# 贝叶斯分类器原理及实践(python)

朴素贝叶斯分类器，及Narive Bayes,给定一个训练数据集，对新的输入例，在训练数据集上采用属性独立假设，并使用联合概率进行分类的计算器。

贝叶斯分类器的思想：

**** (1)

公式中c表示分类，比如商品分类，x表示特征或者属性，比如一个人的属性（身高，体重，爱好），根据一个人的属性向他进行商品的推荐，概率最大的商品作为首要推荐商品。

对x进行分解，变成多个属性的集合

** (2)**

式中表示针对分类c的每个离散属性为真的概率。

对离散属性而言，令表示Dc中第*i*个属性上取值为*xi*的样本组成的集合，则条件概率可以估计为，

** （3）**

分母**Dc表示分类(c)的样本数，而不是整体样本树数。**

如表1，推荐商品为啤酒与口红，1-13行为购买某个商品的用户特征与嗜好，根据这些信息，判断15-16行的用户应该推荐什么商品。

表1 购物网站用户与购买商品信息

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 性别 | 特征 | | | 嗜好 | | | |
| 上衣码 | 裤长 | 学号 | 抽烟 | 购物 | 追剧 | 电影 |
| 1 | 啤酒 | XXL | 105 | 42 | T | F | F | T |
| 2 | 啤酒 | XXXL | 95 | 42 | F | F | T | F |
| 3 | 啤酒 | XL | 90 | 41.5 | T | T | F | T |
| 4 | 啤酒 | XL | 90 | 40 | F | F | F | T |
| 5 | 啤酒 | S | 80 | 40 | T | T | F | T |
| 6 | 啤酒 | M | 90 | 40 | T | F | F | T |
| 7 | 啤酒 | M | 90 | 40 | F | F | F | T |
| 8- | 啤酒 | S | 85 | 39 | T | T | F | T |
| 9 | 口红 | XL | 90 | 39 | F | T | T | F |
| 10 | 口红 | S | 85 | 39 | F | T | T | F |
| 11 | 口红 | M | 85 | 38 | F | T | T | F |
| 12 | 口红 | M | 85 | 37.5 | T | F | T | F |
| 13 | 口红 | XXL | 95 | 40 | F | T | F | T |
| 14 | 口红 | M | 90 | 39 | F | F | T | T |
| 15 | ？ | S | 85 | 37 | F | T | F | F |
| 16 | ？ | XL | 85 | 38 | F | T | T | T |

分类用两种 ：啤酒 Bear,口红 Ls.

特征信息为：性别，上衣码，裤长，鞋号，抽烟，购物，追剧，电玩。

分类概率：

P(B)=7/12 啤酒分类的概率

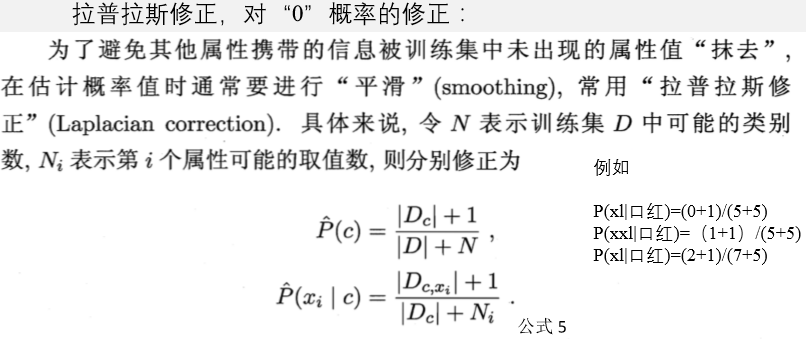
P(Ls)=5/12 口红分类的概率

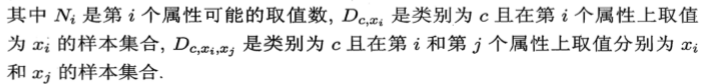
每个属性不同取值的概率

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 推荐类别 |  | 上衣码（5种） |  |  | 抽烟 | 购物 | 追剧 | 电玩 |  |
| 啤酒 | 7 | 0.143 | XXXL |  |  |  |  |  |  |
| 0.413 | XXL |  |  |  |  |  |  |
| 0.286 | XL |  |  |  |  |  |  |
| 0.286 | M |  | 0.571 | 0.286 | 0.143 | 0.857 | T |
| 0.143 | S |  | 0.429 | 0.714 | 0.857 | 0.143 | F |
| 口红 | 5 | 0.0 | XXXL |  |  |  |  |  |  |
| 0.2 | XXL |  |  |  |  |  |  |
| 0.0 | XL |  |  |  |  |  |  |
| 0.6 | M |  | 0.2 | 0.6 | 0.8 | 0.4 | T |
| 0.2 | S |  | 0.8 | 0.4 | 0.2 | 0.6 | F |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8 | 啤酒 | S | 85 | 39 | T | T | F | T |
| 9 | 口红 | XL | 90 | 39 | F | T | T | F |







判断15-16行的用户应该推荐什么商品的代码实现：

# coding=utf-8  
# from numpy import \*  
**import** numpy **as** np  
  
'''  
其中啤酒和口红，使用1和0分别表示。   
'''  
  
# 词表到向量的转换  
**def loadDataSet**():  
 postingList = [['XXL','105','42','T','F','F','T'],  
 ['XXXL','95','42','F','F','T','F'],  
 ['XL','90','41.5','T','T','F','T'],  
 ['XL','90','40','F','F','F','T'],  
 ['S','80','40','T','T','F','T'],  
 ['M','90','40','T','F','F','T'],  
 ['M','90','40','F','F','F','T'],  
 ['S','85','39','T','T','F','T'],  
 ['XL','90','39','T','T','F','T'],  
 ['S','85','39','F','T','T','F'],  
 ['M','85','38','F','T','T','F'],  
 ['M','85','37.5','T','F','T','F'],  
 ['XXL','95','40','F','T','F','T'],  
 ['M','90','39','F','F','T','T']  
 ]  
 classVec = [1,1,1,1,1,1,1,1,0,0,0,0,0,0]  
 **return** postingList, classVec  
  
**def creatVocabList**(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)  
  
**def setOfWordse2Ves**(vocabList, inputSet):  
 returnVec = [0] \* len(vocabList)  
 **for** word **in** inputSet:  
 **if** word **in** vocabList:  
 returnVec[vocabList.index(word)] +=1  
 **else**:  
 **print** 'the 特征:%s is not in my Vocabulary!' % word  
 **return** returnVec  
  
# 朴素贝叶斯分类器训练函数  
  
**def trainNB0**(trainMatrix, trainCategory):  
  
 numTrainDocs = len(trainMatrix)  
 numWords = len(trainMatrix[0])  
 pAbusive = sum(trainCategory) / float(numTrainDocs)  
 p0Num = np.ones(numWords)  
 p1Num = np.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 = np.log(p1Num / p1Denom)  
 p0Vect = np.log(p0Num / p0Denom)  
 **return** p0Vect, p1Vect, pAbusive  
  
# 朴素贝叶斯分类函数  
**def classifyNB**(vec2Classify, p0Vec, p1Vec, pClass1):  
 p1 = sum(vec2Classify \* p1Vec) + np.log(pClass1)  
 p0 = sum(vec2Classify \* p0Vec) + np.log(1.0 - pClass1)  
 **if** p1 > p0:  
 **return** '啤酒'  
 **else**:  
 **return** '口红'  
**def testingNB**(listOposts,listClasses,testEntry):  
 myVocabList = creatVocabList(listOposts)  
 trainMat = []  
 **for** postinDoc **in** listOposts:  
 trainMat.append(setOfWordse2Ves(myVocabList, postinDoc))  
 p0V, p1V, pAb = trainNB0(np.array(trainMat), np.array(listClasses))  
 thisDoc=np.array(setOfWordse2Ves(myVocabList,testEntry))  
 **print** testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb)  
  
  
  
#训练集  
listOposts, listClasses = loadDataSet()  
#测试  
testEntry = ['S','85','37','F','T','F','F']  
testingNB(listOposts,listClasses,testEntry)  
testEntry = ['S','85','38','F','T','T','T']  
testingNB(listOposts,listClasses,testEntry)

结果：

the 特征:37 is not in my Vocabulary!

15 ['S', '85', '37', 'F', 'T', 'F', 'F'] classified as: 口红

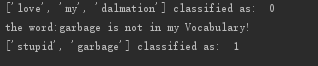
16 ['S', '85', '38', 'F', 'T', 'T', 'T'] classified as: 口红

实例1：使用Python进行文本分类

# coding=utf-8  
# from numpy import \*  
**import** numpy **as** np  
  
'''  
使用Python进行文本分类   
以在线社区的留言板为例,要屏蔽侮辱性的言论  
其中侮辱类和非侮辱类，使用1和0分别表示。   
'''  
  
# 词表到向量的转换

#将把文本看成单词向量或者词条向量，也就是说将句子转换为向量  
**def loadDataSet**():  
 postingList = [['my', 'do', 'has', 'flea', 'probleam', 'help', 'please'],  
 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupid'],  
 ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him'],  
 ['stop', 'posting', 'stupid', 'worthless', 'garbafge'],  
 ['mr', 'licks', 'ate', 'my', 'steak', 'hom', 'to', 'stop', 'him'],  
 ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']  
 ]  
 classVec = [0, 1, 0, 1, 0, 1]  
 **return** postingList, classVec  
  
**def creatVocabList**(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)  
  
**def setOfWordse2Ves**(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  
###########朴素贝叶斯词袋模型  
#它与函数setOfWords2Vec()几乎 完全相同，  
# 唯一不同的是每当遇到一个单词时，它会增加词向量中的对应值，  
# 而不只是将对应的 数值设为1  
**def bagOfWordse2Ves**(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  
  
# 朴素贝叶斯分类器训练函数  
  
**def trainNB0**(trainMatrix, trainCategory):  
  
 numTrainDocs = len(trainMatrix)  
 numWords = len(trainMatrix[0])  
 pAbusive = sum(trainCategory) / float(numTrainDocs)  
 # p0Num=np.zeros(numWords)  
 # p1Num=np.zeros(numWords)  
 # p0Denom=0.0  
 # p1Demon=0.0  
 # 根据现实情况修改分类器  
 p0Num = np.ones(numWords)  
 p1Num = np.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 = np.log(p1Num / p1Denom)  
 p0Vect = np.log(p0Num / p0Denom)  
 **return** p0Vect, p1Vect, pAbusive  
  
# 朴素贝叶斯分类函数  
  
**def classifyNB**(vec2Classify, p0Vec, p1Vec, pClass1):  
 p1 = sum(vec2Classify \* p1Vec) + np.log(pClass1)  
 p0 = sum(vec2Classify \* p0Vec) + np.log(1.0 - pClass1)  
 **if** p1 > p0:  
 **return** 1  
 **else**:  
 **return** 0  
  
**def testingNB**():  
 listOposts, listClasses = loadDataSet()  
 myVocabList = creatVocabList(listOposts)  
 **print** myVocabList  
 trainMat = []  
 **for** postinDoc **in** listOposts:  
 trainMat.append(setOfWordse2Ves(myVocabList, postinDoc))  
 p0V, p1V, pAb = trainNB0(np.array(trainMat), np.array(listClasses))  
 testEntry=['love','my','dalmation']  
 thisDoc=np.array(setOfWordse2Ves(myVocabList,testEntry))  
 **print** testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb)  
 testEntry=['stupid','garbage']  
 thisDoc=np.array(setOfWordse2Ves(myVocabList,testEntry))  
 **print** testEntry,'classified as: ',classifyNB(thisDoc,p0V,p1V,pAb)  
testingNB()

结果：



实例2：使用朴素贝叶斯过滤垃圾邮件

测试算法：使用朴素贝叶斯进行交叉验证

# coding=utf-8  
**import** re  
**from** bayes **import** \*  
**import** random  
  
# mySent='This book is the best book 4 , I have ever laid eyes upon.'  
# # regEx=re.compile('\\W\*')  
# # print regEx.split(mySent)  
# print re.split(r'\W\*',mySent)  
  
#文件解析及完整的垃圾邮件测试函数  
  
**def textParse**(bigString):  
 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):  
 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=creatVocabList(docList)  
 trainingSet=range(50)  
 testSet=[]  
 **for** i **in** range(10):  
 randIndex=int(random.uniform(0,len(trainingSet)))  
 testSet.append(trainingSet[randIndex])  
 **del**(trainingSet[randIndex])  
 trainMat=[]  
 trainClasses=[]  
 **for** docIndex **in** trainingSet:  
 trainMat.append(setOfWordse2Ves(vocabList,docList[docIndex]))  
 trainClasses.append(classList[docIndex])  
 p0v,p1v,pSpam=trainNB0(np.array(trainMat),np.array(trainClasses))  
 errorCount=0  
 **for** docIndex **in** testSet:  
 wordVector=setOfWordse2Ves(vocabList,docList[docIndex])  
 **if** classifyNB(np.array(wordVector),p0v,p1v,pSpam)!=classList[docIndex]:  
 errorCount+=1  
 **print** 'the error rate is: ',float(errorCount)/len(testSet)