Naive Bayes(배포용)

2019년 6월 1일 토요일 오후 4:41

Probability

$$P(A) = \frac{\text{The number of ways event can occur}}{\text{The total number of Possible Outcomes}}$$

$$P(\text{apple}) = \frac{2}{3}$$

$$P(\text{banana}) = \frac{1}{3}$$

Conditional Probability

the probability of an event (*A*), given that another (*B*) has already occurred

조건부 확률로써 어떠한 상황이 주어졌을 때 그 상황속에서 다른 상황이 일어날 확률을 의미한다.

조건부 확률의 경우에는 2가지 경우가 있다.

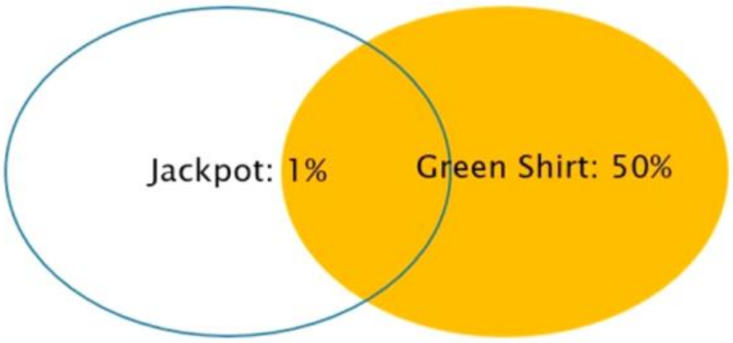
첫번째 경우는 서로 영향을 끼치지 않을때이고 두번째 경우는 영향을 끼칠때이다.
이번 예제는 서로 영향을 끼치지 않을때이다.

calculating the intersection (when two events are independent)

$$P(A \cap B) = P(A) * P(B)$$

$P(A|B) = P(A)$ B has no effect on A

$P(B|A) = P(B)$ A has no effect on B



If there is 1% chance that you get jackpot from casino,
and a 50% chance that your t-shirts is green color,
then what is the probability that
You wear green t-shirts and you get jackpot from casino(assuming that t-shirts has no influence on jackpot)?

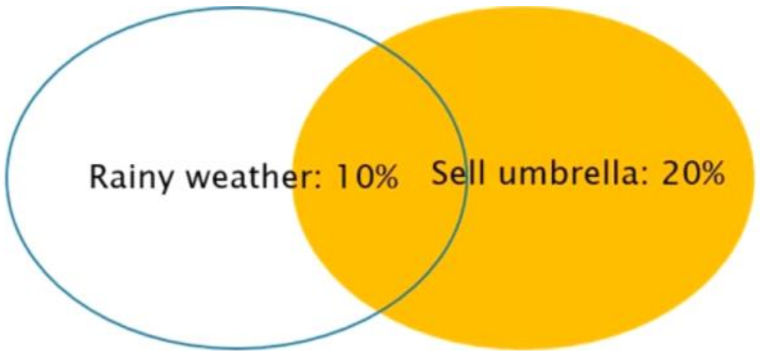
Since t-shirts has no effect on casino result,
we tack these events as independent, and so the probability that both events will occur is

$$P(\text{Jackpot} \cap \text{Green\_shirt}) = P(\text{Jackpot}) * P(\text{Green\_shirt}) = (0.01)*(0.5) = 0.005$$

calculating the intersection (when two events are dependent)

$$P(A \cap B) = \quad P(A) * P(B \mid A)$$

$$P(B \mid A) = P(A \cap B) / P(B)$$

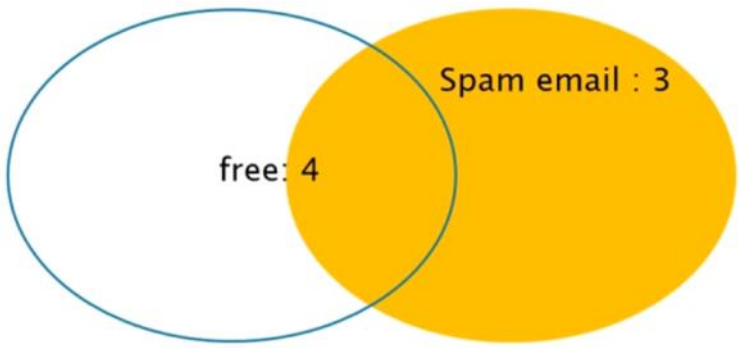


Bayes' Theorem

You don't have a girl friend.
I know it because you watch youtube all weekend, and fat, no sense of fashion!
I can predict you have no girl friend, because I know probability of fat, youtube, no fashion, and these conditional probability given that when they don't have girl friend.
You have 95% chance of no girl friend!

$$P(\text{no girl friend} \mid \text{fat, youtube, no fashion}) = \frac{P(\text{fat} \mid \text{no girl friend}) * P(\text{youtube} \mid \text{no girl friend}) * P(\text{no fashion} \mid \text{no girl frield}) * P(\text{no girl friend})}{P(\text{fat}) * P(\text{youtube}) * P(\text{no fashion})}$$

Bayes' Theorem proof



"spam" and "free" are clearly dependent,

Since we assume when we have spam email, there might be "free" in the text, and also when we have "free" in the email, we suspect it is "spam"

$$P(A|B) = P(A \cap B) \ / \ P(B)$$

$$P(A \cap B) = P(A|B) \cdot P(B)$$

$$P(B \cap A) = P(B|A) \cdot P(A)$$

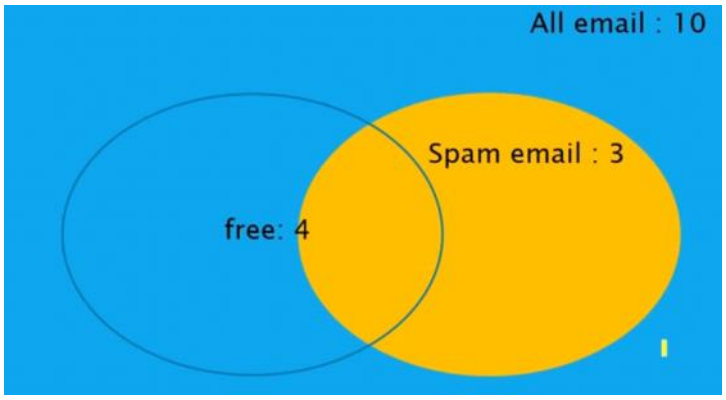
$$P(B \cap A) = P(A \cap B)$$

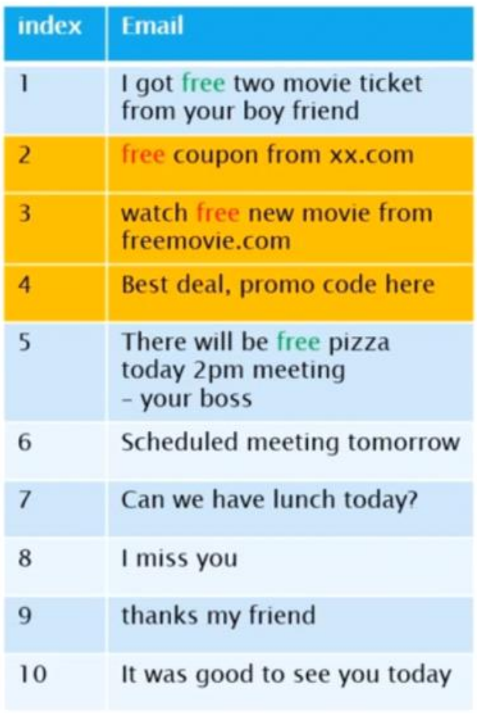
$$P(A \cap B) = P(B|A) \cdot P(A) = P(A|B) \cdot P(B)$$

$$P(B|A) \cdot P(A) = P(A|B) \cdot P(B)$$

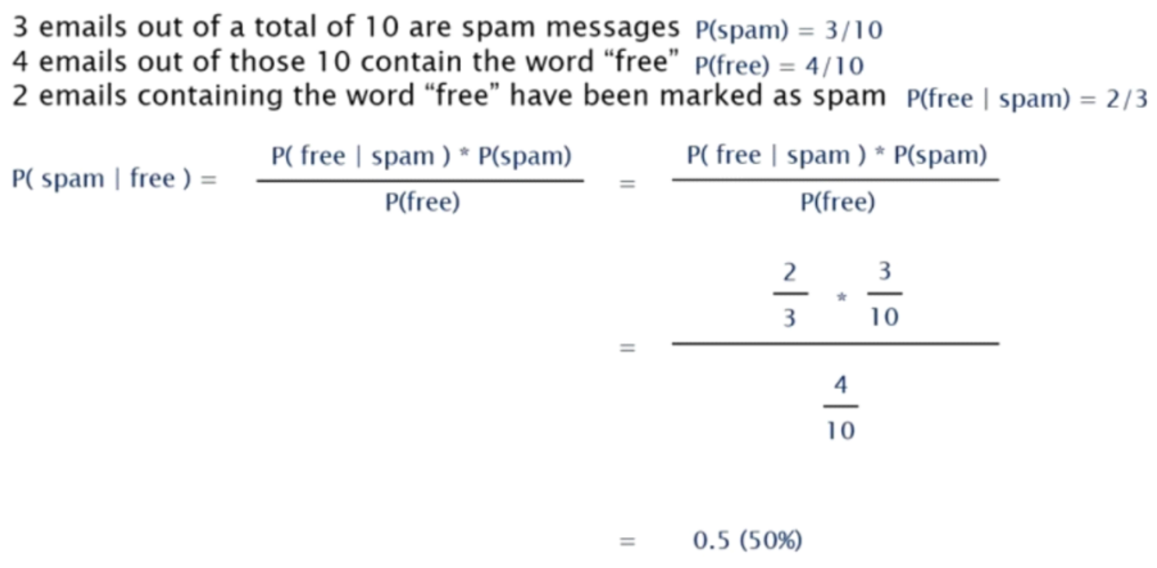
$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

Bayes' Theorem





index	Email
1	I got free two movie ticket from your boy friend
2	free coupon from xx.com
3	watch free new movie from freemovie.com
4	Best deal, promo code here
5	There will be free pizza today 2pm meeting - your boss
6	Scheduled meeting tomorrow
7	Can we have lunch today?
8	I miss you
9	thanks my friend
10	It was good to see you today

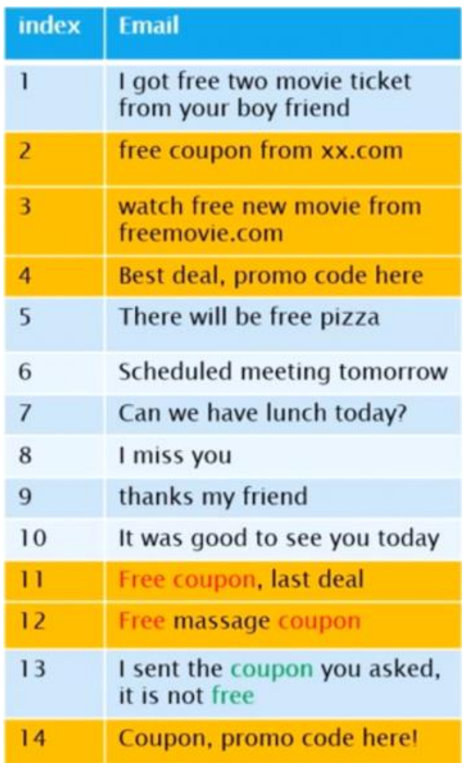

3 emails out of a total of 10 are spam messages $P(spam) = 3/10$
4 emails out of those 10 contain the word "free" $P(free) = 4/10$
2 emails containing the word "free" have been marked as spam $P(free \mid spam) = 2/3$

$$P(\, spam \mid free \,) = \frac{P(\, free \mid spam \,) * P(spam)}{P(free)} = \frac{P(\, free \mid spam \,) * P(spam)}{P(free)} = \frac{\frac{2}{3} * \frac{3}{10}}{\frac{4}{10}} = 0.5\ (50\%)$$

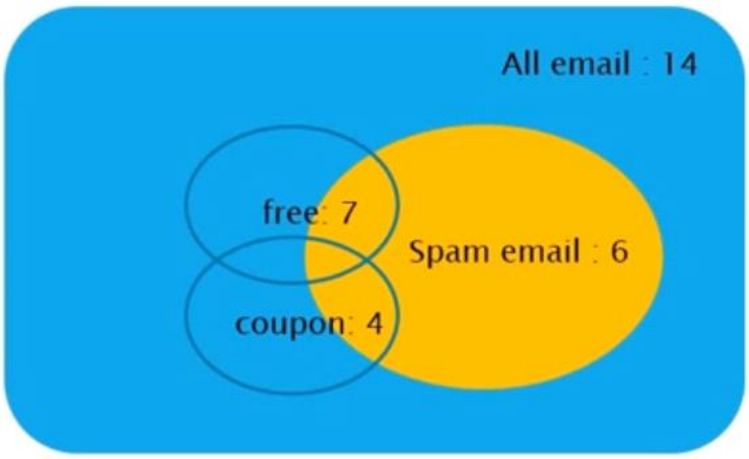
So eash, I think I don't need machine's help for spam filtering...

Well, the real world, is much more complicated
We filter spam with complicated combination of words (free, coupon, $, fuXX, sexy...)
Don't you need machine's help for all the combination?

Bayes' Theorem
(spam classifier, contains 'free', 'coupon')



index	Email
1	I got free two movie ticket from your boy friend
2	free coupon from xx.com
3	watch free new movie from freemovie.com
4	Best deal, promo code here
5	There will be free pizza
6	Scheduled meeting tomorrow
7	Can we have lunch today?
8	I miss you
9	thanks my friend
10	It was good to see you today
11	Free coupon, last deal
12	Free massage coupon
13	I sent the coupon you asked, it is not free
14	Coupon, promo code here!



6 emails out of a total of 14 are spam messages $P(spam) = 6/14$
7 emails out of those 14 contain the word "free" $P(free) = 4/14$
4 emails containing the word "free" have been marked as spam $P(free \mid spam) = 2/6$
4 emails out of a those 14 contain the word "coupon" $P(coupon) = 4/14$
3 emails containing the word "coupon" have been marked as spam $P(coupon \mid spam) = 3/6$

To make this tutorial as easy as possible,
I will assume 'free' and 'coupon' are independent

$$P(\text{ spam} \mid \text{free, coupon }) = \frac{P(\text{ free} \mid \text{spam} \cap \text{coupon }) * P\,(\text{coupon} \mid \text{spam}) * P(\text{spam})}{P(\text{free}|\text{coupon}) * P(\text{coupon})}$$

$$= \frac{\frac{3}{4} * \frac{4}{6} * \frac{6}{14}}{\frac{4}{5} * \frac{5}{14}}$$

$$= \quad 0.75\ (75\%)$$

So What? Shouldn't we talk about machine learning?

If P(spam|free, coupon) > P(not a spam | free, coupon),
the email is spam.

I will filter emails has "free" and "coupon" because it has
higher probability of spam.

Practice

$$P(\text{ spam } | \text{ word }) = \frac{P(\text{ word } | \text{ spam }) * P(\text{spam})}{P(\text{word})}$$

$$P(\text{ spam } | \text{ free, coupon}) = \frac{P(\text{ free, coupon} | \text{ spam }) * P(\text{spam})}{P(\text{free, coupon})}$$

$$P(\text{ spam } | \text{ w0,w1,w2..wn }) = \frac{P(\text{ w0,w1,w2... wn } | \text{ spam }) * P(\text{spam})}{P(\text{w0,w1,w2...wn})}$$

$$P(\text{ spam } | \text{ w0,w1,w2..wn }) = \frac{P(\text{ w0}|\text{spam}) * \dots P(\text{wn}|\text{spam}) * P(\text{spam})}{P(\text{w0}) * P(\text{w1})*P(\text{w2})* \dots * P(\text{wn})}$$

Source is from https://www.manning.com/books/machine-learning-in-action , author : Peter Harrington

```
'''
Created on Oct 19, 2010
@author: Peter
'''
from numpy import *

def loadDataSet():
    postingList=[['I', 'got', 'free', 'two', 'movie', 'ticket', 'from', 'your', 'boy', 'friend'],
                 ['free', 'coupon', 'from', 'xx.com'],
                 ['watch', 'free', 'new', 'movie', 'from', 'freemovie.com'],
                 ['best', 'deal', 'promo', 'code', 'here'],
                 ['there', 'will', 'be', 'free', 'pizza', 'during', 'the', 'meeting'],
                 ['scheduled', 'meeting', 'tomorrow'],
                 ['can','we','have','lunch','today'],
                 ['I','miss','you'],
                 ['thanks','my','friend'],
                 ['it','was','good','to','see','you','today'],
                 ['free','coupon','last','deal'],
                 ['free','massage','coupon'],
                 ['I','sent','the','coupon','you','asked','it','is','not','free'],
                 ['coupon','promo','code','here']]
    classVec = [0,1,1,1,0,0,0,0,0,0,1,1,0,1]    #1 is spam, 0 not
    return postingList,classVec

def createVocabList(dataSet):
    vocabSet = set([])  #create empty set
    for document in dataSet:
        vocabSet = vocabSet | set(document) #union of the two sets
    return list(vocabSet)

def setOfWords2Vec(vocabList, inputSet):
    # len(vocabList) : 49
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] = 1
        else: print("the word: %s is not in my Vocabulary!" % word)
    return returnVec
returnVec

def trainNB00(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix)
    numWords = len(trainMatrix[0])
    pSpam = sum(trainCategory)/float(numTrainDocs)
    p0Num = zeros(numWords); p1Num = 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,pSpam

def trainNB0(trainMatrix,trainCategory):
    numTrainDocs = len(trainMatrix) # 14
    numWords = len(trainMatrix[0])  # 49
    pSpam = sum(trainCategory)/float(numTrainDocs)
    p0Num = ones(numWords); p1Num = ones(numWords)
    p0Denom = 2.0; p1Denom = 2.0
    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 = log(p1Num/p1Denom)
    p0Vect = log(p0Num/p0Denom)
    return p0Vect,p1Vect,pSpam

def classifyNB(vec2Classify, p0Vec, p1Vec, pClass1):
    p1 = sum(vec2Classify * p1Vec) + log(pClass1)    #element-wise mult
    p0 = sum(vec2Classify * p0Vec) + log(1.0 - pClass1)
    if p1 > p0:
        return 1
    else:
        return 0

def bagOfWords2VecMN(vocabList, inputSet):
    returnVec = [0]*len(vocabList)
    for word in inputSet:
        if word in vocabList:
            returnVec[vocabList.index(word)] += 1
    return returnVec
```

```
def testingNB():
    listOPosts,listClasses = loadDataSet()
    myVocabList = createVocabList(listOPosts)
    trainMat=[]

    for postinDoc in listOPosts:
        trainMat.append(setOfWords2Vec(myVocabList, postinDoc))

    p0V,p1V,pSpam = trainNB0(array(trainMat),array(listClasses))

    testEntry = ['free', 'pizza', 'coupon']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))

    if classifyNB(thisDoc, p0V, p1V, pSpam) == 1:
        print(testEntry, 'classified this is a spam')
    else:
        print(testEntry, 'classified this is not a spam')

    testEntry = ['I', 'will', 'miss','free', 'pizza']
    thisDoc = array(setOfWords2Vec(myVocabList, testEntry))

    if classifyNB(thisDoc,p0V,p1V,pSpam) == 1:
        print(testEntry, 'classified this is a spam')
    else:
        print(testEntry, 'classified this is not a spam')

def textParse(bigString):    #input is big string, #output is word list
    import re
    listOfTokens = re.split(r'₩W*', bigString)
    return [tok.lower() for tok in listOfTokens if len(tok) > 2]

def spamTest():
    docList=[]; classList = []; fullText =[]
    for i in range(1,26):
        wordList = textParse(open('email/spam/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1)
        wordList = textParse(open('email/ham/%d.txt' % i).read())
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)#create vocabulary
    trainingSet = range(50); testSet=[]           #create test set
    for i in range(10):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet:#train the classifier (get probs) trainNB0
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    for docIndex in testSet:        #classify the remaining items
```

```
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
            print("classification error",docList[docIndex])
    print('the error rate is: ',float(errorCount)/len(testSet))
    #return vocabList,fullText

def calcMostFreq(vocabList,fullText):
    import operator
    freqDict = {}
    for token in vocabList:
        freqDict[token]=fullText.count(token)
    sortedFreq = sorted(freqDict.iteritems(), key=operator.itemgetter(1), reverse=True)
    return sortedFreq[:30]

def localWords(feed1,feed0):
    import feedparser
    docList=[]; classList = []; fullText =[]
    minLen = min(len(feed1['entries']),len(feed0['entries']))
    for i in range(minLen):
        wordList = textParse(feed1['entries'][i]['summary'])
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(1) #NY is class 1
        wordList = textParse(feed0['entries'][i]['summary'])
        docList.append(wordList)
        fullText.extend(wordList)
        classList.append(0)
    vocabList = createVocabList(docList)#create vocabulary
    top30Words = calcMostFreq(vocabList,fullText)   #remove top 30 words
    for pairW in top30Words:
        if pairW[0] in vocabList: vocabList.remove(pairW[0])
    trainingSet = range(2*minLen); testSet=[]           #create test set
    for i in range(20):
        randIndex = int(random.uniform(0,len(trainingSet)))
        testSet.append(trainingSet[randIndex])
        del(trainingSet[randIndex])
    trainMat=[]; trainClasses = []
    for docIndex in trainingSet:#train the classifier (get probs) trainNB0
        trainMat.append(bagOfWords2VecMN(vocabList, docList[docIndex]))
        trainClasses.append(classList[docIndex])
    p0V,p1V,pSpam = trainNB0(array(trainMat),array(trainClasses))
    errorCount = 0
    for docIndex in testSet:        #classify the remaining items
        wordVector = bagOfWords2VecMN(vocabList, docList[docIndex])
        if classifyNB(array(wordVector),p0V,p1V,pSpam) != classList[docIndex]:
            errorCount += 1
    print('the error rate is: ',float(errorCount)/len(testSet))
    return vocabList,p0V,p1V

def getTopWords(ny,sf):
    import operator
    vocabList,p0V,p1V=localWords(ny,sf)
    topNY=[]; topSF=[]
    for i in range(len(p0V)):
        if p0V[i] > -6.0 : topSF.append((vocabList[i],p0V[i]))
        if p1V[i] > -6.0 : topNY.append((vocabList[i],p1V[i]))
    sortedSF = sorted(topSF, key=lambda pair: pair[1], reverse=True)
    print("SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**SF**")
    for item in sortedSF:
        print(item[0])
    sortedNY = sorted(topNY, key=lambda pair: pair[1], reverse=True)
    print("NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**NY**")
    for item in sortedNY:
        print(item[0])
```

```
testingNB()
['free', 'pizza', 'coupon'] classified this is a spam
['I', 'will', 'miss', 'free', 'pizza'] classified this is not a spam
```