1平滑，是一种常用的平滑方法，经常在解决0概率问题中有所应用[3]。

我们令$\alpha>0$，当$\alpha\to 0$时，$\Pi_{i=1}^{n}P(A_i)\approx\Pi_{i=1}^{n}P(A_i)+\alpha$。

此时分母已经不为0，但是仍然存在一些问题，计算得到的$P(B|A_1,A_2,\cdots,A_n)$可能因为分母过小而过大，所以最好对其加上一个对数函数，防止其因为计算数字过大而溢出。因为使用了对数函数，要求参数>0，因此我们可以对分子$P(B)\Pi_{i=1}^{n}P(A_i|B)$也加上$\alpha$。最终得到的公式表示如下：

$$
\begin{aligned}
Q_B &= \ln(P(B|A_1,A_2,\cdots,A_n)) \\
&= \ln(\frac{P(B)\Pi_{i=1}^{n}P(A_i|B)}{\Pi_{i=1}^{n}P(A_i)}) \\
&\approx \ln P(B)+\sum_{i=1}^{n}\ln(\frac{P(A_i|B)}{P(A_i)+\alpha_2}+\alpha_1) && (\alpha_1,\alpha_2>0) \\
&\approx \ln P(B)+\sum_{i-1}^{n}\ln(P(A_i|B)+\alpha_1)-\sum_{i-1}^{n}\ln(P(A_i)+\alpha_2)
\end{aligned}
$$

2.2 程序运行流程

2.2.1 数据处理

```
def split_text(text: str):
    # 将特殊符号作为切分标志进行字符串切分，即非字母、非数字
    tokens = re.split(r'\W+', text)
    # 除了单个字母，例如大写的I，其它单词变成小写；并且排除纯数字
    return [tok.lower() for tok in tokens if len(tok) > 2 and not
tok.isdigit()]


def get_words_data():
    """
    获取训练用文本数据，并且格式化分割到单词
    :return: 得到的文本数据，分类列表
    """
    class_types = [r for r in os.listdir('email') if
os.path.isdir(os.path.join('email', r))]

    def read_data(filename_: str) -> str:
        with open(filename_, 'r', encoding='gbk') as f:
            return f.read()

    words_data = []
    for c in class_types:
        for filename in os.listdir(os.path.join('email', c)):
            file_data = read_data(os.path.join(f'email/{c}', filename))
            words_data.append((c, split_text(file_data)))
    return words_data, class_types


def get_words_label(words_data: list) -> list:
    """
    得到当前数据集下的词汇表
    :param words_data: 读取到的词语数据
    :return: 词汇表
    """
    # 使用 set 去重
```

```
    words_label = set({})
    for words in words_data:
        words_label.update(words[1])
    res = list(words_label)
    res.sort()
    return res


def get_words_vector(words, words_label: list) -> list:
    """
    得到一个词语或者一个词语列表对应的词向量
    :param words: 词语或者词语列表
    :param words_label: 词汇表
    :return: 词向量
    """
    return [(1 if val == words else 0) if not isinstance(words, list)
            else (1 if val in words else 0) for val in words_label]
```

2.2.2 训练分类器

```
def native_bayes_train(words_label: list, train_data: list,
class_types: list, alpha: float = 1e-3):
    """
    训练朴素贝叶斯分类器
    :param words_label: 词汇表
    :param train_data: 训练数据集
    :param class_types: 分类列表
    :param alpha: > 0 且很小的数，用于拉普拉斯平滑
    :return: 训练结果，各种类的概率
    """
    # 初始化就加上这个 alpha，防止出现 Nan。
    p_result = {c: np.array([alpha for _ in range(len(words_label))])
for c in class_types}
    p_b = {c: alpha for c in class_types}
    for data in train_data:
        # 得到词向量
        words_vector = np.array(get_words_vector(data[1], words_label))
        # 记录到训练数据
        p_result[data[0]] += words_vector + alpha
```

```
        # 记录到对应分类的概率
        p_b[data[0]] += 1 / len(train_data)
    for k in p_result:
        p_result[k] = np.log(p_result[k])
    return p_result, p_b
```

2.2.3 测试分类器

```
def native_bayes_test(words_label: list, p_result: dict, test_data:
list, p_b: dict, alpha: float = 1e-3) -> float:
    """
    测试朴素贝叶斯分类器
    :param words_label: 词汇表
    :param p_result: 训练结果
    :param test_data: 测试数据集
    :param p_b: 各种类的概率
    :param alpha: > 0 且很小的数，用于拉普拉斯平滑
    :return: 测试正确的概率
    """
    accurate = 0
    for data in test_data:
        # 获得词向量
        words_vector = np.array(get_words_vector(data[1], words_label))
        # 统计相对概率
        Qs = {key: np.log(p_b[key]) + sum(p_result[key] * words_vector)
- np.log(sum(words_vector + alpha)) for key in
              p_result}
        # 选取相对概率最高的那个作为分类结果
        classification = reduce((lambda x, y: x if x[1] > y[1] else y),
((k, Qs[k]) for k in Qs))[0]
        print(f"result {classification == data[0]}, classification =
{classification}, label = {data[0]}; Qs: {Qs}")
        # 记录正确率
        accurate += (1 if classification == data[0] else 0) /
len(test_data)
    return accurate
```

2.2.4 运行主程序

```python
def main():
    # 读取训练、测试数据
    words_data, class_types = get_words_data()
    # 打乱训练数据
    random.shuffle(words_data)
    # 生成词汇表
    words_label = get_words_label(words_data)
    # 训练朴素贝叶斯分类器，使用随机40份邮件做训练数据集
    p_result, p_b = native_bayes_train(words_label, words_data[:40],
class_types)
    # 测试朴素贝叶斯分类器，使用剩下的邮件做测试数据集
    accurate = native_bayes_test(words_label, p_result,
words_data[40:], p_b)
    print(f"DONE! accurate = {(accurate * 100):.2f}%")
```

完整程序见附录。

2.3 程序运行结果

```
result True, classification = ham, label = ham; Qs: {'ham':
-59.347845111308025, 'spam': -85.7843006553319}
result True, classification = spam, label = spam; Qs: {'ham':
-36.34975776983966, 'spam': 5.109566958100539}
result True, classification = ham, label = ham; Qs: {'ham':
-92.15951999754893, 'spam': -184.30347898520958}
result True, classification = spam, label = spam; Qs: {'ham':
-57.07097136557958, 'spam': -53.66419688826986}
result True, classification = spam, label = spam; Qs: {'ham':
-90.63426772205429, 'spam': 30.369671882623937}
result True, classification = spam, label = spam; Qs: {'ham':
-90.63426772205429, 'spam': 30.369671882623937}
result True, classification = spam, label = spam; Qs: {'ham':
-28.69504741287626, 'spam': -16.289914553986566}
result True, classification = spam, label = spam; Qs: {'ham':
-111.28152808527354, 'spam': 32.34635005125606}
result True, classification = spam, label = spam; Qs: {'ham':
-105.0581659869714, 'spam': 30.966905653462643}
result True, classification = spam, label = spam; Qs: {'ham':
-90.63426772205429, 'spam': 30.369671882623937}
DONE! accurate = 100.00%
```

程序判断正确率可以达到90%以上。

3 总结

本论文基于概率论与数理统计课程中的贝叶斯分类器部分的理论知识，完成了一个基于朴素贝叶斯算法的垃圾邮件分类器，展示了概率论这一学科的丰富的运用场景。

4 附录

4.1 完整程序

```
import os
import re
import numpy as np
from functools import reduce
import random


def split_text(text: str):
    # 将特殊符号作为切分标志进行字符串切分，即非字母、非数字
    tokens = re.split(r'\W+', text)
    # 除了单个字母，例如大写的I，其它单词变成小写；并且排除纯数字
    return [tok.lower() for tok in tokens if len(tok) > 2 and not
tok.isdigit()]


def get_words_data():
    """
    获取训练用文本数据，并且格式化分割到单词
    :return: 得到的文本数据，分类列表
    """
    class_types = [r for r in os.listdir('email') if
os.path.isdir(os.path.join('email', r))]

    def read_data(filename_: str) -> str:
        with open(filename_, 'r', encoding='gbk') as f:
            return f.read()

    words_data = []
    for c in class_types:
```

```
        for filename in os.listdir(os.path.join('email', c)):
            file_data = read_data(os.path.join(f'email/{c}',
filename))
            words_data.append((c, split_text(file_data)))
    return words_data, class_types


def get_words_label(words_data: list) -> list:
    """
    得到当前数据集下的词汇表
    :param words_data: 读取到的词语数据
    :return: 词汇表
    """
    # 使用 set 去重
    words_label = set({})
    for words in words_data:
        words_label.update(words[1])
    res = list(words_label)
    res.sort()
    return res


def get_words_vector(words, words_label: list) -> list:
    """
    得到一个词语或者一个词语列表对应的词向量
    :param words: 词语或者词语列表
    :param words_label: 词汇表
    :return: 词向量
    """
    return [(1 if val == words else 0) if not isinstance(words, list)
            else (1 if val in words else 0) for val in words_label]


def native_bayes_train(words_label: list, train_data: list,
class_types: list, alpha: float = 1e-3):
    """
    训练朴素贝叶斯分类器
    :param words_label: 词汇表
```

```
    :param train_data: 训练数据集
    :param class_types: 分类列表
    :param alpha: > 0 且很小的数，用于拉普拉斯平滑
    :return: 训练结果，各种类的概率
    """
    # 初始化就加上这个 alpha，防止出现 Nan。
    p_result = {c: np.array([alpha for _ in range(len(words_label))])
for c in class_types}
    p_b = {c: alpha for c in class_types}
    for data in train_data:
        # 得到词向量
        words_vector = np.array(get_words_vector(data[1],
words_label))
        # 记录到训练数据
        p_result[data[0]] += words_vector + alpha
        # 记录到对应分类的概率
        p_b[data[0]] += 1 / len(train_data)
    for k in p_result:
        p_result[k] = np.log(p_result[k])
    return p_result, p_b


def native_bayes_test(words_label: list, p_result: dict, test_data:
list, p_b: dict, alpha: float = 1e-3) -> float:
    """
    测试朴素贝叶斯分类器
    :param words_label: 词汇表
    :param p_result: 训练结果
    :param test_data: 测试数据集
    :param p_b: 各种类的概率
    :param alpha: > 0 且很小的数，用于拉普拉斯平滑
    :return: 测试正确的概率
    """
    accurate = 0
    for data in test_data:
        # 获得词向量
        words_vector = np.array(get_words_vector(data[1],
words_label))
```

```
        # 统计相对概率
        Qs = {key: np.log(p_b[key]) + sum(p_result[key] *
words_vector) - np.log(sum(words_vector + alpha)) for key in
              p_result}
        # 选取相对概率最高的那个作为分类结果
        classification = reduce((lambda x, y: x if x[1] > y[1] else
y), ((k, Qs[k]) for k in Qs))[0]
        print(f"result {classification == data[0]}, classification =
{classification}, label = {data[0]}; Qs: {Qs}")
        # 记录正确率
        accurate += (1 if classification == data[0] else 0) /
len(test_data)
    return accurate


def main():
    # 读取训练、测试数据
    words_data, class_types = get_words_data()
    # 打乱训练数据
    random.shuffle(words_data)
    # 生成词汇表
    words_label = get_words_label(words_data)
    # 训练朴素贝叶斯分类器，使用随机40份邮件做训练数据集
    p_result, p_b = native_bayes_train(words_label, words_data[:40],
class_types)
    # 测试朴素贝叶斯分类器，使用剩下的邮件做测试数据集
    accurate = native_bayes_test(words_label, p_result,
words_data[40:], p_b)
    print(f"DONE! accurate = {(accurate * 100):.2f}%")


if __name__ == '__main__':
    main()
```

运行目录结构：

```
|  bayes.py
└─ email/
    ├─ham
    |      *.txt
    └─spam
            *.txt
```

4.2 数据来源

https://github.com/Asia-Lee/Naive_Bayes

---

1. 王勇, 田波平. 概率论与数理统计[M]. 第二版. 北京: 科学出版社, 2005. ↩

2. 忆臻.[EB/OL]. https://zhuanlan.zhihu.com/p/26262151. 2017年04月10日-2021年11月23日. ↩

3. 郑万通 .[EB/OL]. https://blog.csdn.net/zhengwantong/article/details/72403808. 2017年05月17日-2021年11月23日. ↩