第 7 章 朴素贝叶斯

内容提要

- 使用概率分布进行分类
- 学习朴素贝叶斯分类器

前面我们要求分类器做出艰难决策，给出“该数据实例属于哪一类”这类问题的明确答案。不过，分类器有时会产生错误结果，这时可以要求分类器给出一个最优的类别猜测结果，同时给出这个猜测的概率估计值。

概率论是许多机器学习算法的基础，所以深刻理解这一主题就显得十分重要。本章会给出一些使用概率论进行分类的方法。首先从一个最简单的概率分类器开始，然后给出一些假设来学习朴素贝叶斯分类器。我们称之为“朴素”，是因为整个形式化过程只做最原始、最简单的假设。我们将充分利用 Python 的文本处理能力将文档切分成词向量，然后利用词向量对文档进行分类。我们还将构建另一个分类器，观察其在真实的垃圾邮件数据集中的过滤效果。最后，我们将介绍如何从个人发布的大量广告中学习分类器，并将学习结果转换成人类可理解的信息。

假设现在我们有一个数据集，它由两类数据组成，我们现在用 $p_1(x,y)$ 表示数据点 (x,y) 属于类别 1 的概率，用 $p_2(x,y)$ 表示数据点 (x,y) 属于类别 2 的概率，那么对于一个新数据点 (x,y)，可以用下面的规则来判断它的类别：

- 如果 $p1(x,y) > p2(x,y)$，那么类别为 1。
- 如果 $p2(x,y) > p1(x,y)$，那么类别为 2。

也就是说，我们会选择高概率对应的类别。这就是贝叶斯决策理论的核心思想，即选择具有最高概率的决策。因此需要计算概率 p_1。

7.1 条件概率

假设现在有一个装了 7 块石头的罐子，其中 3 块是灰色的，4 块是黑色的（如图7.1所示）。如果从罐子中随机取出一块石头，那么是灰色石头的可能性是多少？由于取石头有 7 种可能，其中 3 种为灰色，所以取出灰色石头的概率为 3/7。那么取到黑色石头的概率又是多少呢？很显然，是 4/7。我们使用 P(gray) 来表示取到灰色石头的概率，其概率值可以通过灰色石头数目除以总的石头数目来得到。

如果这 7 块石头如图7.2所示放在两个桶中，那么上述概率应该如何计算？

要计算 P(gray) 或者 P(black)，事先得知道石头所在桶的信息会不会改变结果？你有可能已经想到计算从 B 桶中取到灰色石头的概率的办法，这就是所谓的条件概率（conditional probability）。假定计算的是从 B 桶取到灰色石头的概率，这个概率可以记作 P(gray|bucketB)，我们称之为“在已知石头出自 B 桶的条件下，取出灰色石头的概率”。不难得到，P(gray|bucketA) 值为 2/4，P(gray|bucketB) 的值为 1/3。

条件概率的计算公式如下所示：

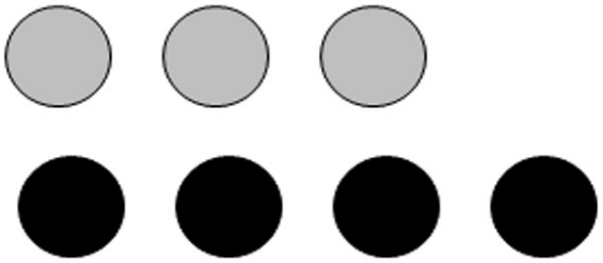

图 7.1: 一个包含 7 块石头的集合，石头的颜色为灰色或者黑色。如果随机从中取一块石头，那么取到灰色石头的概率为 3/7。类似地，取到黑色石头的概率为 4/7。

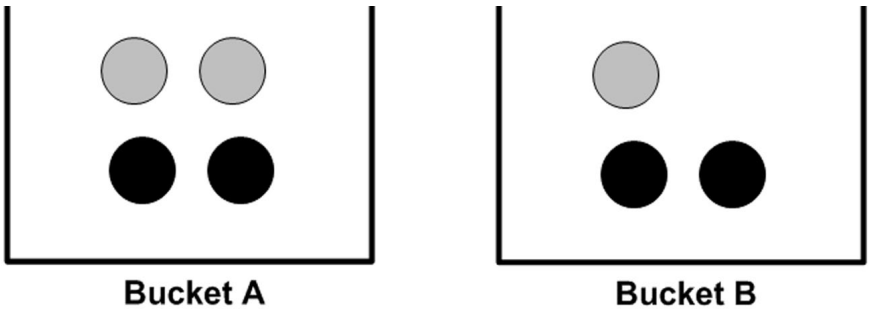

图 7.2: 落到两个桶中的 7 块石头。

$$P(gray|bucketB) = P(gray\ and\ bucketB)/P(bucketB) \tag{7.1}$$

我们来看看上述公式是否合理。首先，用 B 桶中灰色石头的个数除以两个桶中总的石头数，得到 P(gray and bucketB) = 1/7。其次，由于 B 桶中有 3 块石头，而总石头数为 7，于是 P(bucketB) 就等于 3/7。于是有 P(gray|bucketB) = P(gray and bucketB)/P(bucketB) = (1/7) / (3/7) = 1/3。这个公式虽然对于这个简单例子来说有点复杂，但当存在更多特征时是非常有效的。用代数方法计算条件概率时，该公式也很有用。

另一种有效计算条件概率的方法称为贝叶斯准则。贝叶斯准则告诉我们如何交换条件概率中的条件与结果，即如果已知 $P(x|c)$，要求 $P(c|x)$，那么可以使用下面的计算方法：

$$P(c|x) = \frac{P(x|c)p(c)}{p(x)} \tag{7.2}$$

接下来的问题是如何将条件概率应用到分类器中。

7.2 使用条件概率来分类

贝叶斯决策理论要求计算两个概率 $p1(x, y)$ 和 $p2(x, y)$：

- 如果 $p1(x, y) > p2(x, y)$，那么类别为 1。
- 如果 $p2(x, y) > p1(x, y)$，那么类别为 2。

但这两个准则并不是贝叶斯决策理论的所有内容。使用 $p1()$ 和 $p2()$ 只是为了尽可能简化描述，而真正需要计算和比较的是 $p(c_1|x, y)$ 和 $p(c_2|x, y)$。这些符号所代表的具体意义是：给定某个由 x、y 表示的数据点，那么该数据点来自类别 c1 的概率是多少？数据点来自类别 c2 的概率又是多少？注意这些概率与刚才给出的概率 p(x, y|c1) 并不一样，不过可以使用贝叶斯

准则来交换概率中条件与结果。具体地，应用贝叶斯准则得到：

$$P(c_i|x,y) = \frac{P(x,y|c_i)p(c_i)}{p(x,y)} \tag{7.3}$$

使用这些定义，可以定义贝叶斯分类准则为：

- 如果 $p(c_1|x,y) > p(c_2|x,y)$，那么类别为 1。
- 如果 $p(c_2|x,y) > p(c_1|x,y)$，那么类别为 2。

使用贝叶斯准则，可以通过已知的三个概率值来计算未知的概率值。后面就会给出利用贝叶斯准则来计算概率并对数据进行分类的代码。

7.3 使用朴素贝叶斯进行文档分类

机器学习的一个重要应用就是文档的自动分类。在文档分类中，整个文档（如一封电子邮件）是实例，而电子邮件中的某些元素则构成特征。虽然电子邮件是一种会不断增加的文本，但我们同样也可以对新闻报道、用户留言、政府公文等其他任意类型的文本进行分类。我们可以观察文档中出现的词，并把每个词的出现或者不出现作为一个特征，这样得到的特征数目就会跟词汇表中的词目一样多。朴素贝叶斯是上节介绍的贝叶斯分类器的一个扩展，是用于文档分类的常用算法。

假设词汇表中有 1000 个单词。要得到好的概率分布，就需要足够的数据样本，假定样本数为 N。前面讲到的约会网站示例中有 1000 个实例，手写识别示例中每个数字有 200 个样本，而决策树示例中有 24 个样本。其中，24 个样本有点少，200 个样本好一些，而 1000 个样本就非常好了。约会网站例子中有三个特征。由统计学知，如果每个特征需要 N 个样本，那么对于 10 个特征将需要 N^{10} 个样本，对于包含 1000 个特征的词汇表将需要 N^{1000} 个样本。可以看到，所需要的样本数会随着特征数目增大而迅速增长。

如果特征之间相互独立，那么样本数就可以从 N^{1000} 减少到 $1000N$。所谓独立（independence）指的是统计意义上的独立，即一个特征或者单词出现的可能性与它和其他单词相邻没有关系。举个例子讲，假设单词 bacon 出现在 unhealthy 后面与出现在 delicious 后面的概率相同。当然，我们知道这种假设并不正确，bacon 常常出现在 delicious 附近，而很少出现在 unhealthy 附近，这个假设正是朴素贝叶斯分类器中**朴素**（naive）一词的含义。朴素贝叶斯分类器中的另一个假设是，每个特征同等重要。其实这个假设也有问题。如果要判断留言板的留言是否得当，那么可能不需要看完所有的 1000 个单词，而只需要看 10~20 个特征就足以做出判断了。尽管上述假设存在一些小的瑕疵，但朴素贝叶斯的实际效果却很好。

7.4 使用 Python 进行文本分类

要从文本中获取特征，需要先拆分文本。具体如何做呢？这里的特征是来自文本的词条（token），一个词条是字符的任意组合。可以把词条想象为单词，也可以使用非单词词条，如 URL、IP 地址或者任意其他字符串。然后将每一个文本片段表示为一个词条向量，其中值为 1 表示词条出现在文档中，0 表示词条未出现。

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 分别表示。接下来首先给出将文本转换为数字向量的过程，然后介绍如何基于这些向量来计算条件概率，并在此基础上构建分类器，最后还要介绍一些利用 Python 实现朴素贝叶斯过程中需要考虑的问题。

7.4.1 准备数据：从文本中构建词向量

我们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。考虑出现在所有文档中的所有单词，再决定将哪些词纳入词汇表或者说所要的词汇集合，然后必须要将每一篇文档转换为词汇表上的向量。

词表到向量的转换函数:

```
'''
Parameters:
    无
Returns:
    postingList - 实验样本切分的词条
    classVec - 类别标签向量
'''
# 函数说明:创建实验样本
def loadDataSet():
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'], #切分的词条
                 ['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]#类别标签向量，1代表侮辱性词汇，0代表不是
    return postingList,classVec

'''
Parameters:
    vocabList - createVocabList返回的列表
    inputSet - 切分的词条列表
Returns:
    returnVec - 文档向量,词集模型
'''
# 函数说明:根据vocabList词汇表，将inputSet向量化，向量的每个元素为1或0
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:                                   #遍历每个词条
        if word in vocabList:                               #如果词条存在于词汇表中，则置1
            returnVec[vocabList.index(word)] = 1
        else: print("the word: %s is not in my Vocabulary!" % word)
    return returnVec                                        #返回文档向量
```

```
'''
Parameters:
    dataSet - 整理的样本数据集
Returns:
    vocabSet - 返回不重复的词条列表，也就是词汇表
'''
# 函数说明:将切分的实验样本词条整理成不重复的词条列表，也就是词汇表
def createVocabList(dataSet):
    vocabSet = set([])                    #创建一个空的不重复列表
    for document in dataSet:
        vocabSet = vocabSet | set(document) #取并集
    return list(vocabSet)
```

第一个函数 loadDataSet() 创建了一些实验样本。该函数返回的第一个变量是进行词条切分后的文档集合，这些文档来自斑点犬爱好者留言板。这些留言文本被切分成一系列的词条集合，标点符号从文本中去掉，后面会探讨文本处理的细节。loadDataSet( ) 函数返回的第二个变量是一个类别标签的集合。这里有两类，侮辱性和非侮辱性。这些文本的类别由人工标注，这些标注信息用于训练程序以便自动检测侮辱性留言。下一个函数 createVocabList() 会创建一个包含在所有文档中出现的不重复词的列表，为此使用了 Python 的 set 数据类型。将词条列表输给 set 构造函数，set 就会返回一个不重复词表。首先，创建一个空集合，然后将每篇文档返回的新词集合添加到该集合中。操作符 | 用于求两个集合的并集，这也是一个按位或（OR）操作符。在数学符号表示上，按位或操作与集合求并操作使用相同记号。

获得词汇表后，便可以使用函数 setOfWords2Vec()，该函数的输入参数为词汇表及某个文档，输出的是文档向量，向量的每一元素为 1 或 0，分别表示词汇表中的单词在输入文档中是否出现。函数首先创建一个和词汇表等长的向量，并将其元素都设置为 0 。接着，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。一切都顺利的话，就不需要检查某个词是否还在 vocabList 中，后边可能会用到这一操作。

现在看一下这些函数的执行效果：

```
postingList, classVec = loadDataSet()
print('postingList:\n',postingList)
myVocabList = createVocabList(postingList)
print('myVocabList:\n',myVocabList)
trainMat = []
for postinDoc in postingList:
    trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
print('trainMat:\n', trainMat)
```

7.4.2 训练算法：从词向量计算概率

前面介绍了如何将一组单词转换为一组数字，接下来看看如何使用这些数字计算概率。现在已经知道一个词是否出现在一篇文档中，也知道该文档所属的类别。我们重写贝叶斯准则，将之前的 x、y 替换为 $\mathbf{w}$。$\mathbf{w}$ 表示这是一个向量，即它由多个数值组成。在这个例子中，数值

个数与词汇表中的词个数相同。

$$P(c_i|\mathbf{w}) = \frac{P(\mathbf{w}|c_i)p(c_i)}{p(\mathbf{w})} \tag{7.4}$$

我们将使用上述公式，对每个类计算该值，然后比较这两个概率值的大小。如何计算呢？首先可以通过类别 i（侮辱性留言或非侮辱性留言）中文档数除以总的文档数来计算概率 $p(c_i)$。接下来计算 $p(w|c_i)$，这里就要用到朴素贝叶斯假设。如果将 w 展开为一个个独立特征，那么就可以将上述概率写作 $p(w_0, w_1, w_2..w_N|c_i)$。这里假设所有词都互相独立，该假设也称作条件独立性假设，它意味着可以使用 $p(w_0|c_i)p(w_1|c_i)p(w_2|c_i)...p(w_N|c_i)$ 来计算上述概率，这就极大地简化了计算的过程。

该函数的伪代码如下：

```
计算每个类别中的文档数目
对每篇训练文档:
    对每个类别:
        如果词条出现在文档中 增加该词条的计数值
        增加所有词条的计数值
    对每个类别:
        对每个词条:
            将该词条的数目除以总词条数目得到条件概率
    返回每个类别的条件概率
```

朴素贝叶斯分类器训练函数

```
'''
Parameters:
    trainMatrix - 训练文档矩阵，即setOfWords2Vec返回的returnVec构成的矩阵
    trainCategory - 训练类别标签向量，即loadDataSet返回的classVec
Returns:
    p0Vect - 侮辱类的条件概率数组
    p1Vect - 非侮辱类的条件概率数组
    pAbusive - 文档属于侮辱类的概率
'''
# 函数说明:朴素贝叶斯分类器训练函数
def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)                    #计算训练的文档数目
    numWords = len(trainMatrix[0])                     #计算每篇文档的词条数
    pAbusive = sum(trainCategory)/float(numTrainDocs) #文档属于侮辱类的概率
    p0Num = np.zeros(numWords); p1Num = np.zeros(numWords)#创建numpy.zeros数组,词条出现数初始化为0
    p0Denom = 0.0; p1Denom = 0.0 #分母初始化为0
    for i in range(numTrainDocs):
        if trainCategory[i] == 1: #统计属于侮辱类的条件概率所需的数据，即P(w0|1),P(w1|1),P(w2|1)···
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:                     #统计属于非侮辱类的条件概率所需的数据，即P(w0|0),P(w1|0),P(w2|0)···
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = p1Num/p1Denom
```

```
    p0Vect = p0Num/p0Denom
    return p0Vect,p1Vect,pAbusive#返回属于侮辱类的条件概率数组，属于非侮辱类的条件概率数组，文档属于侮辱类
        的概率。
```

代码函数中的输入参数为文档矩阵 trainMatrix，以及由每篇文档类别标签所构成的向量 trainCategory。首先，计算文档属于侮辱性文档（class=1）的概率，即 $P(1)$。因为这是一个二类分类问题，所以可以通过 $1-P(1)$ 得到 $P(0)$。对于多于两类的分类问题，则需要对代码稍加修改。计算 $p(w_i|c_1)$ 和 $p(w_i|c_0)$，需要初始化程序中的分子变量和分母变量。由于 w 中元素如此众多，因此可以使用 NumPy 数组快速计算这些值。上述程序中的分母变量是一个元素个数等于词汇表大小的 NumPy 数组。在 for 循环中，要遍历训练集 trainMatrix 中的所有文档。一旦某个词语（侮辱性或正常词语）在某一文档中出现，则该词对应的个数（p1Num 或者 p0Num）就加 1，而且在所有的文档中，该文档的总词数也相应加 1 。对于两个类别都要进行同样的计算处理。最后，对每个元素除以该类别中的总词数。利用 NumPy 可以很好实现，用一个数组除以浮点数即可，若使用常规的 Python 列表则难以完成这种任务，读者可以自己尝试一下。最后，函数会返回两个向量和一个概率。

```
postingList, classVec = loadDataSet()
myVocabList = createVocabList(postingList)
print('myVocabList:\n', myVocabList)
trainMat = []
for postinDoc in postingList:
    trainMat.append(setOfWords2Vec(myVocabList, postinDoc))
p0V, p1V, pAb = trainNB0(trainMat, classVec)
print('p0V:\n', p0V)
print('p1V:\n', p1V)
print('classVec:\n', classVec)
print('pAb:\n', pAb)
```

首先，我们发现文档属于侮辱类的概率 pAb 为 0.5，该值是正确的。接下来，看一看在给定文档类别条件下词汇表中单词的出现概率，看看是否正确。词汇表中的第一个词是 cute，其在类别 0 中出现 1 次，而在类别 1 中从未出现。对应的条件概率分别为 0.041 666 67 与 0.0。该计算是正确的。我们找找所有概率中的最大值，该值出现在 P(1) 数组第 26 个下标位置，大小为 0.157 894 74。在 myVocabList 的第 26 个下标位置上可以查到该单词是 stupid。这意味着 stupid 是最能表征类别 1（侮辱性文档类）的单词。使用该函数进行分类之前，还需解决函数中的一些缺陷。

7.4.3 测试算法：根据现实情况修改分类器

利用贝叶斯分类器对文档进行分类时，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 $p(w_0|1)p(w_1|1)p(w_2|1)$。如果其中一个概率值为 0，那么最后的乘积也为 0。为降低这种影响，可以将所有词的出现数初始化为 1，并将分母初始化为 2。

将 trainNB0() 进行改进：

```
'''
Parameters:
```

```
    trainMatrix - 训练文档矩阵，即setOfWords2Vec返回的returnVec构成的矩阵
    trainCategory - 训练类别标签向量，即loadDataSet返回的classVec
Returns:
    p0Vect - 侮辱类的条件概率数组
    p1Vect - 非侮辱类的条件概率数组
    pAbusive - 文档属于侮辱类的概率
'''
# 函数说明:朴素贝叶斯分类器训练函数
def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)                 #计算训练的文档数目
    numWords = len(trainMatrix[0])                  #计算每篇文档的词条数
    pAbusive = sum(trainCategory)/float(numTrainDocs) #文档属于侮辱类的概率
    p0Num = np.ones(numWords); p1Num = np.ones(numWords)#创建numpy.ones数组,词条出现数初始化为1，拉普拉
        斯平滑
    p0Denom = 2.0; p1Denom = 2.0                    #分母初始化为2,拉普拉斯平滑
    for i in range(numTrainDocs):
        if trainCategory[i] == 1:#统计属于侮辱类的条件概率所需的数据，即P(w0|1),P(w1|1),P(w2|1)···
            p1Num += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:                   #统计属于非侮辱类的条件概率所需的数据，即P(w0|0),P(w1|0),P(w2|0)···
            p0Num += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = np.log(p1Num/p1Denom)                  #取对数，防止下溢出
    p0Vect = np.log(p0Num/p0Denom)
    #返回属于侮辱类的条件概率数组，属于非侮辱类的条件概率数组，文档属于侮辱类的概率
    return p0Vect,p1Vect,pAbusive
```

另一个遇到的问题是下溢出,这是由于太多很小的数相乘造成的。当计算乘积$p(w_0|c_i)p(w_1|c_i)p(w_2|c_i)...p$时，由于大部分因子都非常小，所以程序会下溢出或者得到不正确的答案。一种解决办法是对乘积取自然对数。在代数中有 $\ln(a \times b) = \ln(a) + \ln(b)$，于是通过求对数可以避免下溢出或者浮点数舍入导致的错误。同时，采用自然对数进行处理不会有任何损失

通过修改 return 前的两行代码:

```
p1Vect = np.log(p1Num/p1Denom) #取对数，防止下溢出
p0Vect = np.log(p0Num/p0Denom)
```

朴素贝叶斯分类函数:

```
'''
Parameters:
    vec2Classify - 待分类的词条数组
    p0Vec - 侮辱类的条件概率数组
    p1Vec -非侮辱类的条件概率数组
    pClass1 - 文档属于侮辱类的概率
Returns:
    0 - 属于非侮辱类
    1 - 属于侮辱类
'''
# 函数说明:朴素贝叶斯分类器分类函数
```

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + np.log(pClass1)   #对应元素相乘。logA * B = logA + logB,
                              #所以这里加上log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + np.log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0
```

7.4.4 准备数据：文档词袋模型

目前为止，我们将每个词的出现与否作为一个特征，这可以被描述为词集模型（set-of-words model）。如果一个词在文档中出现不止一次，这可能意味着包含该词是否出现在文档中所不能表达的某种信息，这种方法被称为词袋模型（bag-of-words model）。在词袋中，每个单词可以出现多次，而在词集中，每个词只能出现一次。为适应词袋模型，需要对函数 setOfWords2Vec() 稍加修改，修改后的函数称为 bagOfWords2Vec()。

下面的程序清单给出了基于词袋模型的朴素贝叶斯代码。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。

朴素贝叶斯词袋模型:

```
'''
Parameters:
    vocabList - createVocabList返回的列表
    inputSet - 切分的词条列表
Returns:
    returnVec - 文档向量,词袋模型
'''
# 函数说明:根据vocabList词汇表，构建词袋模型
def bagOfWords2VecMN(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:                             #遍历每个词条
        if word in vocabList:                         #如果词条存在于词汇表中，则计数加一
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

第 7 章 练习

1. 示例：使用朴素贝叶斯过滤垃圾邮件

   从学在浙大上下载数据。

   (a). 收集数据：提供文本文件。

   (b). 准备数据：将文本文件解析成词条向量。

   (c). 分析数据：检查词条确保解析的正确性。

   (d). 训练算法：使用我们之前建立的 trainNB0() 函数。

   (e). 测试算法：使用 classifyNB()，并且构建一个新的测试函数来计算文档集的错误率。

(f). 使用算法：构建一个完整的程序对一组文档进行分类，将错分的文档输出到屏幕上。代码请独立实现。

7.5 SKlearn 实现朴素贝叶斯

我们使用威斯康乳腺癌数据集比较朴素贝叶斯和逻辑回归分类器的性能。

威斯康乳腺癌数据集使用 30 个描述细胞核的实值特征将肿块分类为恶性或者良性。该数据集包括 212 个恶性实例和 357 个良性实例。

```
%matplotlib inline

import pandas as pd
from sklearn.datasets import load_breast_cancer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

X, y = load_breast_cancer(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=11)

lr = LogisticRegression()
nb = GaussianNB()

lr_scores = []
nb_scores = []

train_sizes = range(10, len(X_train), 10)

for train_size in train_sizes:
    X_slice, _, y_slice, _ = train_test_split(
        X_train, y_train, train_size=train_size, stratify=y_train, random_state=11)
    nb.fit(X_slice, y_slice)
    nb_scores.append(nb.score(X_test, y_test))
    lr.fit(X_slice, y_slice)
    lr_scores.append(lr.score(X_test, y_test))

plt.plot(train_sizes, nb_scores, label='Naive Bayes')
plt.plot(train_sizes, lr_scores, linestyle='--', label='Logistic Regression')
plt.title("Naive Bayes and Logistic Regression Accuracies")
plt.xlabel("Number of training instances")
plt.ylabel("Test set accuracy")
plt.legend()
```

第 7 章 练习

选做利用朴素贝叶斯和逻辑回归模型对皮马印第安人糖尿病数据集进行分类。数据集从学在浙大上下载。**代码独立编写**。

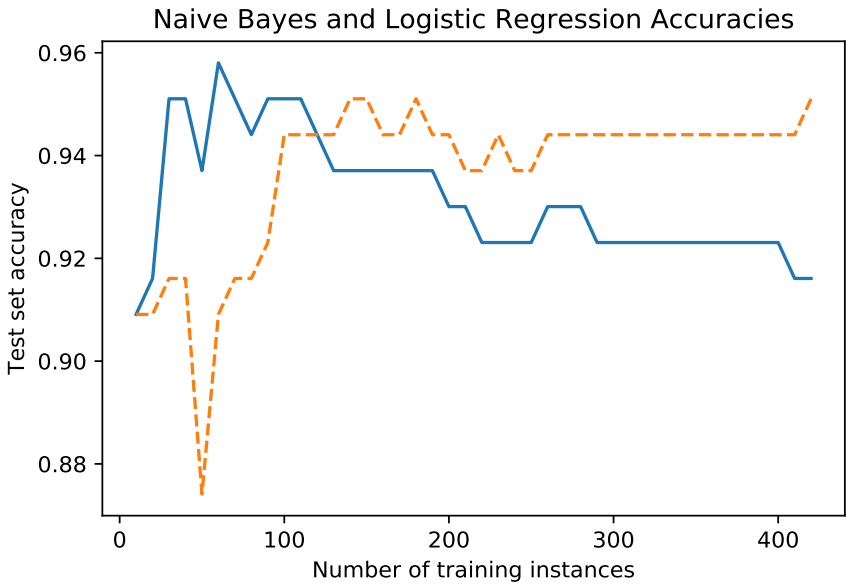

图 7.3: 乳腺癌数据集分类结果。