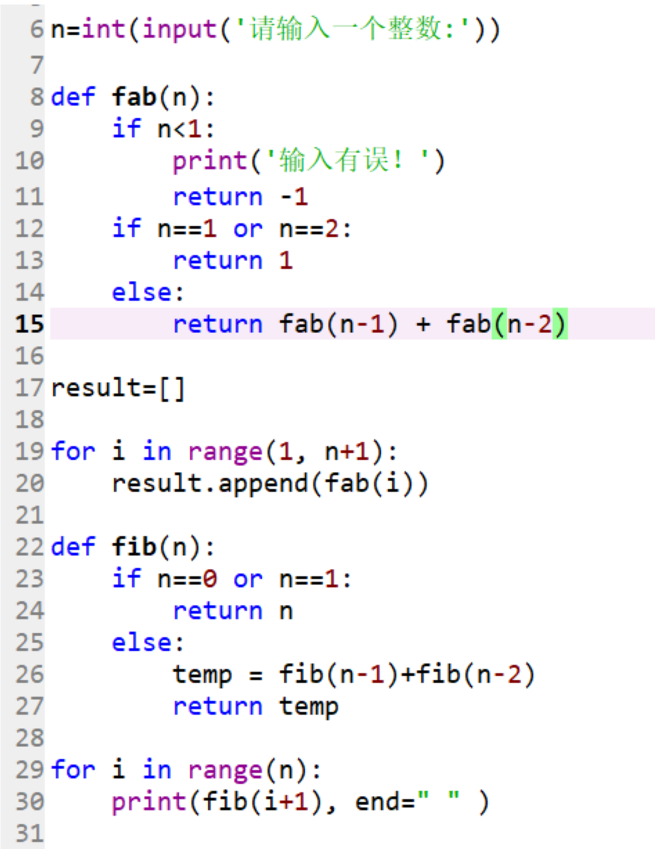
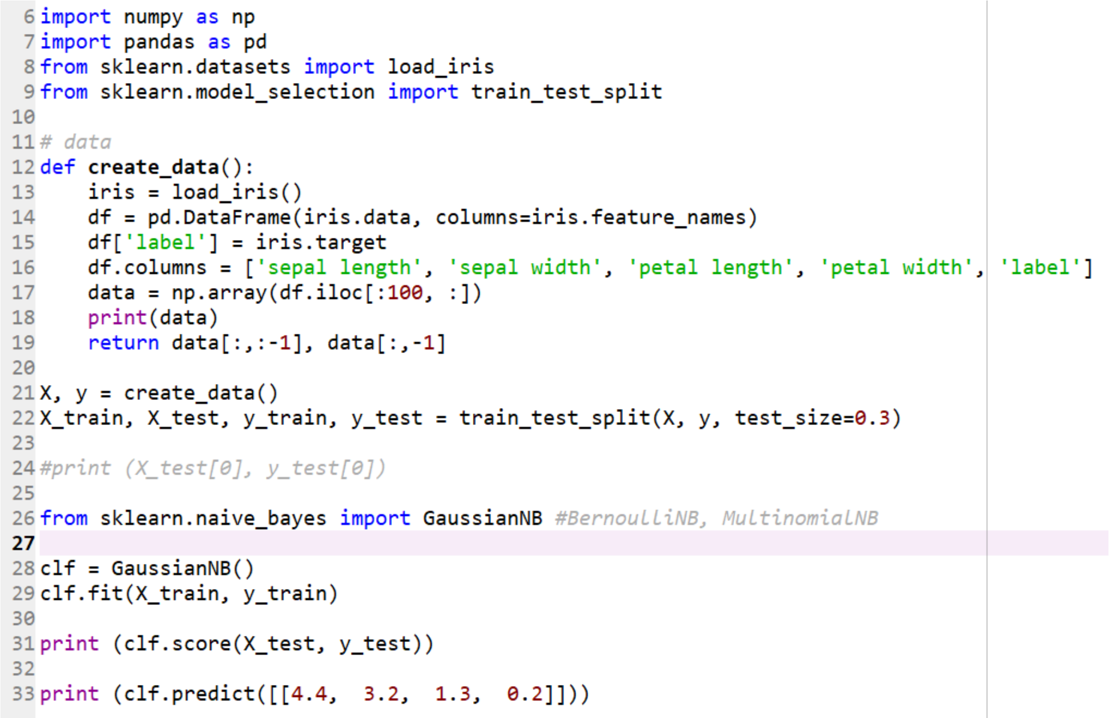
一、Fibonacci 函数



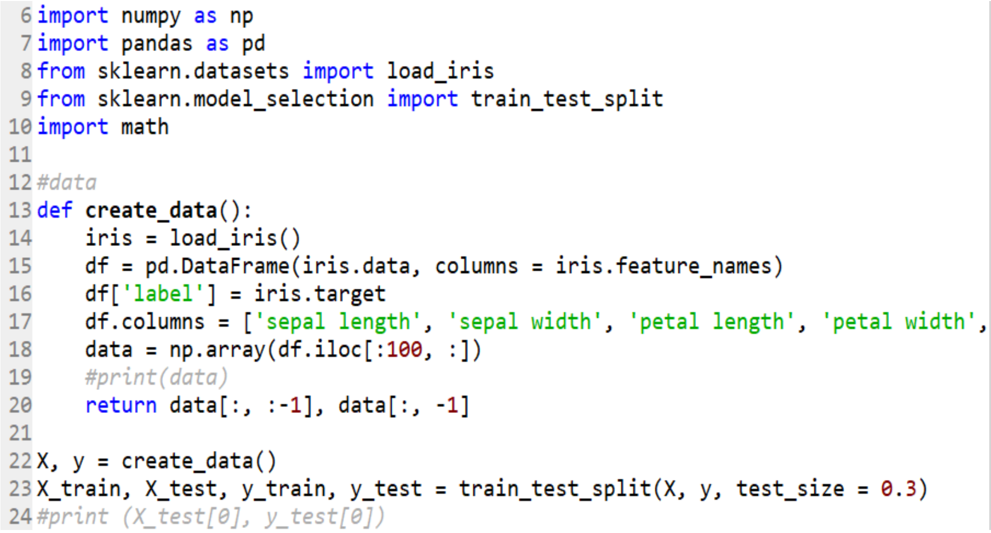
```python
n=int(input('请输入一个整数:'))

def fab(n):
    if n<1:
        print('输入有误！')
        return -1
    if n==1 or n==2:
        return 1
    else:
        return fab(n-1) + fab(n-2)

result=[]

for i in range(1, n+1):
    result.append(fab(i))

def fib(n):
    if n==0 or n==1:
        return n
    else:
        temp = fib(n-1)+fib(n-2)
        return temp

for i in range(n):
    print(fib(i+1), end=" " )
```

二、调用 sklearn 库中 naïve bayes 算法



```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# data
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, :])
    print(data)
    return data[:,:-1], data[:,-1]

X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

#print (X_test[0], y_test[0])

from sklearn.naive_bayes import GaussianNB #BernoulliNB, MultinomialNB

clf = GaussianNB()
clf.fit(X_train, y_train)

print (clf.score(X_test, y_test))

print (clf.predict([[4.4,  3.2,  1.3,  0.2]]))
```

三、编写代码实现贝叶斯算法



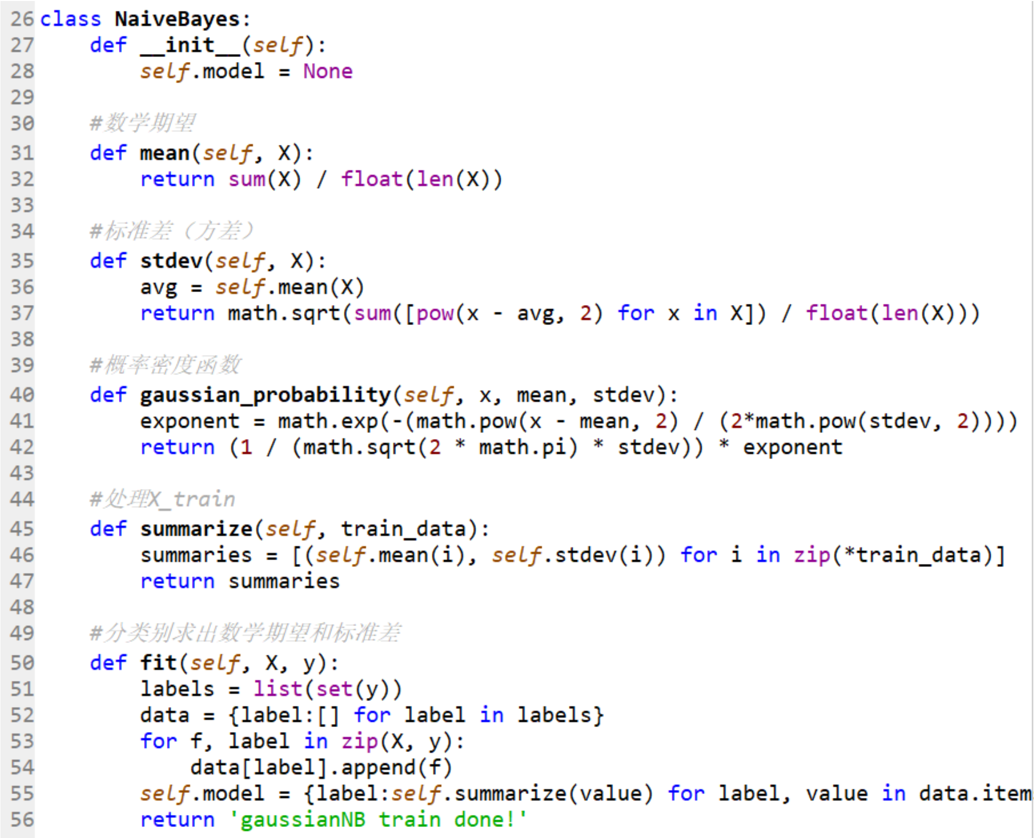
```python
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import math

#data
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns = iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, :])
    #print(data)
    return data[:, :-1], data[:, -1]

X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
#print (X_test[0], y_test[0])
```

概率密度函数：

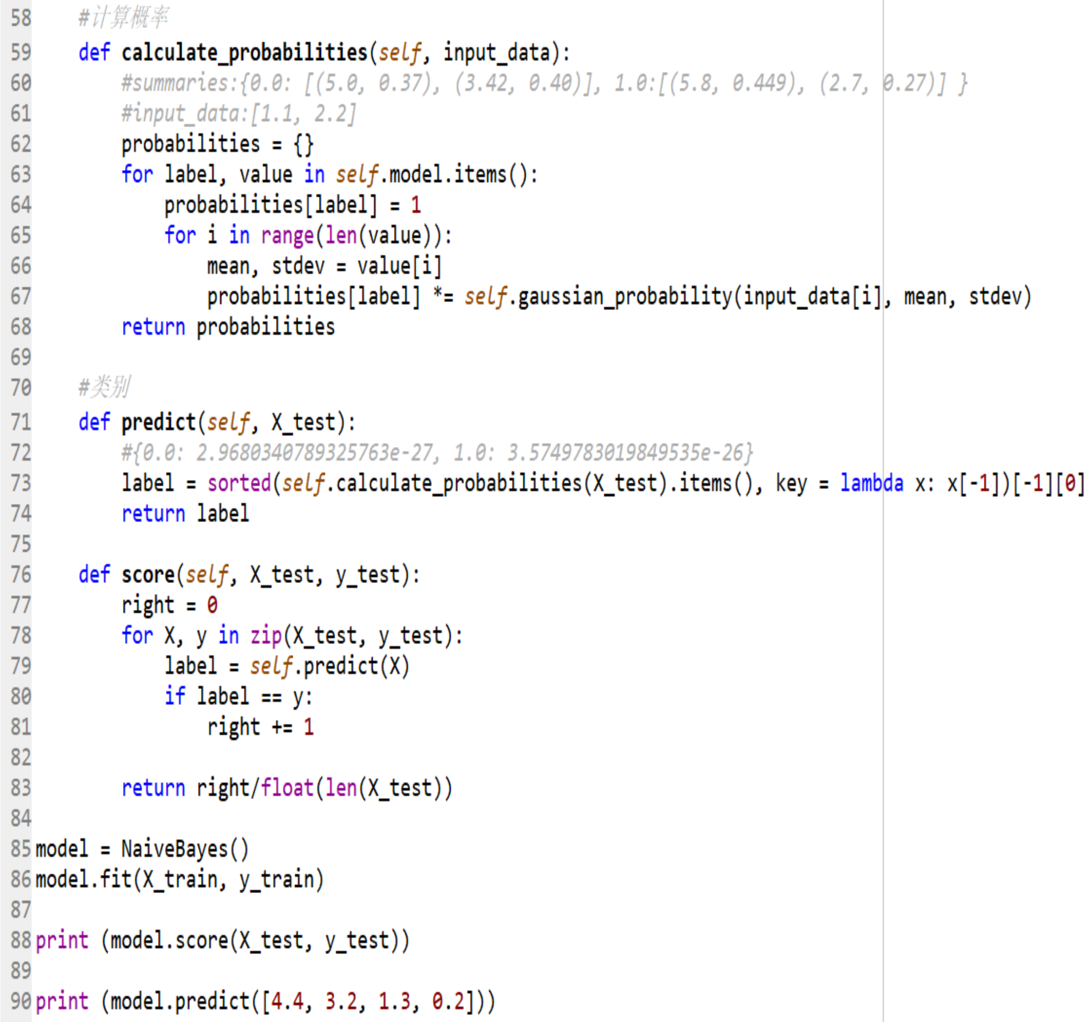
$$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma_{yk}^2}} exp(-\frac{(x_i-\mu_{yk})^2}{2\sigma_{yk}^2}) \qquad (1)$$

数学期望(mean)：

$$\mu\text{，方差：}\sigma^2 = \frac{\sum(X-\mu)^2}{N} \qquad (2)$$

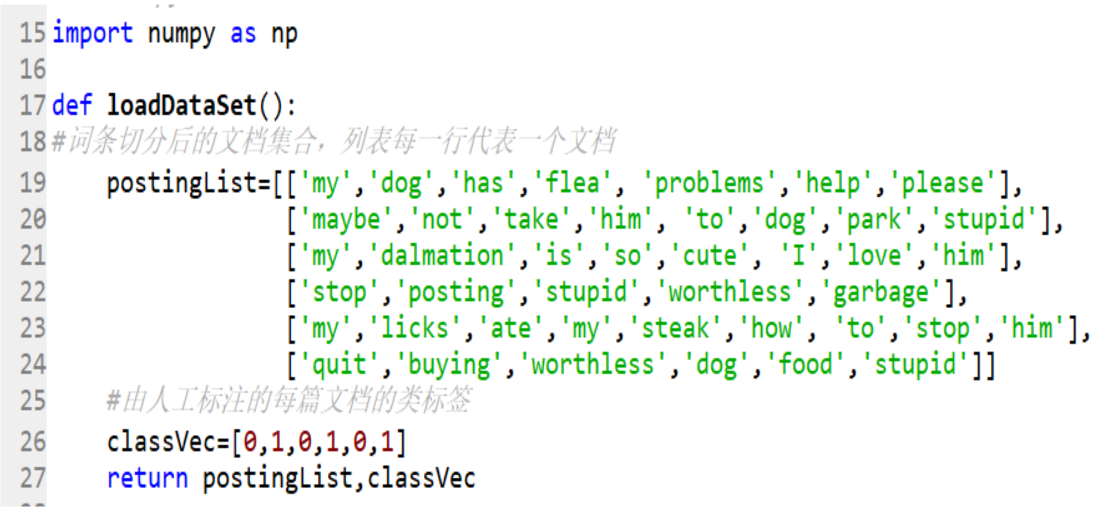


```python
class NaiveBayes:
    def __init__(self):
        self.model = None

    #数学期望
    def mean(self, X):
        return sum(X) / float(len(X))

    #标准差（方差）
    def stdev(self, X):
        avg = self.mean(X)
        return math.sqrt(sum([pow(x - avg, 2) for x in X]) / float(len(X)))

    #概率密度函数
    def gaussian_probability(self, x, mean, stdev):
        exponent = math.exp(-(math.pow(x - mean, 2) / (2*math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

    #处理X_train
    def summarize(self, train_data):
        summaries = [(self.mean(i), self.stdev(i)) for i in zip(*train_data)]
        return summaries

    #分类别求出数学期望和标准差
    def fit(self, X, y):
        labels = list(set(y))
        data = {label:[] for label in labels}
        for f, label in zip(X, y):
            data[label].append(f)
        self.model = {label:self.summarize(value) for label, value in data.items()}
        return 'gaussianNB train done!'
```

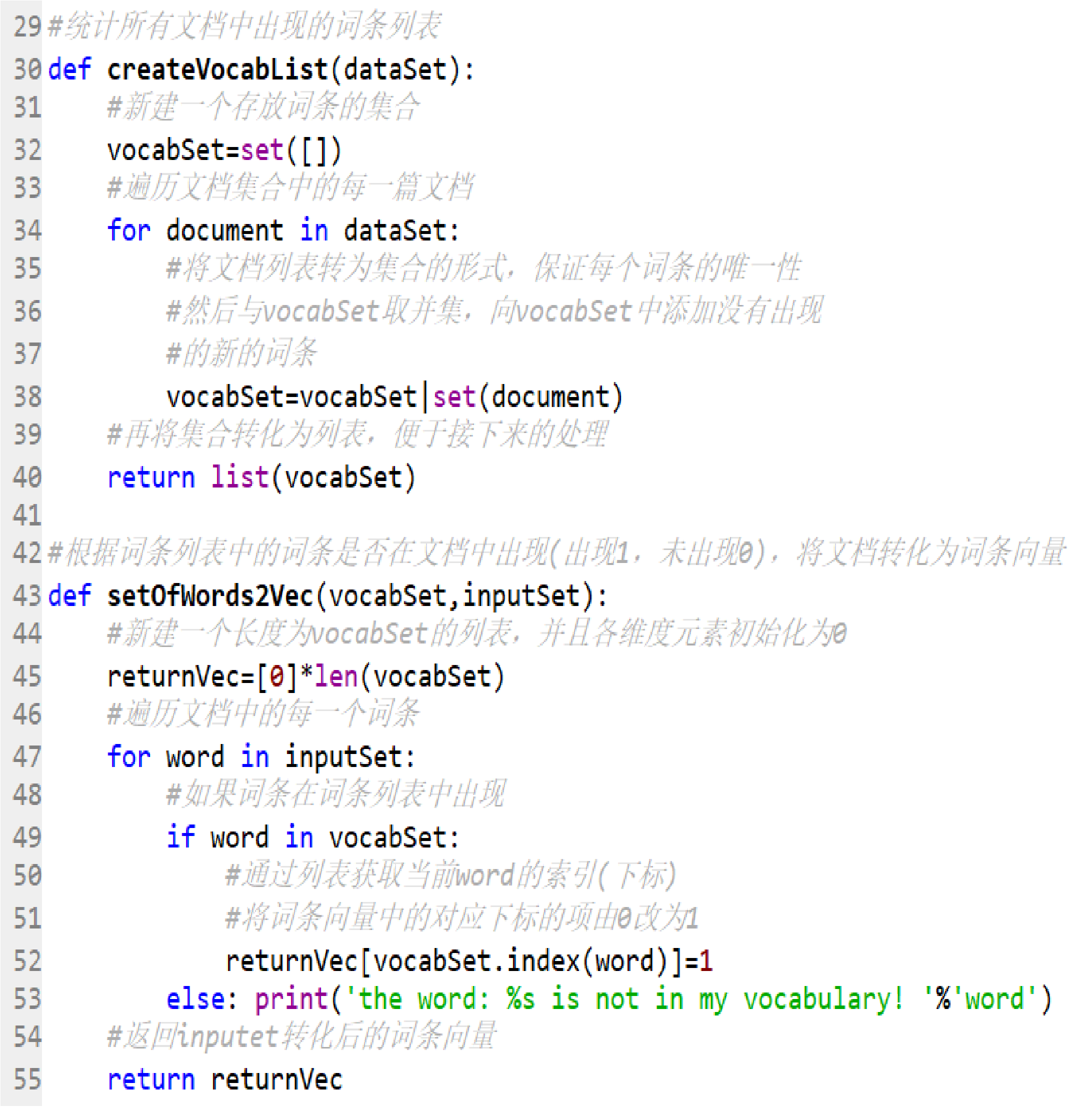


```
    #计算概率
    def calculate_probabilities(self, input_data):
        #summaries:{0.0: [(5.0, 0.37), (3.42, 0.40)], 1.0:[(5.8, 0.449), (2.7, 0.27)] }
        #input_data:[1.1, 2.2]
        probabilities = {}
        for label, value in self.model.items():
            probabilities[label] = 1
            for i in range(len(value)):
                mean, stdev = value[i]
                probabilities[label] *= self.gaussian_probability(input_data[i], mean, stdev)
        return probabilities

    #类别
    def predict(self, X_test):
        #{0.0: 2.9680340789325763e-27, 1.0: 3.5749783019849535e-26}
        label = sorted(self.calculate_probabilities(X_test).items(), key = lambda x: x[-1])[-1][0]
        return label

    def score(self, X_test, y_test):
        right = 0
        for X, y in zip(X_test, y_test):
            label = self.predict(X)
            if label == y:
                right += 1

        return right/float(len(X_test))

model = NaiveBayes()
model.fit(X_train, y_train)

print (model.score(X_test, y_test))

print (model.predict([4.4, 3.2, 1.3, 0.2]))
```

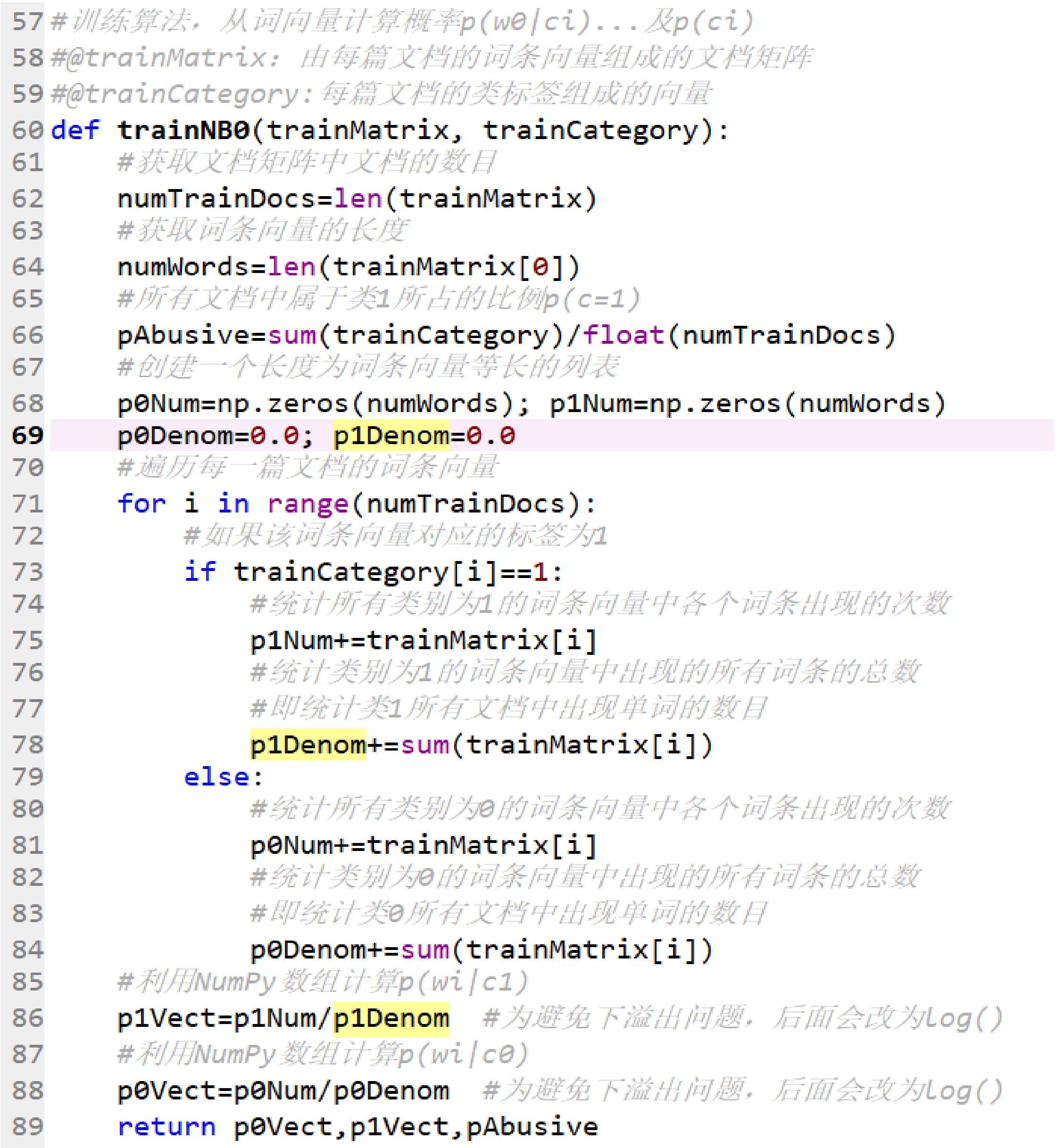
四、应用贝叶斯算法实现文本分类



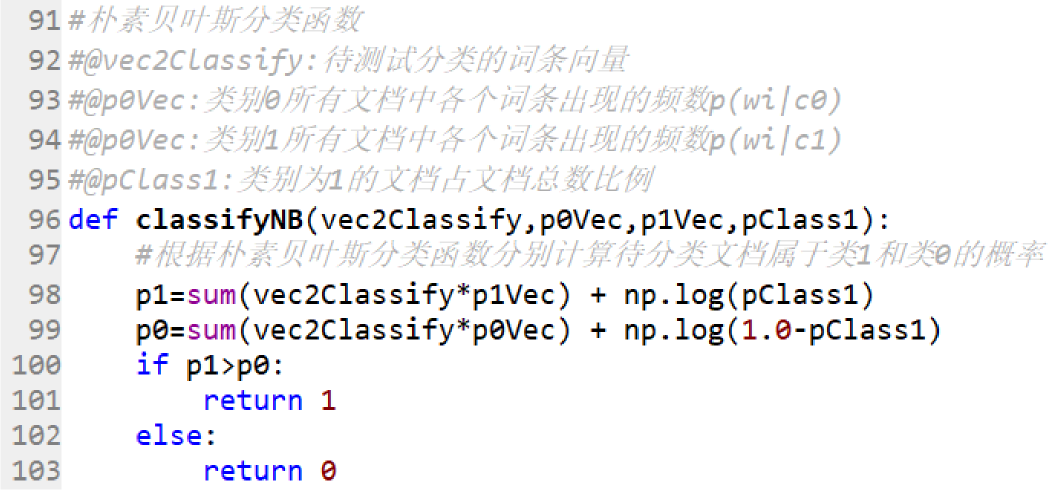
```
import numpy as np

def loadDataSet():
#词条切分后的文档集合，列表每一行代表一个文档
    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'],
                 ['my','licks','ate','my','steak','how', 'to','stop','him'],
                 ['quit','buying','worthless','dog','food','stupid']]
    #由人工标注的每篇文档的类标签
    classVec=[0,1,0,1,0,1]
    return postingList,classVec
```



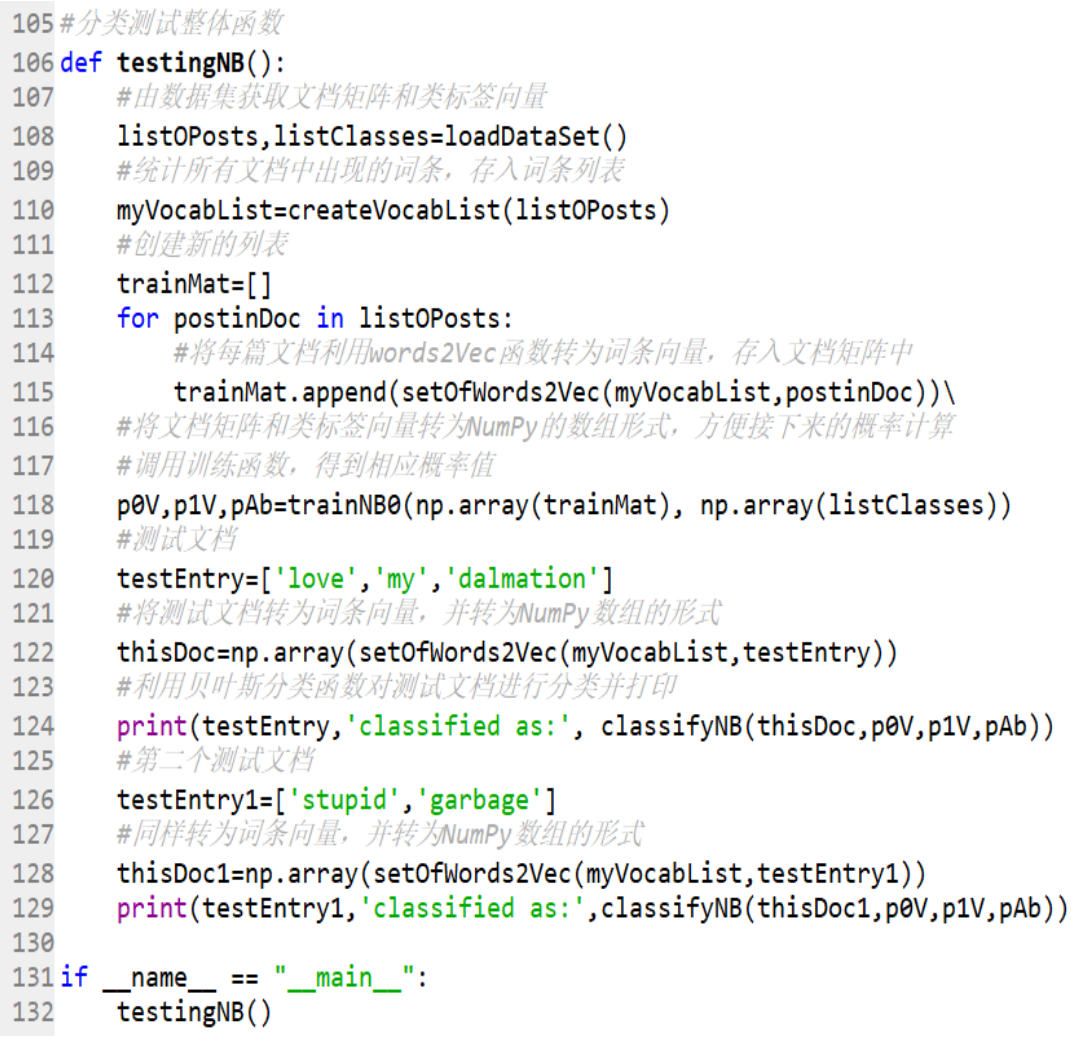
```
#统计所有文档中出现的词条列表
def createVocabList(dataSet):
    #新建一个存放词条的集合
    vocabSet=set([])
    #遍历文档集合中的每一篇文档
    for document in dataSet:
        #将文档列表转为集合的形式，保证每个词条的唯一性
        #然后与vocabSet取并集，向vocabSet中添加没有出现
        #的新的词条
        vocabSet=vocabSet|set(document)
    #再将集合转化为列表，便于接下来的处理
    return list(vocabSet)

#根据词条列表中的词条是否在文档中出现(出现1，未出现0)，将文档转化为词条向量
def setOfWords2Vec(vocabSet,inputSet):
    #新建一个长度为vocabSet的列表，并且各维度元素初始化为0
    returnVec=[0]*len(vocabSet)
    #遍历文档中的每一个词条
    for word in inputSet:
        #如果词条在词条列表中出现
        if word in vocabSet:
            #通过列表获取当前word的索引(下标)
            #将词条向量中的对应下标的项由0改为1
            returnVec[vocabSet.index(word)]=1
        else: print('the word: %s is not in my vocabulary! '%'word')
    #返回inputet转化后的词条向量
    return returnVec
```



```
#训练算法，从词向量计算概率p(w0|ci)...及p(ci)
#@trainMatrix：由每篇文档的词条向量组成的文档矩阵
#@trainCategory:每篇文档的类标签组成的向量
def trainNB0(trainMatrix, trainCategory):
    #获取文档矩阵中文档的数目
    numTrainDocs=len(trainMatrix)
    #获取词条向量的长度
    numWords=len(trainMatrix[0])
    #所有文档中属于类1所占的比例p(c=1)
    pAbusive=sum(trainCategory)/float(numTrainDocs)
    #创建一个长度为词条向量等长的列表
    p0Num=np.zeros(numWords); p1Num=np.zeros(numWords)
    p0Denom=0.0; p1Denom=0.0
    #遍历每一篇文档的词条向量
    for i in range(numTrainDocs):
        #如果该词条向量对应的标签为1
        if trainCategory[i]==1:
            #统计所有类别为1的词条向量中各个词条出现的次数
            p1Num+=trainMatrix[i]
            #统计类别为1的词条向量中出现的所有词条的总数
            #即统计类1所有文档中出现单词的数目
            p1Denom+=sum(trainMatrix[i])
        else:
            #统计所有类别为0的词条向量中各个词条出现的次数
            p0Num+=trainMatrix[i]
            #统计类别为0的词条向量中出现的所有词条的总数
            #即统计类0所有文档中出现单词的数目
            p0Denom+=sum(trainMatrix[i])
    #利用NumPy数组计算p(wi|c1)
    p1Vect=p1Num/p1Denom  #为避免下溢出问题，后面会改为log()
    #利用NumPy数组计算p(wi|c0)
    p0Vect=p0Num/p0Denom  #为避免下溢出问题，后面会改为log()
    return p0Vect,p1Vect,pAbusive
```



```
#朴素贝叶斯分类函数
#@vec2Classify:待测试分类的词条向量
#@p0Vec:类别0所有文档中各个词条出现的频数p(wi|c0)
#@p0Vec:类别1所有文档中各个词条出现的频数p(wi|c1)
#@pClass1:类别为1的文档占文档总数比例
def classifyNB(vec2Classify,p0Vec,p1Vec,pClass1):
    #根据朴素贝叶斯分类函数分别计算待分类文档属于类1和类0的概率
    p1=sum(vec2Classify*p1Vec) + np.log(pClass1)
    p0=sum(vec2Classify*p0Vec) + np.log(1.0-pClass1)
    if p1>p0:
        return 1
    else:
        return 0
```



```python
#分类测试整体函数
def testingNB():
    #由数据集获取文档矩阵和类标签向量
    listOPosts,listClasses=loadDataSet()
    #统计所有文档中出现的词条，存入词条列表
    myVocabList=createVocabList(listOPosts)
    #创建新的列表
    trainMat=[]
    for postinDoc in listOPosts:
        #将每篇文档利用words2Vec函数转为词条向量，存入文档矩阵中
        trainMat.append(setOfWords2Vec(myVocabList,postinDoc))\
    #将文档矩阵和类标签向量转为NumPy的数组形式，方便接下来的概率计算
    #调用训练函数，得到相应概率值
    p0V,p1V,pAb=trainNB0(np.array(trainMat), np.array(listClasses))
    #测试文档
    testEntry=['love','my','dalmation']
    #将测试文档转为词条向量，并转为NumPy数组的形式
    thisDoc=np.array(setOfWords2Vec(myVocabList,testEntry))
    #利用贝叶斯分类函数对测试文档进行分类并打印
    print(testEntry,'classified as:', classifyNB(thisDoc,p0V,p1V,pAb))
    #第二个测试文档
    testEntry1=['stupid','garbage']
    #同样转为词条向量，并转为NumPy数组的形式
    thisDoc1=np.array(setOfWords2Vec(myVocabList,testEntry1))
    print(testEntry1,'classified as:',classifyNB(thisDoc1,p0V,p1V,pAb))

if __name__ == "__main__":
    testingNB()
```