Title: Speech Filter and Spam Filter

Data Description: The bayes_email directory contains two subfolders: ham and spam. The .txt files in the spam folder represent spam emails.

The Naive Bayes Classifier and Its Application

The Naive Bayes classifier assumes **conditional independence of features**, meaning all attributes are considered independent given a class. Therefore, the class-conditional probability in the original Bayesian formula can be rewritten as the product of individual attribute probabilities:

$$P(x \mid c) = \prod_{i=1}^{d} P(x_i \mid c)$$

In this task, words that appear in the training samples are treated as features for the model. For such discrete features, the conditional probability of the (i^{th}) attribute belonging to class (c) is defined as:

$$P(x_i \mid c) = \frac{\text{Number of occurrences of } x_i \text{ in class } c}{\text{Total occurrences of all attributes in class } c}$$

When classifying a document, the classifier multiplies the probabilities of individual features to compute the probability of the document belonging to a specific class. However, if any of the probabilities is **zero**, the entire product becomes zero. To mitigate this issue, we initialize all word counts to **1** and the denominator to **2**. This approach is known as **Laplace smoothing**, a widely used technique to handle zero probabilities.

Additionally, multiplying many small probabilities can result in **underflow** or numerical precision errors. To address this, the **logarithm** of probabilities is used, converting products into sums, which prevents underflow and improves computational stability:

$$\log P(x \mid c) = \sum_{i=1}^{d} \log P(x_i \mid c)$$

In [11]:
```
import re
import os
import numpy as np
import random
import matplotlib.pyplot as plt

def loadDataSet():
    """
    Function: Create experimental samples
    Parameters:
        None
    Returns:
        postingList - Experimental sample split into words
        classVec - Category label vector
    """
    postingList=[['my', 'dog', 'has', 'flea', 'problems', 'help', 'please'],
                 ['maybe', 'not', 'take', 'him', 'to', 'dog', 'park', 'stupi
                 ['my', 'dalmation', 'is', 'so', 'cute', 'I', 'love', 'him']
                 ['stop', 'posting', 'stupid', 'worthless', 'garbage'],
                 ['mr', 'licks', 'ate', 'my', 'steak', 'how', 'to', 'stop',
                 ['quit', 'buying', 'worthless', 'dog', 'food', 'stupid']]
    classVec = [0,1,0,1,0,1]
    return postingList,classVec

def createVocabList(dataSet):
    """
    Function: Organize the split experimental sample words into a non-repeti
    Parameters:
        dataSet - Organized sample dataset
    Returns:
        vocabSet - Returns a non-repetitive word list, i.e., vocabulary list
    """
    vocabSet = []
    for sentence in dataSet:
        for word in sentence:
            if word not in vocabSet:
                vocabSet.append(word)

    return vocabSet

def setOfWords2Vec(vocabList, inputSet):
    """
    Function: Vectorize the inputSet according to the vocabList vocabulary l
    Parameters:
        vocabList - List returned by createVocabList
        inputSet - Split word list
    Returns:
        returnVec - Document vector, word set model
    """
    returnVec = np.zeros(len(vocabList))
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1

    return returnVec.astype(int).tolist()

def trainNB(trainMatrix,trainCategory):
    """
    Function: Naive Bayes classifier training function
    Parameters:
        trainMatrix - Training document matrix, i.e., the matrix composed of
        trainCategory - Training category label vector, i.e., classVec retur
    Returns:
        p0Vect - Conditional probability array of non-abusive class
        p1Vect - Conditional probability array of abusive class
        pAbusive - Probability that the document belongs to the abusive clas
    """
    numtrain = len(trainMatrix)
    numwords = len(trainMatrix[0])
    pAbusive = sum(trainCategory)/float(numtrain)
    p0Vect = np.zeros(numwords) + 1
    p1Vect = np.zeros(numwords) + 1
    p0Denom = 2
    p1Denom = 2
    for i in range(numtrain):
        if trainCategory[i] == 1:
            p1Vect += trainMatrix[i]
            p1Denom += sum(trainMatrix[i])
        else:
            p0Vect += trainMatrix[i]
            p0Denom += sum(trainMatrix[i])
    p1Vect = p1Vect/p1Denom
    p0Vect = p0Vect/p0Denom

    return p0Vect, p1Vect, pAbusive

def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    """
    Function: Naive Bayes classifier classification function
    Parameters:
        vec2Classify - Word array to be classified
        p0Vec - Conditional probability array of non-abusive class
        p1Vec - Conditional probability array of abusive class
        pClass1 - Probability that the document belongs to the abusive class
    Returns:
        0 - Belongs to non-abusive class
        1 - Belongs to abusive class
    """
    p1 = 1; p0 = 1;
    for i in range(len(vec2Classify)):
        if vec2Classify[i] != 0:
            p1 *= p1Vec[i]
            p0 *= p0Vec[i]
    p1 = np.log(p1*pClass1)
    p0 = np.log(p0*(1.0-pClass1))
    if p1 > p0:
        return 1
    else:
        return 0

def testingNB():
    """
    Function: Test the Naive Bayes classifier
    Test sample 1: ['love', 'my', 'dalmation']
    Test sample 2: ['stupid', 'garbage']
    Parameters:
        None
    Returns:
        None
    """
    postingList, classVec = loadDataSet()
    vocabSet = createVocabList(postingList)
    vec_train = []
    for sentence in postingList:
        vec_train.append(setOfWords2Vec(vocabSet, sentence))
    p0Vect, p1Vect, pAbusive = trainNB(vec_train, classVec)

    test = [['love', 'my', 'dalmation'], ['stupid', 'garbage']]
    vec_test = []
    for sentence in test:
        vec_test.append(setOfWords2Vec(vocabSet, sentence))
    for i in range(len(vec_test)):
        result = classifyNB(vec_test[i], p0Vect, p1Vect, pAbusive)
        if result == 1:
            print('{} belongs to abusive class'.format(test[i]))
        else:
            print('{} belongs to non-abusive class'.format(test[i]))

def textParse(bigString):
    """
    Function: Receive a string and parse it into a list of words
    Parameters:
        bigString - String
    Returns:
        List of words (except for single letters, such as uppercase I, other
    """
    bigString = re.sub(r'[\W_]+', ' ', bigString)
    raw = bigString.split()
    words = []
    for word in raw:
        if len(word) >= 3:
            words.append(word.lower())

    return words

def spamTest():
    """
    Function: Divide the dataset into training and test sets, and use cross-
    """
    # Get text data and labels
    filenames = []
    path_pos = 'datasets//bayes_email//ham'
    path_neg = 'datasets//bayes_email//spam'
    for files in os.listdir(path_pos):
        if files.endswith('txt'):
            file = os.path.join(path_pos, files)
            filenames.append(file)
    for files in os.listdir(path_neg):
        if files.endswith('txt'):
            file = os.path.join(path_neg, files)
            filenames.append(file)
    data = []
    classlist = []
    for filename in filenames:
        with open(filename, encoding='cp1252') as file:
            data.append(textParse(file.read()))
        if 'ham' in filename:
            classlist.append(0)
        else:
            classlist.append(1)
    # Get vocabulary list and convert to vector
    vocablist = createVocabList(data)
    datamat = []
    for da in data:
        datamat.append(setOfWords2Vec(vocablist, da))

    # Divide training and test sets
    index = random.sample(range(50), 50)
    trainset = []; trainclass = [];
    testset = []; testclass = [];
    for i in range(0,40):
        trainset.append(datamat[index[i]])
        trainclass.append(classlist[index[i]])
    for i in range(40, 50):
        testset.append(datamat[index[i]])
        testclass.append(classlist[index[i]])

    # Start training and testing
    p0V, p1V, pSpam = trainNB(trainset, trainclass)
    errorcount = 0
    for i in range(len(testset)):
        result = classifyNB(testset[i], p0V, p1V, pSpam)
        if result != testclass[i]:
            errorcount += 1
            print('{} was misclassified, the classification result is {}, th

    print('Error rate: {}'.format(errorcount/len(testset)))
```

In [12]:
```
testingNB()
spamTest()
```

```
['love', 'my', 'dalmation'] belongs to non-abusive class
['stupid', 'garbage'] belongs to abusive class
['yeah', 'ready', 'may', 'not', 'here', 'because', 'jar', 'jar', 'has', 'pla
ne', 'tickets', 'germany', 'for'] was misclassified, the classification resu
lt is 1, the true result is 0
Error rate: 0.1
```