```
# -*- coding: UTF-8 -*-
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', '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]
    return postingList,classVec

def createVocabList(dataSet):
    """
    Function: Organize the split experimental sample words into a non-repetitive word list, i.e., vocabulary list
    Parameters:
        dataSet - Organized sample dataset
    Returns:
        vocabSet - Returns a non-repetitive word
```

```
32 list, i.e., vocabulary list
33     """
34     vocabSet = []
35     for sentence in dataSet:
36         for word in sentence:
37             if word not in vocabSet:
38                 vocabSet.append(word)
39 
40     return vocabSet
41 
42 def setOfWords2Vec(vocabList, inputSet):
43     """
44     Function: Vectorize the inputSet according to
   the vocabList vocabulary list, each element of the
   vector is 1 or 0
45     Parameters:
46         vocabList - List returned by
   createVocabList
47         inputSet - Split word list
48     Returns:
49         returnVec - Document vector, word set model
50     """
51     returnVec = np.zeros(len(vocabList))
52     for word in inputSet:
53         if word in vocabList:
54             returnVec[vocabList.index(word)] += 1
55 
56     return returnVec.astype(int).tolist()
57 
58 def trainNB(trainMatrix,trainCategory):
59     """
60     Function: Naive Bayes classifier training
   function
61     Parameters:
62         trainMatrix - Training document matrix, i.e
   ., the matrix composed of returnVec returned by
   setOfWords2Vec
63         trainCategory - Training category label
   vector, i.e., classVec returned by loadDataSet
64     Returns:
65         p0Vect - Conditional probability array of
```

```
65 non-abusive class
66         p1Vect - Conditional probability array of
   abusive class
67         pAbusive - Probability that the document
   belongs to the abusive class
68     """
69     numtrain = len(trainMatrix)
70     numwords = len(trainMatrix[0])
71     pAbusive = sum(trainCategory)/float(numtrain)
72     p0Vect = np.zeros(numwords) + 1
73     p1Vect = np.zeros(numwords) + 1
74     p0Denom = 2
75     p1Denom = 2
76     for i in range(numtrain):
77         if trainCategory[i] == 1:
78             p1Vect += trainMatrix[i]
79             p1Denom += sum(trainMatrix[i])
80         else:
81             p0Vect += trainMatrix[i]
82             p0Denom += sum(trainMatrix[i])
83     p1Vect = p1Vect/p1Denom
84     p0Vect = p0Vect/p0Denom
85
86     return p0Vect, p1Vect, pAbusive
87
88 def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1
   ):
89     """
90     Function: Naive Bayes classifier
   classification function
91     Parameters:
92         vec2Classify - Word array to be classified
93         p0Vec - Conditional probability array of
   non-abusive class
94         p1Vec - Conditional probability array of
   abusive class
95         pClass1 - Probability that the document
   belongs to the abusive class
96     Returns:
97         0 - Belongs to non-abusive class
98         1 - Belongs to abusive class
```

```
 99     """
100     p1 = 1; p0 = 1;
101     for i in range(len(vec2Classify)):
102         if vec2Classify[i] != 0:
103             p1 *= p1Vec[i]
104             p0 *= p0Vec[i]
105     p1 = np.log(p1*pClass1)
106     p0 = np.log(p0*(1.0-pClass1))
107     if p1 > p0:
108         return 1
109     else:
110         return 0
111 
112 def testingNB():
113     """
114     Function: Test the Naive Bayes classifier
115     Test sample 1: ['love', 'my', 'dalmation']
116     Test sample 2: ['stupid', 'garbage']
117     Parameters:
118         None
119     Returns:
120         None
121     """
122     postingList, classVec = loadDataSet()
123     vocabSet = createVocabList(postingList)
124     vec_train = []
125     for sentence in postingList:
126         vec_train.append(setOfWords2Vec(vocabSet,
    sentence))
127     p0Vect, p1Vect, pAbusive = trainNB(vec_train,
    classVec)
128 
129     test = [['love', 'my', 'dalmation'], ['stupid'
    , 'garbage']]
130     vec_test = []
131     for sentence in test:
132         vec_test.append(setOfWords2Vec(vocabSet,
    sentence))
133     for i in range(len(vec_test)):
134         result = classifyNB(vec_test[i], p0Vect,
    p1Vect, pAbusive)
```

```
135         if result == 1:
136             print('{} belongs to abusive class'.
    format(test[i]))
137         else:
138             print('{} belongs to non-abusive class
    '.format(test[i]))
139 
140 def textParse(bigString):
141     """
142     Function: Receive a string and parse it into a
     list of words
143     Parameters:
144         bigString - String
145     Returns:
146         List of words (except for single letters,
    such as uppercase I, other words are converted to
    lowercase, and strings with a length of less than
    3 are filtered)
147     """
148     bigString = re.sub(r'[\W_]+', ' ', bigString)
149     raw = bigString.split()
150     words = []
151     for word in raw:
152         if len(word) >= 3:
153             words.append(word.lower())
154 
155     return words
156 
157 def spamTest():
158     """
159     Function: Divide the dataset into training and
     test sets, and use cross-validation to test the
    accuracy of the Naive Bayes classifier
160     """
161     # Get text data and labels
162     filenames = []
163     path_pos = 'bayes_email//ham'
164     path_neg = 'bayes_email//spam'
165     for files in os.listdir(path_pos):
166         if files.endswith('txt'):
167             file = os.path.join(path_pos, files)
```

```
168             filenames.append(file)
169     for files in os.listdir(path_neg):
170         if files.endswith('txt'):
171             file = os.path.join(path_neg, files)
172             filenames.append(file)
173     data = []
174     classlist = []
175     for filename in filenames:
176         with open(filename, encoding='cp1252') as
    file:
177             data.append(textParse(file.read()))
178         if 'ham' in filename:
179             classlist.append(0)
180         else:
181             classlist.append(1)
182 
183     # Get vocabulary list and convert to vector
184     vocablist = createVocabList(data)
185     datamat = []
186     for da in data:
187         datamat.append(setOfWords2Vec(vocablist,
    da))
188 
189     # Divide training and test sets
190     index = random.sample(range(50), 50)
191     trainset = []; trainclass = [];
192     testset = []; testclass = [];
193     for i in range(0,40):
194         trainset.append(datamat[index[i]])
195         trainclass.append(classlist[index[i]])
196     for i in range(40, 50):
197         testset.append(datamat[index[i]])
198         testclass.append(classlist[index[i]])
199 
200     # Start training and testing
201     p0V, p1V, pSpam = trainNB(trainset, trainclass
    )
202     errorcount = 0
203     for i in range(len(testset)):
204         result = classifyNB(testset[i], p0V, p1V,
    pSpam)
```

```
205         if result != testclass[i]:
206             errorcount += 1
207             print('{} was misclassified, the
    classification result is {}, the true result is
     {}'.format(data[index[40+i]], result, testclass[i
    ]))
208 
209     print('Error rate: {}'.format(errorcount/len(
    testset)))
210 
211 if __name__ == '__main__':
212     testingNB()
213     spamTest()
```