

華東理工大學

模式识别大作业

题 目	基于朴素贝叶斯的垃圾短信分类
学 院	信息科学与工程
专 业	控制科学与工程
组 员	王峰
指导教师	赵海涛

完成日期： 2018 年 10 月 25 日

基于朴素贝叶斯的垃圾短信分类

一、朴素贝叶斯

1. 贝叶斯定理

在概率论与统计学中，贝叶斯定理（或称贝叶斯法则、贝叶斯规则）描述了一个事件的可能性，这个可能性是基于了预先对于一些与该事件相关的情况的知识。举例来说，如果癌症和年龄有关，那么使用贝叶斯定理的话，相比根本不了解关于此人的任何其他信息，知道了它的年龄的话就可以用来更准确地帮助评估它得癌症与否的概率。

那么其实很明显了，这里的“可能性”也是考虑了与随机事件相关的因素的，所以贝叶斯定理所阐述的也就是后验概率的获得方法。

用数学公式来表述贝叶斯定理：

$$P(c|x) = \frac{p(x|c)P(c)}{p(x)} \quad (1-1)$$

c 表示的是随机事件发生的一种情况。 x 表示的就是证据（evidence），泛指与随机事件相关的因素。

$P(c|x)$: 在 x 的条件下，随机事件出现 c 情况的概率。（后验概率）

$p(x)$: x 出现的概率

$P(x|c)$: 在已知事件出现 c 情况的条件下，条件 x 出现的概率。（似然函数）

$P(c)$: （不考虑相关因素）随机事件出现 c 情况的概率（先验概率）

2. 朴素贝叶斯简介

上面讲到，为了获得 $P(c|x)$ ，我们的计算任务最终落脚到了获得 $p(x|c)$ 和 $P(c)$ 上。假设我们有了数据集 D ，那么 $p(c)$ 的获得其实也较为简单：计算 D 中 c 的各个情况出现的频率即可。比如计算 $P(c_1)$ ，直接用 c_1 情况出现次数在所有情况中所占的比例值即可。（这里用到了大数定律：当训练集包含充足的独立同分布样本时， $P(c)$ 可通过各类样本出现的频率来进行估计。）而获得 $p(x|c)$ 就略显困难，因为 x 往往包含多个相关因素（是一个多种因素构成的向量），即它可能有多个需要考虑的属性值： $x=(x_1, x_2, x_3, \dots, x_n)$ ，其中任一 x_i 都代表了所有相关因素中的其中一个。在癌症辅助判断中，它可能是患者的年龄，也可能是患者的性别，也可能是患者是否吸烟等等。因此当 x 是一个向量时，我们若要计算 $p(x|c)$ ，实际上就是要计算 $p(x_1, x_2, x_3, \dots, x_n | c)$ 。这个理论上也是可以利用我们的数据集 D 来进行估计的，但是现实情况是， n 的值往往非常大（特征属性非常多），而我们的数据集往往不能保证我们的样本包含了属性值的所有可能组合（假设每个属性都是二值属性，那么就有 2^n 种属性组合）。那么很多 $p(x|c)$ 我们估计得到的值就是 0。然而这些样本很可能仅仅是我们的数据集中没包含到，即“未被观测到”，但不代表它们现实中“出现概率为 0”。于是这就给我们计算出真实合理的目标 $P(c|x)$ 值造成了障碍。

于是，朴素贝叶斯的“朴素”就发挥作用了。朴素贝叶斯方使用一个非常简单的方式来处理似然函数 $p(x_1, x_2, x_3, \dots, x_n | c)$ 估计的问题。假设 $(x_1, x_2, x_3, \dots, x_n)$ 各特征之间相互独立，即：

$$p(x_1, x_2, x_3, \dots, x_n | c) = \prod_{i=1}^n p(x^i | c) \quad (1-2)$$

这个假设认为每个属性取它的各个值的可能性是独立的，与其它属性的取值不相关。

二、朴素贝叶斯在垃圾邮件分类中的应用

1. 方案设计

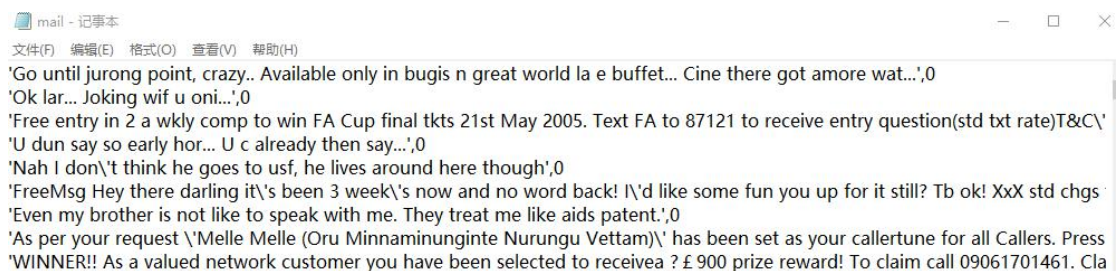
我们手上有一个大约含有 5574 条短信的数据，其中包括相应数量的垃圾邮件和正常邮件。首先读取所有的邮件，将数据处理成我们所需要的类型。接着训练算法，先确定所有的特征以及特征数量，并计算不同特征的条件概率。随后随机抽取训练集中 1000 条短信进行测试，计算测试的正确率。最后使用该算法，应用到普通邮件分类中去。

在数据处理过程中，我们使用所有邮件中出现过的单词的数量作为特征数量。并根据朴素贝叶斯定理，假设每个特征之间是相互独立的，即一个单词的出现的可能性与其他单词相邻没有关系。

2. python 具体实现步骤

(1) 读取数据

本次所读取的数据格式为 TXT 格式，每一行前一部分为短信内容，后一部分为标签，0 代表正常邮件，1 代表垃圾邮件。如下图所示



```
mail - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
'Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...',0
'Ok lar... Joking wif u oni...',0
'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std txt rate)T&C\
'U dun say so early hor... U c already then say...',0
'Nah I don't think he goes to usf, he lives around here though',0
'FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like some fun you up for it still? Tb ok! XxX std chgs
'Even my brother is not like to speak with me. They treat me like aids patent.',0
'As per your request '\Melle Melle (Oru Minnaminunginte Nurungu Vettam)\' has been set as your callertune for all Callers. Press
'WINNER!! As a valued network customer you have been selected to receive a ? £ 900 prize reward! To claim call 09061701461. Cla
```

代码如下：

```
def textparse(string):
    import re
    listwords=re.split(r" |\!|\?|\.|\\, |\t|\\;|\\*|\\'|\\\"|\\-", string)
    return [word.lower() for word in listwords if len(word)>2] #只取具
    有价值的单词，所以长度大于 2
def readtxt():
    text=open('C:\mail.txt',encoding='gb18030',errors='ignore')
```

```

arraylines=text.readlines()
doclist=[]
classlist=[]
for line in arraylines:
    doclist.append(textparse(line[:-3]))
    try:
        classlist.append(int(line[-2]))
    except:
        classlist.append(0)
return doclist,classlist

```

准备数据过程中要切分文本，本程序中从每一个短信文本中提取字符串长度大于二的单词，并将其转为小写。每一行短信经过处理后存入列表中，同时将标签存入另一列表。读取过程中有两个短信标签格式是非法的，因大部分短信为正常短信，故将其设置为 0。

(2) 构建短信词向量

代码如下：

#创建一个包含所有短信单词的单词表

```

def Creatvocablist(datalist):
    vocabset=set([])
    for x in datalist:
        vocabset=vocabset|set(x)
    return list(vocabset)

```

#创建一个只含 1 或 0 的邮件向量

```

def Wordstovector(vocablist,inputset):
    returnvec=[0]*len(vocablist)
    for word in inputset:
        returnvec[vocablist.index(word)]+=1
    return returnvec

```

函数 Creatvocablist() 创建一个包含在所有短信文本出现的不重复单词的列表。函数 Wordstovector() 是将输入的每一条短信转换为文档向量，向量中 1 或 0 代表对应的单词是否出现，但是上述这种简单地构建向量的方式，会导致每个词的重要过于均衡，无法体现词向量中词汇的重要性，导致分类结果不理想。

改进中，可以采用 TF_IDF 方式来构建词向量。后面将比较这种方式的改进程度。

(3) 训练算法

代码如下：

```

def trainNB(trainmat,trainlabel):
    numTraindocs=len(trainmat)
    numwords=len(trainmat[0])
    prel=sum(trainlabel)/float(numTraindocs) #类别为垃圾短信的先验概率
    p0=ones(numwords)*0.0001 #初始化正常短信对应的各个特征数量

```

```

p1=ones(numwords)*0.0001#初始化垃圾短信对应的各个特征数量
sump0=0 #正常短信的词数
sump1=0 #垃圾短信的词数
for i in range(numTraindocs):
    if trainlabel[i]==0:
        p0+=trainmat[i,:]
        sump0+=sum(trainmat[i,:])
    else:
        p1+=trainmat[i,:]
        sump1+=sum(trainmat[i,:])
p0vect=log(p0/sump0)#列表里每一项为各个词(特征)对应的条件概率(正常邮件)
p1vect=log(p1/sump1)
return p0vect, p1vect, prel

```

#由后验概率判断短信属于哪一类

```

def classifyNB(vec, p0vect, p1vect, prel):
    post0=sum(vec*p0vect)+log(1-prel)#属于正常短信的后验概率
    post1=sum(vec*p1vect)+log(prel)#属于垃圾短信的后验概率
    if post1>post0:
        return 1
    else:
        return 0

```

trainNB() 函数输入的参数为文档矩阵和文档标签向量。首先计算计算短信属于垃圾短信的概率 $p(c1)$ ，由 $1-p(c1)$ 得到了属于正常短信的概率。接着在每一类下计算似然函数 $p(x|c)$ ，这一过程需要初始化分子和分母。最后相乘得到每一类的后验概率。

(4) 测试算法

代码如下：

```

def test():
    doclist, classlist=readtxt()
    vocablist=Creatvocablist(doclist)
    lendoc=len(doclist)
    trainset=range(lendoc)
    testset=[]
    testnum=1000
    for i in range(testnum):
        randindex=int(random.uniform(0, lendoc))
        testset.append(trainset[randindex])
    trainmat1=[]
    for j in range(lendoc):
        trainmat1.append(Wordstovector(vocablist, doclist[j]))
    p0, p1, prel=trainNB(array(trainmat1), array(classlist))

```

```

rightcount=0
for i in testset:
    wordvector=Wordstovector(vocablist,doclist[i])
    if classifyNB(array(wordvector),p0,p1,pre1)==classlist[i]:
        rightcount+=1
return rightcount/testnum

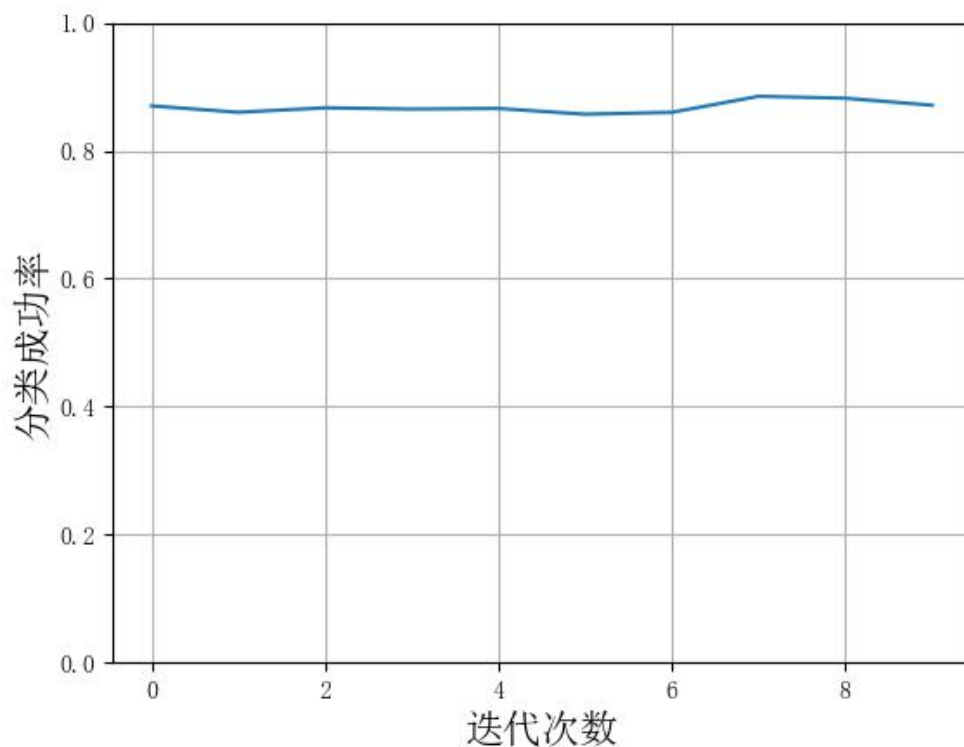
rate=[]
for i in range(10):
    rate.append(test())
print(rate)

```

测试集是从训练集中随机抽取的 1000 条短信, 为了更精确的估计分类器的正确率, 我们迭代了十次观察正确率的变化情况, 结果如下:

[0.87, 0.86, 0.867, 0.865, 0.866, 0.857, 0.86, 0.885, 0.882, 0.871]

正确率变化如图所示:



分类的平均正确率大概为 0.87。

使用改进的方法 **TF-IDF** 方式来构建词向量时, 代码如下:

#TF-IDF 方式来构建词向量。

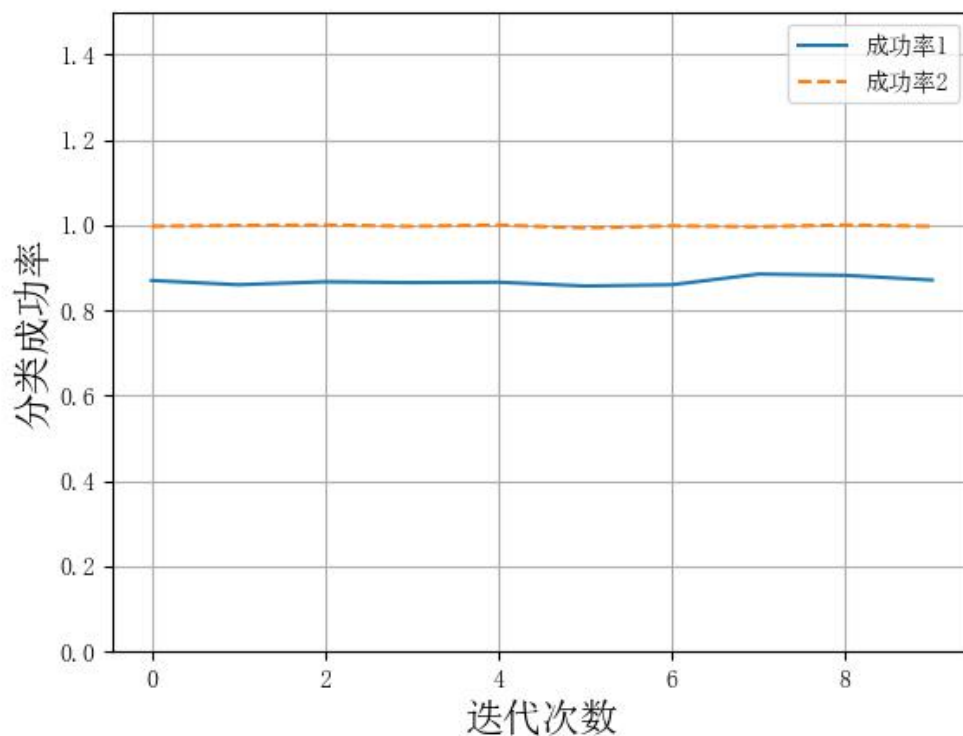
```
def creatif_idf(traindata):  
    tfidf = TfidfTransformer()  
    set_printoptions(2)  
    trainmat=tfidf.fit_transform(traindata).toarray()  
    return trainmat
```

迭代十次的结果如下图所示：

```
[0.997, 0.999, 1.0, 0.997, 1.0, 0.993, 0.998, 0.996, 1.0, 0.997]
```

分类的平均正确率接近于 1。

两种方法的成功率对比如下：



其中成功率 1 为普通朴素贝叶斯分类器分类的结果，成功率 2 为使用 TF-IDF 方式构建词向量的朴素贝叶斯分类器分类的结果。可以看出结果有了很大的优化。

三、总结

这次的大作业做起来很不容易，但最终还是完成了。过程很充实，增加了我对模式识别的兴趣，感谢赵老师的认真教学，谢谢老师对我们的指导。

