机器学习算法 day02_贝叶斯分类算法及应用

课程大纲

朴素贝叶斯算法原理	Bayes 算法概述
	Bayes 算法思想
	Bayes 算法要点
朴素贝叶斯算法案例 1	需求
	Python 实现
朴素贝叶斯算法案例 2	需求
	Python 实现

课程目标：

1、理解朴素贝叶斯算法的核心思想

2、理解朴素贝叶斯算法的代码实现

3、掌握朴素贝叶斯算法的应用步骤：数据处理、建模、运算和结果判定

1. 朴素贝叶斯分类算法原理

1.1 概述

贝叶斯分类算法是一大类分类算法的总称
贝叶斯分类算法以**样本可能属于某类的概率来作为分类依据**
朴素贝叶斯分类算法是贝叶斯分类算法中最简单的一种
注：朴素的意思是条件概率独立性

1.2 算法思想

朴素贝叶斯的**思想**是这样的：
　　如果一个事物在一些属性条件发生的情况下，事物属于 A 的概率>属于 B 的概率，则判定事物属于 A

通俗来说比如，你在街上看到一个黑人，我让你猜这哥们哪里来的，你十有八九猜非洲。为什么呢？
在你的脑海中，有这么一个判断流程：
1、这个人的肤色是黑色 <特征>
2、非洲人中黑色人种概率最高 <已知的是条件概率：p(黑色|非洲人)>
而用于判断的标准是：　P(非洲人|黑色)
3、没有其他辅助信息的情况下，最好的判断就是非洲人
这就是朴素贝叶斯的思想基础。

再扩展一下，假如某条街上，有 100 人，其中有 50 个美国人，50 个非洲人，看到一个讲英语的黑人，那我们是怎么去判断他来自于哪里？
提取特征：
肤色：　黑
语言：　英语

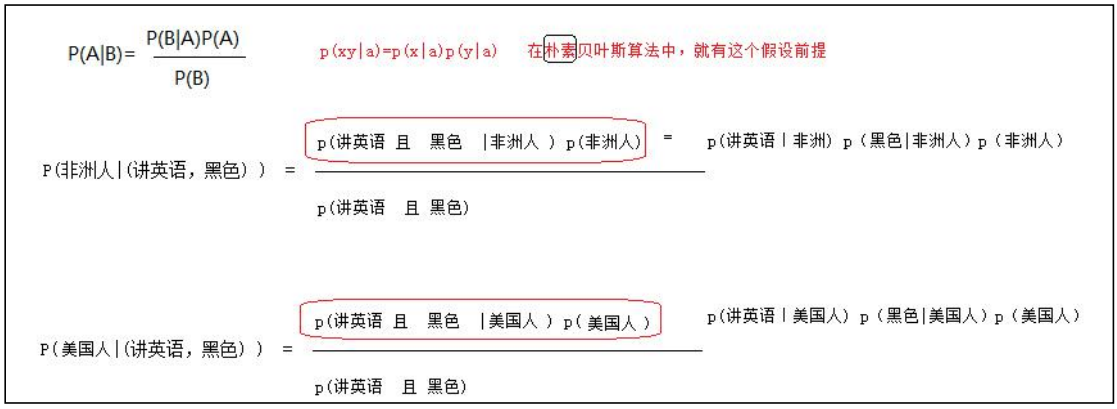
先验知识：
P(黑色|非洲人) = 0.8
P(讲英语|非洲人)=0.1

P(黑色|美国人)= 0.2
P(讲英语|美国人)=0.9

要判断的概率是：
P(非洲人|(讲英语，黑色))

P(美国人|(讲英语，黑色))

思考过程：



P(非洲人|(讲英语，黑色))　的 分子= 0.1 * 0.8 *0.5 =0.04

P(美国人|(讲英语，黑色))　的 分子= 0.9 *0.2 * 0.5 = 0.09

从而比较这两个概率的大小就　等价于比较这两个分子的值：
可以得出结论，此人应该是 ：美国人

我们的判断结果就是：此人来自美国！

其蕴含的数学原理如下：

p(A|xy)=p(Axy)/p(xy)=p(Axy)/p(x)p(y)=p(A)/p(x)*p(A)/p(y)* p(xy)/p(xy)=p(A|x)p(A|y)

朴素贝叶斯分类器

讲了上面的小故事，我们来朴素贝叶斯分类器的表示形式：

$$y=f(x)=\arg\max_{c_k} P(Y=c_k|X=x)=\arg\max_{c_k}\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)}$$

当特征为为 x 时，计算所有类别的条件概率，选取条件概率最大的类别作为待分类的类别。由于上公式的分母对每个类别都是一样的，因此计算时可以不考虑分母，即

$$y=f(x)=\arg\max_{c_k} P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)}|Y=c_k)$$

朴素贝叶斯的朴素体现在其对各个条件的独立性假设上，加上独立假设后，大大减少了参数假设空间。

1.3 算法要点

1.3.1 算法步骤

1、分解各类先验样本数据中的特征

2、计算各类数据中，各特征的条件概率

（比如：特征 1 出现的情况下，属于 A 类的概率 p(A|特征 1)，属于 B 类的概率 p(B|特征 1)，属于 C 类的概率 p(C|特征 1)……）

3、分解待分类数据中的特征（特征 1、特征 2、特征 3、特征 4……）

4、计算各特征的各条件概率的乘积，如下所示：

判断为 A 类的概率：*p(A|特征 1)*p(A|特征 2)*p(A|特征 3)*p(A|特征 4)…..*

判断为 B 类的概率：*p(B|特征 1)*p(B|特征 2)*p(B|特征 3)*p(B|特征 4)…..*

判断为 C 类的概率：*p(C|特征 1)*p(C|特征 2)*p(C|特征 3)*p(C|特征 4)…..*

……

5、结果中的最大值就是该样本所属的类别

1.3.2 算法应用举例

大众点评、淘宝等电商上都会有大量的用户评论，比如：

1、衣服质量太差了！！！！颜色根本不纯！！！	*0*
2、我有一有种上当受骗的感觉！！！！	*0*
3、质量太差，衣服拿到手感觉像旧货！！！	*0*
4、上身漂亮，合身，很帅，给卖家点赞	*1*
5、穿上衣服帅呆了，给点一万个赞	*1*
6、我在他家买了三件衣服！！！！质量都很差！	*0*

其中 1/2/3/6 是差评，4/5 是好评
现在需要使用朴素贝叶斯分类算法来自动分类其他的评论，比如：

a、这么差的衣服以后再也不买了 *b、帅，有逼格* *……*

1.3.3 算法应用流程

1、分解出先验数据中的各特征
（即分词，比如“衣服”“质量太差”“差”“不纯”“帅”“漂亮”，“赞”……）
2、计算各类别（好评、差评）中，各特征的条件概率
（比如 p(“衣服”|差评)、p(“衣服”|好评)、p(“差”|好评) 、p(“差”|差评)……）
3、计算类别概率
P(好评|(c1,c2,c5,c8)) 的分子=p(“c1”|好评)*p(“c2”|好评)*……p(好评)
P(好评|(c1,c2,c5,c8)) 的分子= p(“c1”|差评)*p(“c2”|差评)*……p(差评)

p(好评 | (c1,c2,c5,c8)) = p(c1,c2,c5,c8 | 好评) p (好评) =p(c1|好评)p(c2|好评)p(c5|好评)p(c8|好评)p(好评)

p(差评 | (c1,c2,c5,c8))= p(c1,c2,c5,c8 | 差评) p (差评) =p(c1|差评)p(c2|差评)p(c5|差评)p(c8|差评)p(差评)

5、显然 P(差评)的结果值更大，因此 a 被判别为“差评”

2. 朴素贝叶斯分类算法案例 1

2.1 需求

客户评论分类：
以在线社区的留言板为例。为了不影响社区的发展，我们要屏蔽侮辱性的言论，所以要构建一个快速过滤器，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。过滤这类内容是一个很常见的需求。对此问题建立两个类别：侮辱类和非侮辱类，使用 1 和 0 分别标识。
有以下先验数据，使用 bayes 算法对未知类别数据分类

帖子内容	类别
'my','dog','has','flea','problems','help','please'	0

'maybe','not','take','him','to','dog','park','stupid'	1
'my','dalmation','is','so','cute','I','love','him'	0
'stop','posting','stupid','worthless','garbage'	1
'mr','licks','ate','my','steak','how','to','stop','him'	0
'quit','buying','worthless','dog','food','stupid'	1

待分类数据：

'love','my','dalmation'	?
'stupid','garbage'	?

p(好评|c1,c2,c3) --> p(c1,c2,c3|好评)p(好评)-->p(c1|好评)p(c2|好评)p(c3|好评)p(好评)

p(c1|好评)求解： c1 在样本好评中出现的总次数/先算出样本好评类的句子中总的词数
p(好评) 求解： 样本中好评的条数/样本的总条数

p(差评|c1,c2,c3) --> p(c1,c2,c3|差评)p(差评)-->p(c1|(差评)p(c2|好(差评)p(c3|(差评)p(差评)

2.2 模型分析

参见 1.3.2
跟 1.3.2 节中的举例基本一致，中文换成英文即可

2.2 Python 实现

(1) 词表到词向量的转换函数

```
from numpy import *
#过滤网站的恶意留言
# 创建一个实验样本
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
# 创建一个包含在所有文档中出现的不重复词的列表
def createVocabList(dataSet):
    vocabSet = set([])          #创建一个空集
    for document in dataSet:
        vocabSet = vocabSet | set(document)     #创建两个集合的并集
    return list(vocabSet)

#将文档词条转换成词向量
def setOfWords2Vec(vocabList, inputSet):
    returnVec = [0]*len(vocabList)              #创建一个其中所含元素都为 0 的向量
    for word in inputSet:
        if word in vocabList:
            #returnVec[vocabList.index(word)] = 1         #index 函数在字符串里找到字符第一次出现的位置  词集模型
            returnVec[vocabList.index(word)] += 1         #文档的词袋模型      每个单词可以出现多次
        else: print "the word: %s is not in my Vocabulary!" % word
    return returnVec
```

(2) 从词向量计算概率

```
#朴素贝叶斯分类器训练函数    从词向量计算概率
def trainNB0(trainMatrix, trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numTrainDocs)
   # p0Num = zeros(numWords); p1Num = zeros(numWords)
    #p0Denom = 0.0; p1Denom = 0.0
    p0Num = ones(numWords);    #避免一个概率值为 0,最后的乘积也为 0
    p1Num = ones(numWords);    #用来统计两类数据中，各词的词频
    p0Denom = 2.0;   #用于统计 0 类中的总数
    p1Denom = 2.0   #用于统计 1 类中的总数
    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 = p1Num / p1Denom
     #p0Vect = p0Num / p0Denom
     p1Vect = log(p1Num / p1Denom)      #在类 1 中，每个次的发生概率
     p0Vect = log(p0Num / p0Denom)         #避免下溢出或者浮点数舍入导致的错误     下溢出是由太多很小的数相乘得到的
     return p0Vect, p1Vect, pAbusive
```

(3) 根据现实情况修改分类器

注意：主要从以下两点对分类器进行修改

① 贝叶斯概率需要计算多个概率的乘积以获得文档属于某个类别的概率，即计算 p(w0|1)p(w1|1)p(w2|1)。如果其中一个概率值为 0，那么最后的乘积也为 0

② 第二个问题就是下溢出，这是由于太多过小的数相乘造成的。由于大部分因子都非常小，所以程序会下溢出或者得不到正确的答案。解决办法是对乘积取自然对数这样可以避免下溢出或者浮点数舍入导致的错误。

③ 每个单词的出现与否作为一个特征，被称为词集模型；在词袋模型中，每个单词可以出现多次。

```
#朴素贝叶斯分类器
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 testingNB():
    listOPosts, listClasses = loadDataSet()
    myVocabList = createVocabList(listOPosts)
    trainMat = []
    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
    p0V, p1V, pAb = trainNB0(array(trainMat), array(listClasses))
    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)
```

(4) 运行测试

```
>>>reload(bayes)
<module ‘bayes’ from ‘bayes.py’>
>>>bayes.testingNB()
['love','my','dalmation'] classified as: 0
['stupid','garbage'] classified as: 1
```

3、朴素贝叶斯分类算法案例 2

3.1 需求

利用大量邮件先验数据，使用朴素贝叶斯分类算法来自动识别垃圾邮件

3.2 python 实现

```
#过滤垃圾邮件
def textParse(bigString):          #正则表达式进行文本解析
    import re
    listOfTokens = re.split(r'\W*',bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def spamTest():
    docList = []; classList = []; fullText = []
    for i in range(1,26):                                    #导入并解析文本文件
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)
    trainingSet = range(50);testSet = []
    for i in range(10):                                    #随机构建训练集
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])       #随机挑选一个文档索引号放入测试集
        del(trainingSet[randIndex])                   #将该文档索引号从训练集中剔除
```

```
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:                    #对测试集进行分类
    wordVector = setOfWords2Vec(vocabList, docList[docIndex])
    if classifyNB(array(wordVector), p0V, p1V, pSpam) != classList[docIndex]:
        errorCount += 1
print 'the error rate is: ', float(errorCount)/len(testSet)
```