机器学习：使用朴素贝叶斯过滤垃圾邮件（Python 完成）

杨逸翔（2015301020105）

武汉大学 物理科学与技术学院 弘毅班

摘　要　垃圾邮件可以说是因特网带给人类最具争议性的副产品，它的泛滥已经使整个因特网不堪重负。垃圾邮件（spam）现在还没有一个非常严格的定义。一般来说，凡是未经用户许可就强行发送到用户的邮箱中的任何电子邮件。在互联网高度发达的今天，几乎每个人都被垃圾邮件所困扰，甚至不少人每天接收的垃圾邮件数目远多于正常邮件，如果有办法能用计算机批量处理垃圾邮件，那是再好不过的了。本文就讨论了一种使用机器学习的处理垃圾邮件的方法（用 Python 实现）。

关键词　Python 垃圾邮件处理 朴素贝叶斯 机器学习

一， 条件概率

在概率论中，我们学过一种有效计算条件概率的方法贝叶斯准则：

如果 A 事件发生的概率 p（A），B 事件发生的概率为 p（B），A，B 同时发生的概率为 p(AB)，B 发生的条件下 A 事件发生的概率为 p（A\B），则：

$$\mathrm{p(B\backslash A)} = \frac{p(A\backslash B)p(B)}{p(A)}$$

二， 使用条件概率来分类

本文讨论的处理垃圾邮件的项目本质上是对事件进行分类。贝叶斯分类的准则的是：

如果向量（x，y）属于类别 1 的概率是 p1（x，y），属于类别二的概率为 p2（x，y），那么 p1（x，y）> p2（x，y）时，属于类别 2；p1（x，y）<p2（x，y）时，属于类别 2。

在这里我们为了简便起见使用 p1，p2，实际上对于区分 c1 类何 c2 类应该比较的是 p（c1\x，y）和 p（c2\x，y）。然而在分析样本时我们只能得到某一类中出现某一向量的概率也就是 p（x，y\c1）。此时则需要用（一）中所讲的贝叶斯法则来进行计算 p1，和 p2。下面我们用一个简单的对文档分类来具体讨论分类方法。

三， 使用朴素贝叶斯进行文档分类

在我们对文档进行分类时，我们需要将文档处理为一种方便被计算机处理的格式。一种方式就是将文档处理为一个向量。我们把每个词出现或者不出现作为一个特征（向量的一个维度）。这样只要建立一个关于所有词的向量，就能用这个向量来表示一个文档。

我们举一个简单的例子（配合相应的代码）来说明这个问题。

```
def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],
                 ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],
                 ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                 ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop', 'him'],
                 ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1]
    return postingList,classVec

def createVocabList(dataSet):
    vocabSet = set([])
    for document in dataSet:
        vocabSet = vocabSet | set(document)
    return list(vocabSet)

def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else: print ("the word: %s is not in my Vocabulary!" % word)
    return returnVec
```

我们先用 loadDataList（）创建了一个用来实验的样本。这个样本中话分为侮辱性和非侮辱性的，我们用 0 和 1 来表示，并建立了一个标签集合。

建立的第二个函数创建了一个包含文档中不重复词的列表。

获得了词汇列表后，我们定义了第三个函数。该函数的输入参数为词汇表及某个文档。输出的则是文档向量，向量的每个元素为 0 或者 1（0 代表此文档没有出现这个词，1 则代表出现了这个词）。如果我们把每个文档都带入这个函数则可以获得所有的文档的向量。

四， 训练算法：由词向量计算概率

获得文档向量后，我们要算一算侮辱类和非侮辱类的文档的中每个词出现的概率。如果认为文档中每个词出现的概率是相互独立的那么，那我们就可以很容易得出 p

（w1，w2，……..wn\c1）=p(w1\c1)p(w2\c1)……..p(wn,c1)。下面我们就用代码求侮辱性与非侮辱性每个词的词频，侮辱文档的频率。

```
def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
    p0Num = ones(numWords); p1Num = ones(numWords)
    p0Denom = 2.0; p1Denom = 2.0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = log(p1Num/p1Denom)
    p0Vect = log(p0Num/p0Denom)
    return p0Vect,p1Vect,pAbusive
```

在这里我对概率的计算进行了微小的调整。

1，有些词的一类文档中从没出现过，那就导致 p（w1，w2，……..wn\c1）=p(w1\c1)p(w2\c1)……..p(wn,c1)算出来总是 0。这显然不合理，所以我们初始化每个词出现的次数为 1。

2，等会儿很多小数的相乘很有可能导致下溢出。于是对所有概率取对数。

所有的准备工作都已经做完现在可以用这个来测试某一个未知的文档究竟是侮辱性还是非侮辱性的。代码如下：

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0
```

五， 使用朴素贝叶斯过滤垃圾邮件

在把上面的方法用于垃圾邮件过滤之前，先对前面的方法进行一点优化。前面的讨论中，我们的文档向量的只是考虑了每个词是否出现，这显然是不够严谨的。很容易想到，每个词出现的真实频率和文档的性质也有着密切的关联。这里用文档词袋模型来建立每个文档的向量。

```
57 def bagOfWords2VecMN(vocabList, inputSet):
58     returnVec = [0]*len(vocabList)
59     for word in inputSet:
60         if word in vocabList:
61             returnVec[vocabList.index(word)] += 1
62     return returnVec
```

现在就可以用上述区分文档的方法来区分一封邮件是否是垃圾邮件。与前文描述有区别的是我们需要把训练的文本切分成一个个的词，然后建立词向量。要考虑到的是实际邮件中会有标点符号，还有为了利于建立词汇表要把所有的大写都变成小写。这些都不难用 python 实现。

为了测试这个算法的可靠性，我用了 50 封邮件进行实验（25 封是来自 spam 文件夹，25 封来自 ham 文件夹）。用这 50 封邮件作为训练集（显然作为测试集的样本越多，这个算法越可靠。虽然样本不多，但是实际上已经有了很好的效果）。接下来随机选择 10 封邮件进行测试（相应地从训练集中剔除这十封），并计算错误率。（因为是随机选 10 封测试）多次测试，求出错误率的平均值就可以得出此测试器的平均错误率。

具体代码见附件，现在贴出测试结果（文末附图）

六， 结论

由于资源有限，可供训练的文档相当少。但是令人惊喜的是测试的结果依然不错，平均错误率在 0.09。这种贝叶斯分类器在处理垃圾邮件方面是十分有用的。

```
In [26]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.0
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [27]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [28]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [29]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.0
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)
```

```
In [30]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [31]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.0
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [32]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [33]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.2
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
```

```
In [34]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.2
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [35]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [36]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)

In [37]: runfile('C:/Users/YOUNG/.spyder-py3/temp.py', wdir='C:/Users/
YOUNG/.spyder-py3')
the error rate is:  0.1
C:\Users\YOUNG\Anaconda3\lib\re.py:203: FutureWarning: split() requires a
non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)
```