基于朴素贝叶斯算法实现的屏蔽社区留言板的侮辱性言论问题（附：Python 代码）

——机器学习部分

姓名：曾昕菟　学号：17211170

1 前言

通过老师对应用层部分的讲解，我了解到机器学习可以应用于歌曲风格识别、手写体识别、人脸动漫化等方面，因此也被深深吸引，对机器学习部分产生了较为浓厚的兴趣，结合课后我对该部分资料的查询，可以做出以下概括，机器学习是使机器具有智能的根本途径，其主要使用的是归纳，而不是演绎推理。机器学习步骤可以概括为六部分，收集数据、整理数据、分析数据、训练算法、测试算法、使用算法。之后的具体代码也会采用该方法进行分析，在该过程中最重要的是训练算法部分，较为主流的算法有 KNN 算法、朴素贝叶斯算法、决策树算法等，此处讨论的是监督学习，即数据带有标签，对于非监督学习，此部分可以不存在，过去几周我主要学习了就是以上三种算法，但是该文中作为案例讲解的算法是朴素贝叶斯算法。

2 朴素贝叶斯算法

朴素贝叶斯算法的根本就是贝叶斯，对于贝叶斯公式应该都不太陌生，在概率论中讲到贝叶斯为：$P(c_i \mid x,y)=\dfrac{P(x,y \mid c_i)P(c_i)}{P(x,y)}$

其中我们可以把 ci 可以看成分类 i，(x,y)为特征向量，采用 MAP 准则，对于 $\max\{P(c_i \mid x,y)\}$ 的 i，我们就认为该数据是属于分类 i 的，而以上式子右边的值都能通过训练数据得到，因此机器可以判断出分类。根据以上的六步概括，朴素贝叶斯算法可以进一步解释。

①收集数据: 通过某种方法进行数据的收集，可以是读取文档，也可以是自己定义数据集。
②整理数据: 将收集的数据集转化为方便进行计算的数据类型，提取好特征向量。
③分析数据: 将整理好的数据进行分析，并根据是否有不符合要求的数据，进行剔除，以免干扰判断。
④训练算法: 通过贝叶斯公式计算概率。
⑤测试算法: 计算错误率。
⑥使用算法: 通过以上方式即可实现分类。

3 实例分析

实例（来源于 Github）：构建一个快速过滤器来屏蔽在线社区留言板上的侮辱性言论。如果某条留言使用了负面或者侮辱性的语言，那么就将该留言标识为内容不当。对此问题建立两

个类别: 侮辱类和非侮辱类，使用 1 和 0 分别表示。

3.1 方法分析

套用上面概括的六步走策略。
①首先进行收集数据，收集方法可以任意，因此我们随便取一种，即自定义数据集；
②整理数据中，我们要做的是把收集好的句子分解，提取出单词表，每句话中的单词是否出现就可以作为特征向量，考虑到我们后续需要计算概率，因此我们在这一步也可以统计输入的情况，即输入中出现过的单词；
③分析数据，分析数据需要做的是先删除单词表中重复的单词，因为重复会影响后续判断，然后再进行一次人为确认，无重复单词；
④训练算法，整个实例的关键在于训练算法，由于采用的是朴素贝叶斯算法，因此需要计算概率，通过以上我们确认了特征向量 w 为一段话中单词在单词表中是否出现，例如 w=[0,1,0,0]，代表了单词表中的第二个单词出现了，其他的单词未出现，在贝叶斯公式中，由于分母都相同，因此可以省略分母计算，只计算分子，更进一步可以假设相互独立，因此可以拆成 $P(w|c_i)=P(w_0|c_i)P(w_1|c_i)P(w_2|c_i)P(w_3|c_i)$，这样更方便计算，$P(w_j|c_i)$ 就等于在类别 i 中，单词 j 的出现概率，因此通过遍历可以很容易计算；$P(c_i)$ 为分类 i 出现的概率，只需将训练数据的分类 i 中句数除以总的句数即可得到。这样即可判断后验概率最大的分类 i；
⑤测试算法，利用数据，测试以上训练算法是否可行，若不可行则需要进行额外调整；
⑥应用算法，通过一个调整好的训练算法，就可以将步骤②中提取的输入数据向量作为训练算法的输入，得到的即使分类结果。

3.2 算法实现

有了以上的分析，可以写出以下代码，仍然分步骤来看。

3.2.1 收集数据

```
def loadDataSet():
"""
 创建数据集 :
return: 单词列表 postingList,所属类别 classVec
"""
 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] # 1 代表侮辱性言论，0 代表正常言论
return postingList, classVec
```

可以看出，代码部分和以上分析一样，利用自定义一个训练数据集，代表哪些是侮辱性言论，哪些是正常言论，作为收集数据的函数，需要返回数据集和数据集的标签。

3.2.2 整理数据

```
def createVocabList(dataSet):
 """
获取所有单词的集合
:dataSet: 数据集
:return: 所有单词的集合(即不含重复元素的单词列表)
 """
 vocabSet = set([]) # 创建好集合以存储单词
 for document in dataSet: # 操作符 | 用于求两个集合的并集
 vocabSet = vocabSet | set(document) # 以获取单词表
 return list(vocabSet)
```

按照以上分析，我们要在获取言论中得到所有单词表，以方便创建特征向量，因此以言论文档为输入数据，提取出单词表，并作为返回值。

```
def setOfWords2Vec(vocabList, inputSet):
"""
 遍历查看该单词是否出现，出现该单词则将该单词置 1
 :param vocabList: 所有单词集合列表
 :param inputSet: 输入数据集
 :return: 匹配列表[0,1,0,1...]，其中 1 与 0 表示词汇表中的单词是否出现在输入的数据集中
 """
 # 创建一个和词汇表等长的向量，并将其元素都设置为 0
 returnVec = [0] * len(vocabList)# [0,0......] # 遍历文档中的所有单词，如果出现了词汇表中的单词，则将输出的文档向量中的对应值设为 1
 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
```

正如分析中谈到，对于输入数据的特征向量提取也需要在这一步做到，我们采用的具体方法是将输入数据言论，并将输入数据言论的单词逐个和之前提取出的单词表进行对比，将出现的单词在单词表对应的位置标记为 1，这样得到的就是输入言论的特征向量。

3.2.3 分析数据

```
listOPosts, listClasses = loadDataSet()
myVocabList=createVocabList(listOPosts)
print(myVocabList)  #展现单词表以查看有无重复单词
print(setOfWords2Vec(myVocabList, listOPosts[0]))
print(setOfWords2Vec(myVocabList, listOPosts[3]))  #展现函数是否有效
```

这部分可以不用单独设置子函数，作为表明单词表有无问题出现。此外还需要测试言论的特征向量是否能够成功提取，这里采用言论文档第一条和第四条作为测试。

3.2.4 训练算法

```
def _trainNB0(trainMatrix, trainCategory):
 """ 训练数据原版
 :param trainMatrix: 文件单词矩阵 [[1,0,1,1,1....],[],[]...]
 :param trainCategory: 文件对应的类别[0,1,1,0....]，列表长度等于单词矩阵数，其中的 1 代表对应的文件是侮辱性文件，0 代表不是侮辱性矩阵
 :return:
 """
 # 文件数
 numTrainDocs = len(trainMatrix)
 # 单词数
 numWords = len(trainMatrix[0])
 # 侮辱性文件的出现概率，即 trainCategory 中所有的 1 的个数，
 # 代表的就是多少个侮辱性文件，与文件的总数相除就得到了侮辱性文件的出现概
 pAbusive = sum(trainCategory) / float(numTrainDocs) # 构造单词出现次数列表
 p0Num = zeros(numWords) # [0,0,0,.....]
 p1Num = zeros(numWords) # [0,0,0,.....]
 # 整个数据集单词出现总数
 p0Denom = 0.0
 p1Denom = 0.0
 for i in range(numTrainDocs): # 是否是侮辱性文件
 if trainCategory[i] == 1: # 如果是侮辱性文件，对侮辱性文件的向量进行加和
 p1Num += trainMatrix[i] #[0,1,1,....] + [0,1,1,....]->[0,2,2,...]
 # 对向量中的所有元素进行求和，也就是计算所有侮辱性文件中出现的单词总数
 p1Denom += sum(trainMatrix[i])
 else: p0Num += trainMatrix[i]
 p0Denom += sum(trainMatrix[i])
 # 侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)....]表
 # 即在 1 类别下，每个单词出现的概率
 p1Vect = p1Num / p1Denom  # [1,2,3,5]/90->[1/90,...]
 # 正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)....]表
 # 即在 0 类别下，每个单词出现的概率
 p0Vect = p0Num / p0Denom
 return p0Vect, p1Vect, pAbusive
```

训练算法作为核心部分，按照前文的分析，需要计算概率，因此代码最开始就计算侮辱性言论所占总言论的百分比，即 $P(c_i)$,接下来还需要计算 $P(w|c_i)=\prod_j P(w_j|c_i)$，因此我们需要单独计算侮辱性言论和非侮辱性言论中，索引 j 所指示的单词出现概率，最后返回这个概率表和之前计算的 $P(c_i)$，有了这两项就很好的计算后验概率。

3.2.5 测试算法

```
def trainNB0(trainMatrix, trainCategory):
"""
 训练数据优化版本 :param trainMatrix:
 文件单词矩阵 :param trainCategory:
 文件对应的类别 :return:
 """
 # 总文件数
 numTrainDocs = len(trainMatrix)
 # 总单词数
 numWords = len(trainMatrix[0])
 # 侮辱性文件的出现概率
 pAbusive = sum(trainCategory) / float(numTrainDocs)
 # 构造单词出现次数列表
 # p0Num 正常的统计
 # p1Num 侮辱的统计
 p0Num = ones(numWords)#[0,0......]->[1,1,1,1,1.....]
 p1Num = ones(numWords) # 整个数据集单词出现总数，2.0 根据样本/实际调查结果调整分母的值（2 主要是避免分母为 0，当然值可以调整）
 # p0Denom 正常的统计
 # p1Denom 侮辱的统计
 p0Denom = 2.0
 p1Denom = 2.0
 for i in range(numTrainDocs):
 if trainCategory[i] == 1: # 累加辱骂词的频次
 p1Num += trainMatrix[i] # 对每篇文章的辱骂的频次 进行统计汇总
 p1Denom += sum(trainMatrix[i])
 else:
 p0Num += trainMatrix[i]
 p0Denom += sum(trainMatrix[i]) # 类别 1，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]列表
 p1Vect = log(p1Num / p1Denom) # 类别 0，即正常文档的[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]列表
 p0Vect = log(p0Num / p0Denom)
```

```
return p0Vect, p1Vect, pAbusive
```

在训练算法中，采用基本的贝叶斯算法思维得到了数据，但是由于言论不是始终如一的，即有的非侮辱性言论中可能含有侮辱性言论中存在的单词，有的侮辱性言论也可能含有非侮辱性言论中存在的单词，例如存在一个侮辱性言论，其存在一个仅在非侮辱性言论中出现的单词，这时候的概率计算就存在一项 $P(w_j|c_i)$为 0，由于我们假设的是相互独立，因此最后的概率乘积就是 0，无法进行正确分类，所以在这一步需要进行训练算法调整，我们可以考虑通过将所有单词的初始化次数为 1，这样可以避免出现概率为 0 的情况，输入言论单词量越大，这种造成的不准确度越小，但是如果过大，出现的问题是每个单词的概率会减小，这样过多小于 1 的数相乘，可能造成得到的结果也为 0，因此在这里做一个数学变化，我们计算的是概率 P，那么加上一个对数函数后，即 ln(P)，再做分析。我们可以考虑到，两种函数单调性大致相同，但也有不同，当各独立的概率直接相乘，由于概率值都比较小，计算机内存有限，很容易将结果默认为 0，而采用对数函数，在接近概率值接近 0 时候就不会出现这种状况，因此我们采用 ln 函数进行近似计算以规避后验概率计算到 0 的情况。

3.2.6 应用算法

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
 """
 使用算法: # 将乘法转换为加法
 :param vec2Classify:待测数据[0,1,1,1,1...]，即要分类的向量
 :param p0Vec: 类别 0，即正常文档的
[log(P(F1|C0)),log(P(F2|C0)),log(P(F3|C0)),log(P(F4|C0)),log(P(F5|C0))....]
列表
 :param p1Vec: 类别 1，即侮辱性文档的
[log(P(F1|C1)),log(P(F2|C1)),log(P(F3|C1)),log(P(F4|C1)),log(P(F5|C1))....]
列表
 :param pClass1: 类别 1，侮辱性文件的出现概率
 :return: 类别 1 or 0 """
 # 计算公式 log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C))
 p1 = sum(vec2Classify * p1Vec) + log(pClass1) # P(w|c1) * P(c1) ，即贝叶斯准
则的分子
 p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1) # P(w|c0) * P(c0) ，即贝
叶斯准则的分子·
 if p1 > p0:
 return 1   #表言论含有侮辱性
 else:
 return 0   #表言论无侮辱性
```

有了以上调整后的训练算法，即可用于应用，在步骤②中提取好了输入言论的特征向量，又在训练算法计算好了贝叶斯算法中的各种需要的参量，因此只需要根据输入言论的特征向量计算好对应后验概率，再根据 MAP 准则即可判断出该言论的合法性。

3.3 实验结果

我们分两步呈现结果，第一步是分析结果的呈现，即呈现向量提取函数的正确性和单词表无重复性。实现以下代码：

```
listOPosts, listClasses = loadDataSet()
myVocabList=createVocabList(listOPosts)
print(myVocabList)  #展现单词表以查看有无重复单词
print(setOfWords2Vec(myVocabList, listOPosts[0]))
print(setOfWords2Vec(myVocabList, listOPosts[3]))
```

结果为：

```
['him', 'stop', 'dalmation', 'garbage', 'not', 'I', 'please', 'dog', 'stupid', 'problems', 'quit', 'maybe', 'love', 'how', 'is', 'park', 'cute', 'mr', 'buying', 'has', 'take', 'posting', 'worthless', 'ate', 'licks', 'my', 'he
lp', 'flea', 'so', 'to', 'food', 'steak']
[0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0]
[0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

回到训练数据集，可以对比所有单词已出现，且单词表中无重复性的单词，此外我们采用的是言论 0 和言论 1 作为分析，该言论分别为：

```
['my', 'dog', 'has', 'flea', 'problems', 'help', 'please']
['stop', 'posting', 'stupid', 'worthless', 'garbage']
```

对比单词表，发现出现的单词已全部为 1，说明特征向量提取函数有效。

接下来执行整个代码，输入如下：

```
    listOPosts, listClasses = loadDataSet()     #  收集数据
    myVocabList = createVocabList(listOPosts)  #  整理数据的单词表
    trainMat = []  # 提取特征向量
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    p0V, p1V, pAb = trainNB0(array(trainMat), array(listClasses))  #训练数据
    testEntry = ['love', 'my', 'dalmation',]   #  应用数据，这里的数据是假设的，
实际中可以采用任何方式获取真实输入数据
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
#查看言论的分类
    testEntry = ['how','stupid', 'garbage']   #应用数据
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))
    print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))
#查看言论的分类
```

得到的结果如下：

```
['love', 'my', 'dalmation'] classified as:  0
['how', 'stupid', 'garbage'] classified as:  1
```

可以看出对于第一个言论已经分类到非侮辱性言论中，第二个言论被分类到了侮辱性言论，

从实际言论意思中分析，确实如此。

3.4 可选优化和个人感悟

通过分析以上程序，发现也有着可选的优化方案，原始方案的缺点一个在于收集的言论集不变，因此概率集不改变，而事实上随着时代变化，可能有些单词会蕴育出新的含义，甚至可能从非侮辱性变为侮辱性，第二个缺点在于只能根据单词表中存在的单词进行判断，如果单词表中不存在该单词，则无法准确判断言论的侮辱性。

因此提出一种新优化方案，在每次判断言论之后，将该言论和它的侮辱性作为训练数据而更新到训练数据集中，假设有单词表不存在的单词，则在这次计算中，将概率默认为 1，相当于不存在，而计算过后，添加的单词表中，这样就能自适应。

考虑一种情况，有一个单词从非侮辱性变为了侮辱性单词，而某输入言论恰好用了该单词，假设该言论是侮辱性言论，由于物以类聚，因此除了该单词以外很可能大部分都是侮辱性单词，其中既存在单词表能识别的，也存在单词表不能识别的，假设采用自适应算法，那么单词表更新，既能把变味的单词给识别出来，也能把不存在的单词给添加到单词表中。很好的规避了以上两个缺点。

但是事实上，还有新的情况需要考虑，假设一个言论中所有单词都是新单词，依然难以通过优化后的方案分类，所以说一个问题消失往往又面临另一个问题的诞生，像通信网络基础中说到一样，分布式路由算法的问题层出不穷，从计数到无穷开始，提出反向毒化、水平分割，又出现多节点的自环问题，再提出携带整条路由路径，结果又出现了新的问题，只能不断优化，提出新的方案，才能不断稳定。