参考网址如下，讲解更详细：

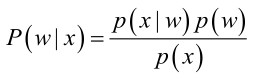
<http://www.jianshu.com/p/5fd446efefe9>

<http://blog.csdn.net/v_victor/article/details/51319873>

朴素navie：条件独立性

【问题的引入】

经典的贝叶斯公式



实际问题中，能获得的数据可能只有有限数目的样本数据，而先验概率和类条件概率(各类的总体分布) 都是未知的。解决：需要先对先验概率和类条件概率进行估计，然后再套用贝叶斯分类器。

**先验概率**的估计较简单：1、每个样本所属的自然状态都是已知的（有监督学习）；2、依靠经验；3、用训练样本中各类出现的频率估计。

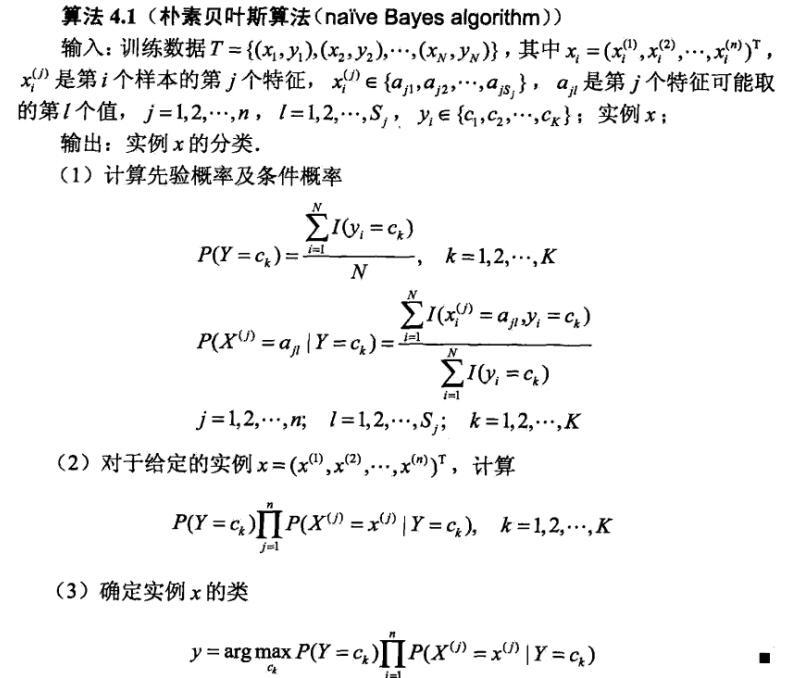
**类条件概率**的估计（非常难），原因包括：概率密度函数包含了一个随机变量的全部信息；样本数据可能不多；特征向量x的维度可能很大等。要直接估计类条件概率的密度函数很难。解决的办法：把估计完全未知的概率密度转化为估计参数（即转换成参数估计问题），极大似然估计就是一种参数估计方法。

关于**极大似然估计**的引入和解释：

http://blog.csdn.net/zengxiantao1994/article/details/72787849

http://www.cnblogs.com/hbwxcw/p/6824400.html

【算法】



给定的训练集是标定了 侮辱性/非侮辱性 的句子（因为是英语句子，所以基本视分词为已经解决的问题，如果是汉语，则要先进行分词），我们认为特征就是句子中的单个词语。单个词语有极性表征，整个句子所包含的单词的极性表征就是句子的极性。

由以上基础，应用朴素贝叶斯分类。

* 初始化：构建可以表征句子的特征向量（词汇表）。并根据这个特征向量，把训练集表征出来。从训练集中分离部分数据作为测试集。
* 学习：计算类的先验概率和特征向量对应每一类的条件概率向量。
* 分类：计算测试集中待分类句子在每一类的分类后验概率，取最大值作为其分类，并与给定标签比较，得到误分类率。

【代码】# 屏蔽社区留言板的侮辱性言论

1.初始化-社区留言板文本如下：

其中，我们对其进行负面和正面分类：0代表正类，1代表负类。

def loadDataSet():

"""

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 侮辱性文字 ， 0 代表正常言论

return postingList,classVec

2. 初始化-根据上述文本建立一个词汇表，即把重复的词汇剔除。

def createVocabList(dataSet):

"""

dataSet: 数据集，return: vocabList不含重复元素的单词列表

"""

vocabSet = set()

for data in dataSet:

vocabSet |= set(data)

return list(vocabSet)

3. 初始化-首先把留言转化为向量，如下所示：

**词汇表**: ['help', 'to', 'stupid', 'problem', 'garbage', 'love', 'licks', 'dog', 'stop', 'so', 'ny', 'take', 'is', 'food', 'worthless', 'park', 'flea', 'steak', 'has', 'dalmation', 'how', 'posting', 'please', 'my', 'maybe', 'buying', 'quit', 'ate', 'cute', 'mr', 'not', 'I', 'him']

**对于文本**：['my', 'dog', 'has', 'flea', 'problem', 'help', 'please']

**转化为向量**：[0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0]

def wordsToVector (vocabList, inputSet):

"""

vocabList词汇，inputSet: 为要转化为向量的输入文本

return: returnVec

"""

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 the vocabList!'%word)

return returnVec

4.先验概率和条件概率的求解

负类的**先验概率** = 负类的数量/留言的总数量

**条件概率**：计算每个单词分别在负类和正类条件下的概率

代码中各变量含义如下：

|  |  |  |
| --- | --- | --- |
| 输入 | trainset | 已转换为向量的输入，内容[[0,0,1,...],[1,0]...] |
| trainCategory | 输入文本所对应的类别 |
|  | num\_train | 训练集数目 |
| num\_vocab | 整个vocabList的长度 |
| 构造单词出现次数列表 | 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 | |
| pnorm\_vector | 正常的统计 |
| pabu\_vector | 侮辱的统计 |
| 整个数据集单词出现总数：2 | 根据样本/实际结果调整分母的值（2是避免分母为0，值可以调整） | |
| pnorm\_denom | 正常的统计 |
| pabu\_denom | 侮辱的统计 |
| 输出 | pAbusive | 负类的概率 = 负类的数量/训练集的数目 |
| pabu\_vector | 类别1下单词出现的条件概率，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)) ....]列表 |
| pnorm\_vector | 类别0下单词出现的条件概率，即正常文档的[log(P(F1|C0)),log(P(F2|C0)) ....]列表 |

def train\_bayes(trainset, trainCategory):

"""

先验概率和条件概率的求解

return:

pAbusive:负类的概率 = 负类的数量/训练集的数目

pabu\_vector: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

pnorm\_vector: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

"""

num\_train = len(trainset)

num\_vocab = len(trainset[0])

pAbusive = sum(trainCategory) / num\_train

#求条件概率：构造单词出现次数列表

pnorm\_vector = np.ones(num\_vocab)

pabu\_vector = np.ones(num\_vocab)

#初始化整个数据集单词出现总数2

pnorm\_denom = 2

pabu\_denom = 2

for i in range(num\_train):

if trainCategory[i] == 1:

pabu\_vector += trainset[i]

pabu\_denom += sum(trainset[i])

else:

pnorm\_vector += trainset[i]

pnorm\_denom += sum(trainset[i])

pabu\_vector = np.log(pabu\_vector / pabu\_denom)

pnorm\_vector = np.log(pnorm\_vector / pnorm\_denom)

return pAbusive,pabu\_vector,pnorm\_vector

5. 分类

|  |  |  |
| --- | --- | --- |
| 使用算法 | # 将乘法转换为加法 | 乘法：P(C|F1F2...Fn) = P(F1F2...Fn|C)P(C)/P(F1F2...Fn) |
| 加法：P(F1|C)\*P(F2|C)....P(Fn|C)P(C) ->  log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C)) |
| 输入 | testVector | 待测数据[0,1,1,1,1...]，即要分类的向量 |
| pnorm\_vector | 类别0下单词出现的条件概率，即正常文档的[log(P(F1|C0)),log(P(F2|C0)) ....]列表 |
| pabu\_vector | 类别1下单词出现的条件概率，即侮辱性文档的[log(P(F1|C1)),log(P(F2|C1)) ....]列表 |
| pabusive | 类别1，侮辱性文件的出现概率 |
| 计算公式 | | log(P(F1|C))+log(P(F2|C))+....+log(P(Fn|C))+log(P(C)) |
| 输出 | 类别1/0 | pabu = sum(testVector\*pabu\_vector) + np.log(pabusive) |

def classify(testVector, pnorm\_vector, pabu\_vector, pabusive):

"""

testVector:待测数据[0,1,1,1,1...],即要分类的向量

pnorm\_vector:类别0,pabu\_vector:类别1,pabusive: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))

testVector\*pnorm\_vector:单词在词汇表中的条件下，文件是good 类别的概率

"""

pabu = sum(testVector \* pabu\_vector) + np.log(pabusive)

pnorm = sum(testVector \* pnorm\_vector) + np.log(1-pabusive)

if pabu > pnorm:

return 1

else:

return 0

6.测试代码：

if \_\_name\_\_ == '\_\_main\_\_':

# 1. 加载数据集

posts\_List, classes\_list = loadDataSet()

# 2. 创建单词集合

vocab\_List = createVocabList(posts\_List)

# 3. 计算单词是否出现并创建数据矩阵

trainset = []

for post in posts\_List:

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

trainset.append(wordsToVector(vocab\_List, post))

# 4. 训练数据

pAbusive,pabu\_vector,pnorm\_vector = train\_bayes(trainset,classes\_list)

# 5. 测试数据

testEntry1 = ['love', 'my', 'dalmation']

testVector1 = wordsToVector(vocab\_List, testEntry1)

print (testEntry1, 'classified as: ', classify(testVector1, pnorm\_vector, pabu\_vector, pAbusive))

testEntry2 = ['stupid', 'garbage']

testVector2 = wordsToVector(vocab\_List, testEntry2)

print (testEntry2, 'classified as: ', classify(testVector2, pnorm\_vector, pabu\_vector, pAbusive))