# Lab\_1 基于朴素贝叶斯的垃圾邮件检测

## 一、实习目的与要求

1、结合实际应用理解分类的分类过程；

2、深入理解特征提取、数据处理、训练、测试、分类评估等过程；

3、理论结合实践，采用朴素贝叶斯方法实现垃圾邮件的检测；

4、通过进一步查阅文献，了解相关研究方向的最新研究进展。

## 二、实习题目

利用基于概率论的分类方法——朴素贝叶斯方法，实现垃圾邮件的检测。

#### 【实验数据】

50封包含纯文本内容的电子邮件，其中50% SPAM，50% HAM。

#### 【分类过程描述】

1. **数据预处理**

第23个样本中有一个“？”读取不了，所以直接做了一件简单的删除

1. **特征选择**

①将文本解析成词条向量

使用正则表达式来切分文本

Hi Peter,

With Jose out of town, do you want to

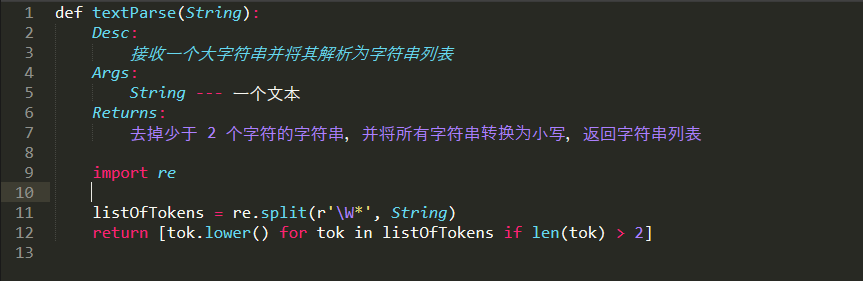
meet once in a while to keep things

going and do some interesting stuff?

Let me know

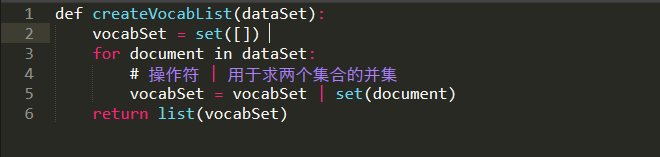
Eugene

去掉少于 2 个字符的字符串，并将所有字符串转换为小写，返回字符串列表



②分析数据: 检查词条确保解析的正确性

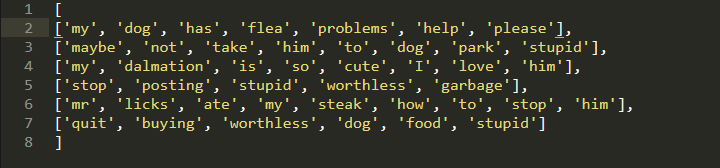
检查函数执行情况，检查词表，不出现重复单词，这里需要使用到集合来对重复的单词进行去重

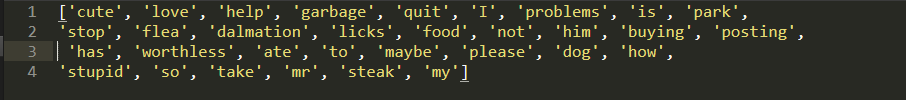


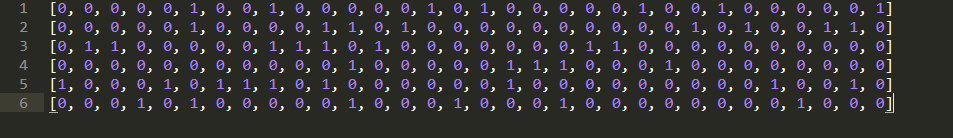
1. **模型训练**

①训练算法：从词条向量计算概率

举个例子：





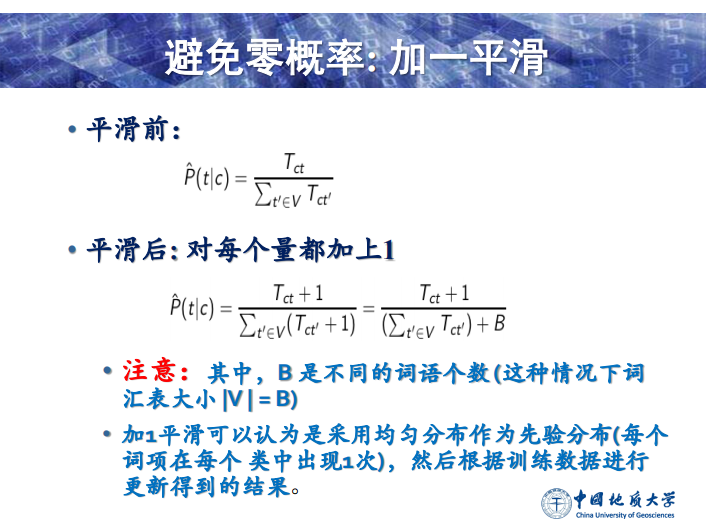


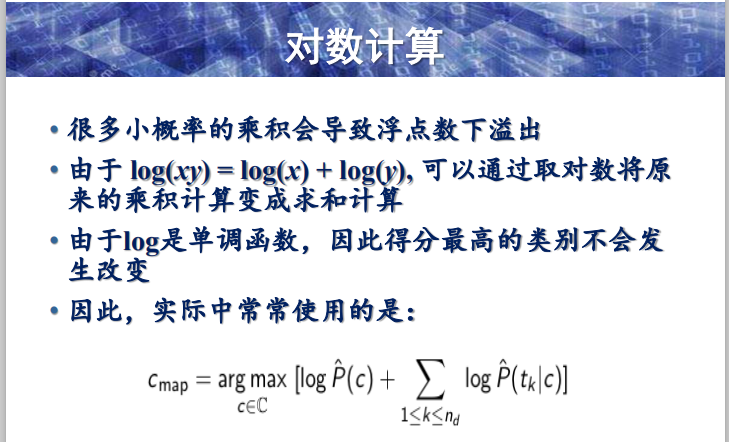
['love', 'my', 'dalmation']

['stupid', 'garbage']

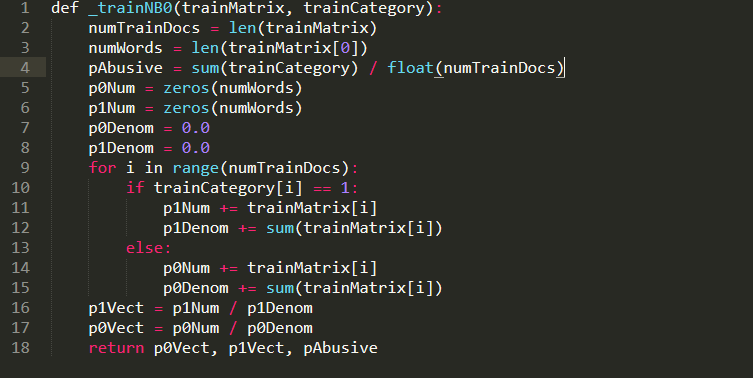
根据公式：



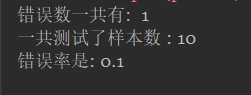
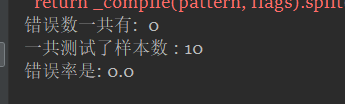




②使用算法：进行垃圾邮件分类和检测



1. **分类结果评价**



①根据样本的随机性结果也是不稳定的，有时候准确率是100%,有的时候确是90%，归根结底是数据选择的问题

#### 【源代码】

见附录

#### 【改进设想】

（或查阅文献提出改进设想；或考虑与第三方机器学习库分类结果的比较；或与采用其他分类算法实现的结果比较等）。

## 三、本次实习小结

（1）还是要继续学习数据挖掘这门课程的慕课，上课老师所讲的例子太单一了，而且上课有时没有听清导致下来了还是不会做，让人头大

（2）软件工程是一门比较注重编码基础的学科，对于自己认为好的编码方式，应该把思路记下来，方便以后的使用。

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'], #[0,0,1,1,1......]

['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 is abusive, 0 not

return postingList, classVec

def createVocabList(dataSet):

"""

获取所有单词的集合

:param 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([]) # create empty set

for document in dataSet:

# 操作符 | 用于求两个集合的并集

vocabSet = vocabSet | set(document) # union of the two sets

return list(vocabSet)

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

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率

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):

# 遍历所有的文件，如果是侮辱性文件，就计算此侮辱性文件中出现的侮辱性单词的个数

if trainCategory[i] == 1:

p1Num += trainMatrix[i] #[0,1,1,....]->[0,1,1,...]

p1Denom += sum(trainMatrix[i])

else:

# 如果不是侮辱性文件，则计算非侮辱性文件中出现的侮辱性单词的个数

p0Num += trainMatrix[i]

p0Denom += sum(trainMatrix[i])

# 类别1，即侮辱性文档的[P(F1|C1),P(F2|C1),P(F3|C1),P(F4|C1),P(F5|C1)....]列表

# 即 在1类别下，每个单词出现次数的占比

p1Vect = p1Num / p1Denom# [1,2,3,5]/90->[1/90,...]

# 类别0，即正常文档的[P(F1|C0),P(F2|C0),P(F3|C0),P(F4|C0),P(F5|C0)....]列表

# 即 在0类别下，每个单词出现次数的占比

p0Vect = p0Num / p0Denom

return p0Vect, p1Vect, pAbusive

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

# 避免单词列表中的任何一个单词为0，而导致最后的乘积为0，所以将每个单词的出现次数初始化为 1

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

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

def bagOfWords2VecMN(vocabList, inputSet):

returnVec = [0] \* len(vocabList)

for word in inputSet:

if word in vocabList:

returnVec[vocabList.index(word)] += 1

return returnVec

def testingNB():

"""

测试朴素贝叶斯算法

"""

# 1. 加载数据集

listOPosts, listClasses = loadDataSet()

# 2. 创建单词集合

myVocabList = createVocabList(listOPosts)

# 3. 计算单词是否出现并创建数据矩阵

trainMat = []

for postinDoc in listOPosts:

# 返回m\*len(myVocabList)的矩阵， 记录的都是0，1信息

print(setOfWords2Vec(myVocabList, postinDoc))

trainMat.append(setOfWords2Vec(myVocabList, postinDoc))

# 4. 训练数据

p0V, p1V, pAb = trainNB0(array(trainMat), array(listClasses))

# 5. 测试数据

testEntry = ['love', 'my', 'dalmation']

thisDoc = array(setOfWords2Vec(myVocabList, testEntry))

print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))

testEntry = ['stupid', 'garbage']

thisDoc = array(setOfWords2Vec(myVocabList, testEntry))

print(testEntry, 'classified as: ', classifyNB(thisDoc, p0V, p1V, pAb))

# ------------------------------------------------------------------------------------------

# 项目案例2: 使用朴素贝叶斯过滤垃圾邮件

# 切分文本

def textParse(bigString):

'''

Desc:

接收一个大字符串并将其解析为字符串列表

Args:

bigString -- 大字符串

Returns:

去掉少于 2 个字符的字符串，并将所有字符串转换为小写，返回字符串列表

'''

import re

# 使用正则表达式来切分句子，其中分隔符是除单词、数字外的任意字符串

listOfTokens = re.split(r'\W\*', bigString)

return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def spamTest():

'''

Desc:

对贝叶斯垃圾邮件分类器进行自动化处理。

Args:

none

Returns:

对测试集中的每封邮件进行分类，若邮件分类错误，则错误数加 1，最后返回总的错误百分比。

'''

docList = []

classList = []

fullText = []

for i in range(1, 26):

# 切分，解析数据，并归类为 1 类别

wordList = textParse(open('spam/%d.txt' % i).read())

docList.append(wordList)

classList.append(1)

# 切分，解析数据，并归类为 0 类别

wordList = textParse(open('ham/%d.txt' % i).read())

docList.append(wordList)

fullText.extend(wordList)

classList.append(0)

# 创建词汇表

vocabList = createVocabList(docList)

trainingSet = list(range(50))

testSet = []

# 随机取 10 个邮件用来测试

for i in range(10):

# random.uniform(x, y) 随机生成一个范围为 x - y 的实数

randIndex = int(random.uniform(0, len(trainingSet)))

testSet.append(trainingSet[randIndex])

del(trainingSet[randIndex])

trainMat = []

trainClasses = []

for docIndex in trainingSet:

trainMat.append(setOfWords2Vec(vocabList, docList[docIndex]))

trainClasses.append(classList[docIndex])

p0V, p1V, pSpam = trainNB0(array(trainMat), array(trainClasses))

errorCount = 0

for docIndex in testSet:

wordVector = setOfWords2Vec(vocabList, docList[docIndex])

if classifyNB(array(wordVector), p0V, p1V, pSpam) != classList[docIndex]:

errorCount += 1

print('错误数一共有: ', errorCount)

print('一共测试了样本数 :', len(testSet))

print('错误率是:', float(errorCount)/len(testSet))

def testParseTest():

print(textParse(open('data/4.NaiveBayes/email/ham/1.txt').read()))

=======================================================

我的联系方式：

电话：13349830890

邮箱：liuyy@cug.edu.cn