# 朴素贝叶斯算法

# 1朴素贝叶斯算法

### 1.1 垃圾邮件处理模型

以预测垃圾邮件为例子，介绍贝叶斯算法。词库中有5000个单词。我们有输入x，，是一个50000维的向量，其中表示词库中第i个单次在该邮件中出现，为0表示词库中第i个单次在该邮件中没有出现。

已经邮件是否为邮件,样本出现的概率为，可以定义为下式:



注: 上式很好理解。右侧第一个式子为已知y，x1出现的概率。第二个为一直y和x1出现x2的概率。因此两个式子相乘表示一直y初选(x1,x2)的概率。依次类推，可以得到这个公式。

接下来我们做个假设，即表示每个词在邮件中出现不影响其他词在邮件中出现，这些时间是独立的。这样就可以得到下式:



### 1.2 公式推导

关于最大释然函数如何设置，我们需要知道我们的目标是找到参数，使得训练时候在给定的情况下，得到最准确的概率最大，最大释然函数就是这些概率的连乘，具体是需要保证下面的式子最大：



在很多教材中都介绍最大释然函数是如下的表达，实际上是一样的。下面的式子的直接意义是在已知情况下，出现的概率，实际上面的分析过程是一样的。



注: 后面为了简写后面省略了。

我们设,,。









然后我们对求偏导数：



所以有:



然后继续求偏导数:





如果一个单词不再训练样本中，就会出现0/0的现象。避免这个事件发生，引入拉普拉斯平滑。所以有:



如果一个单词不再训练样本中，就会出现0/0的现象。避免这个事件发生，引入拉普拉斯平滑。

同理有:



注:试分析这样的公式，就是总的垃圾邮件数目除以从样本数。就是所有垃圾邮件中出现对应的单词的数目除以所有垃圾邮件的数目。实际上这是个很容理解的道理。当然这就是数学的魅力，即便很简单的公司都是有着其理论依据的。

### 1.3 算法的实现

我们使用样本得到各个单词的概率分布，然后预测邮件是否为垃圾邮件。运行下面的程序，可以得到了正确的垃圾邮件分类。

**import** numpy **as** np  
**import** re  
  
**global** DEBUG  
  
**def** constructDic():  
 file = open(**"../res/NaiveBayes/dict.txt"**)  
 *# key is the word, value is the sequence number* dict = {} *# In fact, treeMap is better than dict* count = 0;  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break  
 for** word **in** line.split():  
 **if** word.lower() **not in** dict:  
 dict[word.lower()]=count  
 count = count+1  
 **return** dict  
  
**def** findIndexInDict(word,dict):  
 **if** word **in** dict:  
 **return** dict[word]  
 **else**:  
 **return** -1  
  
**def** parseSpamOrHam(words):  
 **if** words[0] == **"spam"**:  
 **return** 1  
 **elif** words[0] == **"ham"**:  
 **return** 0  
 **else**:  
 **if** DEBUG:  
 print(**"error email type in training sample!"**)  
 **return** -1  
  
**def** splitEmail(line):  
 regEx = re.compile(**r'[^a-zA-Z]|\d'**)  
 **return** list(filter(**lambda** word: word!=**""**, regEx.split(line)))  
  
*# spam is 1, ham is 0***def** parseEmails(x\_arr,y\_arr,dict,dictLen):  
 file = open(**"../res/NaiveBayes/emails.txt"**)  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break** words = splitEmail(line)  
 y = parseSpamOrHam(words)  
 x = np.zeros(dictLen)  
 **if** y == -1:  
 **continue**;  
 **for** word **in** words[1:]:  
 **if** word.lower() **in** dict:  
 index = dict[word.lower()]  
 x[index]=1  
 **else**:  
 **if** DEBUG:  
 print(word,**" is not in dic!"**)  
 x\_arr.append(x)  
 y\_arr.append(y)  
 *#if DEBUG:  
 # for i in range(len(x)):  
 # if x[i] == 1:  
 # print(i)* **return** dict  
  
**def** calPhiY(y\_arr):  
 *# p(y=1)* sum = 0  
 **for** i **in** y\_arr:  
 sum = sum + i  
 **return** sum/len(y\_arr)  
  
**def** calPhiXY(y\_arr,x\_arr,knownY,dictLen):  
 *# return a vector* sum = 0  
 ret = np.zeros(dictLen)  
 **for** i **in** range(len(y\_arr)):  
 **if** y\_arr[i]!=knownY:  
 **continue** sum=sum+1  
 **for** j **in** range(dictLen):  
 **if** x\_arr[i][j]==1:  
 ret[j]=ret[j]+1  
 **return** (ret+1)/(sum+2)  
  
**def** classify(phi,phi\_y0,phi\_y1,dictLen):  
 file = open(**"../res/NaiveBayes/testEmails.txt"**)  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break** words = splitEmail(line)[1:]  
 x = np.zeros(dictLen)  
 **for** word **in** words:  
 **if** word.lower() **in** dict:  
 index = dict[word.lower()]  
 x[index]=1  
 res = calcPro(phi,phi\_y1,x)/(calcPro(phi,phi\_y1,x)+calcPro(phi,phi\_y0,x))  
 **if** res > 0.5:  
 print(**"spam :"** + line)  
 **else**:  
 print(**"ham :"** + line)  
 **return** dict  
  
**def** calcPro(phi,phi\_y,x):  
 *# p(x|y=phi\_y)* ret = 1  
 **for** i **in** range(len(x)):  
 **if** x[i] == 1:  
 ret = ret \* phi\_y[i]  
 **else**:  
 ret = ret \* (1-phi\_y[i])  
 **return** ret\*phi  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *# 0. debug options* DEBUG = **False** *# 1. construct Dict* dict = constructDic()  
 dictLen = len(dict)  
  
 *# 2. parse the email to generate the sample, x and y* x\_arr=[]  
 y\_arr=[]  
 parseEmails(x\_arr,y\_arr,dict,dictLen)  
  
 *# 3. learn from the sample* phi = calPhiY(y\_arr) *# p(y)=1* phi\_y1 = calPhiXY(y\_arr, x\_arr, 1, dictLen)  
 phi\_y0 = calPhiXY(y\_arr, x\_arr, 0, dictLen)  
  
 *# 4. test classify  
 #p(y=1|x)=p(x|y=1)p(y=1)/(p(x|y=1)p(y=1)+p(x|y=0)p(y=0))  
 #p(x|y=1) = Pe(p(x=x^i|y=1))* classify(phi,phi\_y0,phi\_y1,dictLen)

## 2 多元伯努利事件模型

### 2.1 公式推导

下面采用另外一种方式进行建模。对于。其中代表邮件的第一个单词在词库中的索引为1。仍然有如下公式



我们设置，。

然后求最大释然函数









我们对求偏导数，与之前相同，这里直接写出：



我们对求偏导数，如下：



其中有：





所以有：



根据拉普拉斯平滑，所以最后得到下面的公式，其中V为词库中单词的数目。



同理有:



### 2.2算法的实现

我们使用样本得到各个单词的概率分布，然后预测邮件是否为垃圾邮件。运行下面的程序，可以得到了正确的垃圾邮件分类。

注: 由于概率值过小，多次乘积会超过浮点数精度范围，所以程序这里乘以一个固定的比例系数

**import** numpy **as** np  
**import** re  
  
**global** DEBUG  
**global** SCALE  
  
**def** constructDic():  
 file = open(**"../res/NaiveBayes/dict.txt"**)  
 *# key is the word, value is the sequence number* dict = {} *# In fact, treeMap is better than dict* count = 0;  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break  
 for** word **in** line.split():  
 **if** word.lower() **not in** dict:  
 dict[word.lower()]=count  
 count = count+1  
 **return** dict  
  
**def** findIndexInDict(word,dict):  
 **if** word **in** dict:  
 **return** dict[word]  
 **else**:  
 **return** -1  
  
**def** parseSpamOrHam(words):  
 **if** words[0] == **"spam"**:  
 **return** 1  
 **elif** words[0] == **"ham"**:  
 **return** 0  
 **else**:  
 **if** DEBUG:  
 print(**"error email type in training sample!"**)  
 **return** -1  
  
**def** splitEmail(line):  
 regEx = re.compile(**r'[^a-zA-Z]|\d'**)  
 **return** list(filter(**lambda** word: word!=**""**, regEx.split(line)))  
  
*# spam is 1, ham is 0***def** parseEmails(x\_arr,y\_arr,dict,dictLen):  
 file = open(**"../res/NaiveBayes/emails.txt"**)  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break** words = splitEmail(line)  
 y = int(parseSpamOrHam(words))  
 words = words[1:]  
 x = np.zeros(len(words),int)  
 **if** y == -1:  
 **continue**;  
 **for** i **in** range(len(words)):  
 **if** words[i].lower() **in** dict:  
 index = dict[words[i].lower()]  
 x[i]=index  
 **else**:  
 x[i]=dictLen+1  
 x\_arr.append(x)  
 y\_arr.append(y)  
 *#if DEBUG:  
 # for i in range(len(x)):  
 # print(x[i])* **return** dict  
  
**def** calPhiY(y\_arr):  
 *# p(y=1)* sum = 0  
 **for** i **in** y\_arr:  
 sum = sum + i  
 **return** sum/len(y\_arr)  
  
**def** calPhiXY(y\_arr,x\_arr,knownY,dictLen):  
 *# return a vector* sum = 0  
 ret = np.zeros(dictLen)  
 **for** i **in** range(len(y\_arr)):  
 **if** y\_arr[i]!=knownY:  
 **continue** sum=sum+len(x\_arr[i])  
 **for** j **in** range(len(x\_arr[i])):  
 index = x\_arr[i][j]  
 ret[index]=ret[index]+1  
 **return** SCALE\*(ret+1)/(sum+dictLen)  
  
**def** classify(phi,phi\_y0,phi\_y1,dictLen):  
 file = open(**"../res/NaiveBayes/testEmails.txt"**)  
 **while** 1:  
 line = file.readline()  
 **if not** line:  
 **break** words = splitEmail(line)[1:]  
 x = np.zeros(len(words),int)  
 **for** i **in** range(len(words)):  
 **if** words[i].lower() **in** dict:  
 index = dict[words[i].lower()]  
 x[i]=index  
 res = calcPro(phi,phi\_y1,x)/(calcPro(phi,phi\_y1,x)+calcPro(phi,phi\_y0,x))  
  
 **if** res > 0.5:  
 print(**"spam :"** + line)  
 **else**:  
 print(**"ham :"** + line)  
 **return** dict  
  
**def** calcPro(phi,phi\_y,x):  
 *# p(x|y=phi\_y)* ret = 1  
 **for** i **in** range(len(x)):  
 ret = ret\*phi\_y[x[i]]  
 *#print(ret)* **return** ret\*phi  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 *# 0. debug options* DEBUG = **True** SCALE = 10e3  
  
 *# 1. construct Dict* dict = constructDic()  
 dictLen = len(dict)  
  
 *# 2. parse the email to generate the sample, x and y* x\_arr=[]  
 y\_arr=[]  
 parseEmails(x\_arr,y\_arr,dict,dictLen)  
  
 *# 3. learn from the sample* phi = calPhiY(y\_arr) *# p(y)=1* phi\_y1 = calPhiXY(y\_arr, x\_arr, 1, dictLen)  
 phi\_y0 = calPhiXY(y\_arr, x\_arr, 0, dictLen)  
  
 *# 4. test classify  
 #p(y=1|x)=p(x|y=1)p(y=1)/(p(x|y=1)p(y=1)+p(x|y=0)p(y=0))  
 #p(x|y=1) = Pe(p(x=x^i|y=1))* classify(phi,phi\_y0,phi\_y1,dictLen)