实验一　贝叶斯分类器的设计及应用实验

实验目标：理解朴素贝叶斯分类器的原理；
能独立实现常用贝叶斯分类器的设计；
准确评估分类器精度。

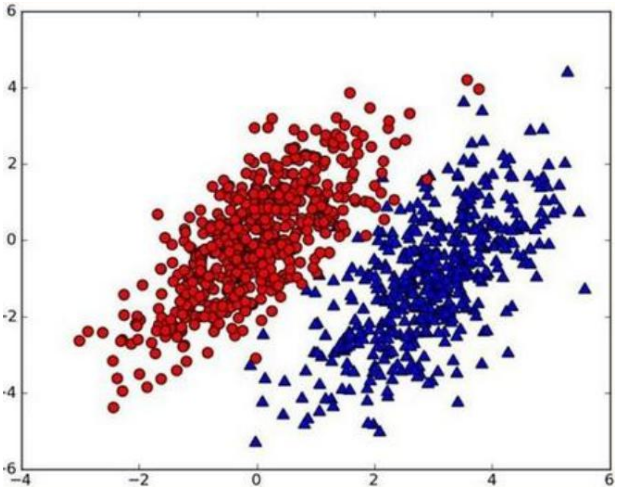
实验工具：Python(推荐) 或 C/C++

实验步骤：

一、朴素贝叶斯分类算法原理理解

1. 贝叶斯决策理论

假设有一个数据集，由两类组成（简化问题），对于每个样本的分类，如下图(from MLiA):



现有一个新的点 new_point (X,Y)，其分类未知。我们用 p1(X,Y)表示数据点 x (X,Y)属于红色一类的概率，用 p2(X,Y)表示数据点(X,Y)属于蓝色一类的概率。定制规则：

- p1(X,Y) > p2(X,Y)，则(X,Y)为红色一类。
- p1(X,Y) <p2(X,Y)，则(X,Y)为蓝色一类。

即：选择概率高的一类作为新点的分类。

用等价的条件概率的方式定义这一贝叶斯分类准则：

- p(red| x) > p(blue| x)，则(X,Y)属于红色一类。
- p(red| x) < p(blue| x)，则(X,Y)属于蓝色一类。

将 p(red| x)对应 P(c_1| x)，将 p(blue| x)对应 P(c_2| x)，贝叶斯公式如下：

$$P(c_1 \mid x) = \frac{P(x \mid c_1) * P(c_1)}{P(x)}$$

$$P(c_2 \mid x) = \frac{P(x \mid c_2) * P(c_2)}{P(x)}$$

出现一个需要分类的新点 x 时，我们只要计算 max (P(c_1 | x),P(c_2 | x),P(c_3 | x),...,P(c_n | x))。

2. 朴素贝叶斯分类

朴素贝叶斯分类的正式定义如下：

（1）设 $x=\{a_1,a_2,\ldots,a_m\}$ 表示一个待分类项，而每个 a_i 为 x 的一个特征属性。

（2）有类别集合 $C=\{y_1,y_2,\ldots,y_n\}$。

（3）计算 $P(y_1|x),P(y_2|x),\ldots,P(y_n|x)$。

（4）如果 $P(y_k|x)=\max\{P(y_1|x),P(y_2|x),\ldots P(y_n|x)\}$，则 $x\in y_k$。

其中，$P(y_i|x)=\dfrac{P(x|y_i)P(y_i)}{P(x)}$

朴素贝叶斯一个很强的条件独立假设：当 y_i 确定时，的各个特征之间相互独立。

即：$P(x|y_i)P(y_i)=P(a_1|y_i)P(a_2|y_i)\ldots P(a_m|y_i)P(y_i)=P(y_i)\prod_{j=1}^{m}P(a_j|y_i)$

分类中，我们只要将 $P(y_i)\prod_{j=1}^{m}P(a_j|y_i)$ 最大化即可。

二、使用朴素贝叶斯进行文本分类（词集模型）

文本中的特征是来自文本的词条（token），一个词条是字符的任意组合。

1. 准备数据：从文本中构建词向量

确定将哪些词纳入词汇集合，然后将每一篇输入文本转换为与词汇表同维度的向量。

1）创建实验样本，这些文本被切分成一系列词条集合，将标点符号从文本中去除。另外返回类别标签的集合，代表侮辱性和非侮辱性。

```
#创建一些实验样本，返回词条切分后的文档集合和类别标签的集合。

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]  #人工标注，1代表侮辱性词汇，0代表不是
    return postingList, classVec
```

2）优化词条列表，形成不重复词的列表。

```
# 创建词汇表，词集模型，即包含所有词的列表
def createVocabList(dataSet):
    vocab_set = set([])
    for document in dataSet:
        vocab_set = vocab_set | set(document)
    return list(vocab_set)
```

3）结合词汇表，将输入文档转换为文档向量。

```
# 返回值类似[1, 0, 1, 0, 0, 0,...]
def setWordsToVec(vocabList, inputSet):
    returnVec = [0] * len(vocabList)
    for word in inputSet:  # 遍历输入的每个词条
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else:
            print("the word: %s is not in my Vocabulary!" % word)
    return returnVec  # 返回文档向量
```

2. 训练算法：从词向量计算概率

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（for 循环或者矩阵相加）
        增加所有词条的计数值
    对每个类别：
        对每个词条：
            该词条的数目/总词条数目=条件概率（P(词条|类别)）
    返回每个类别的条件概率（P(类别|文档的所有词条)）

```
# 训练算法：从词向量计算概率
def trainNB0(trainMatrix, trainCategory):
    """
    :param trainMatrix: 文件单词矩阵 [[1,0,1,1,1....],[],[]...]
    :param trainCategory: 文件对应的标签类别[0,1,1,0....]，1代表对应的文件是侮辱性，0代表不是侮辱性
    :return:
        p0Vect:    各单词在分类0的条件下出现的概率
        p1Vect:    各单词在分类1的条件下出现的概率
        pAbusive:    文档属于分类1的概率
    """
    numTrainDocs = len(trainMatrix) # 文件数
    numWords = len(trainMatrix[0])  # 单词数

    pAbusive = sum(trainCategory) / float(numTrainDocs)#初始化概率
    p0Num = np.zeros(numWords); p1Num =np.zeros(numWords)
    p0Denom = 0.0;p1Denom = 0.0
    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 = p1Num / p1Denom
    p0Vect = p0Num / p0Denom
    return p0Vect, p1Vect, pAbusive
```

3. 测试算法：修改分类器

1）将所有词的出现数初始化为 1，将分母初始化为 2。将 trainNB0（）的第 4 行和第 5 行修改如下：

```
"""
 pAbusive = sum(trainCategory) / float(numTrainDocs)#初始化概率
 p0Num = np.zeros(numWords); p1Num =np.zeros(numWords)
 p0Denom = 0.0;p1Denom = 0.0
 """
 p0Num = np.ones(numWords); p1Num =np.ones(numWords)
 p0Denom =2.0;p1Denom = 2.0
```

2）将 return 的前两行代码做如下修改：

```
"""
 p1Vect = p1Num / p1Denom
 p0Vect = p0Num / p0Denom
"""
 p1Vect = p1Num / p1Denom
 p0Vect = p0Num / p0Denom
```

4. 朴素贝叶斯分类函数

```
def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    '''
    :param vec2Classify: 测试文档向量
    :param p0Vec: 在非侮辱性文档中， 词汇表的单词的概率
    :param p1Vec:
    :param pClass1: 侮辱性的先验概率
    :return:
    '''
    # 做对数映射
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0
```

三、使用朴素贝叶斯进行文本分类（词袋模型）

词袋模型中，每个单词可以出现多次，当遇到一个单词时，就会增加词向量中的对应值，而不只将对一个数值设为 1。

其中，词集模型的 setOfWords2Vec( )被替换为 bagOfWords2Vec( )，详细代码如下：

```
#词袋模型示例
def bagOfWords2VecMN(vocabList, inputSet):
    '''
    :param vocabList:  词汇列表
    :param inputSet:   文档
    :return:    词汇向量
    '''
    returnVec = [0] * len(vocabList)
    for word in inputSet:
        if word in vocabList:
            # 此处与词集模型的区别： 记录每个单词出现的次数， 而不是出现与否
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

四、实验要求

1. 将上面的词集模型代码整合，并自己编写文本切分和测试程序，分类下列文本：

“It’s worthless to stop my dog eating food.”

“Please help me to solve this problem!”

“This dog is so stupid，But that one is so cute.”

将输出结果截图并解释提交，说明是否分类准确？为什么？

2. 上面的第 3 步骤，为什么要对分类器进行两个修改？

3. 试用词袋模型对实验要求 1 中的三句话进行分类，分析结果，并说明代码中做了哪些具体改动？

注意：请将实验报告以文件形式提交到 QQ 群里，统一命名为：学号+姓名+实验几.doc/rar/zip/pdf