Jeremy Midvidy, jam658
EECS 349, Fall 2017
Assignment 5

3.

Equation six and seven will always produce the same results, given no issues with underflow or precision.

This makes sense logically, because equation six and seven pick the smaller argument as its classification for either spam or ham. In each case, in order to calculate the argument, it computes a probability by either multiplying or adding probabilities. In each case, the relation between arguments for p(spam) or p(ham) will still be the same, regardless if each terms were added or multiplied together.

p(Ham) = p(a1) , p(a2) , p(a3), p(a4), p(a5)
p(Spam) = p(a1) , p(a2) , p(a3), p(a4), p(a5)

For Ham:

p(a1) * p(a2) * p(a3) * p(a4) * p(a5) = hM
p(a1) + p(a2) + p(a3) + p(a4) + p(a5) = hA

For Spam:

p(a1) * p(a2) * p(a3) * p(a4) * p(a5) = sM
p(a1) + p(a2) + p(a3) + p(a4) + p(a5) = sA

The relations between argument will remain the same, such that,

If hM < sM, then hA < sA

AND if hM > sM, then hA > sA

This can be checked with a quick example in python:

```
a = range(1,10)
b = range(41, 50)

aSum = 0
for row in a:
    aSum = aSum + row
bSum = 0
for row in b:
    bSum = bSum + row
```

```
aMult = 1
for row in a:
    aMult = aMult*row
bMult = 1
for row in b:
    bMult = bMult*row

print("aSum is " + str(aSum) + " / bSum is " + str(bSum))
print("\n")
print("aMult is " + str(aMult) + " / bMult is " + str(bMult))
```

Returns:

aSum is 45 / bSum is 405

aMult is 362880 / bMult is 745520860465920

Which illustrates how the relations aren't affected by summing or multiplying the terms.

IF there is underflow, then this relationship could not hold, because a computer would start to truncate numbers and thus change the computations. Multiplying and adding could yield different results, and then the relations would not hold, so then equations (6) and (7) might not always yield the correct result.

4.

Experiment:

My experiment was simple. I created an altered version of the submitted spamfilter.py script (**Appendix**) which added two new functions and then altered `spamsort()`.

The first new function was `experiment()`. This function took, as inputs, a directory full of unsorted mail, a directory full of `known_spam`, a directory full of `known_ham`, a dictionary of probabilities made by a `trainingSet`, and the `priorProb` of spam.

With these inputs, it sorted the mail from the unsorted directory into two new directories, `sorted_spam` and `sorted_ham`. It then compared the `sorted_spam` and `sorted_ham` to known_spam and known_ham, and returned how many in the known sets were in the sorted set.

This showed how well the spamfilter classified the data with my programmed bayesian classifier.

The second new function was `runExperiment()`. This function called `experiment()` 4 times, each time with different inputs.

The inputs were:

DataSet	Filter Type
Same as `trainingSet`	prevProb
Same as `trainingSet`	Bayesian
Different from `trainingSet`	prevProb
Different from `trainingSet`	Bayesian

So the experiment ran 4 times. The first two runs were on a directory full of unsorted mail that was all the mail which the program trained on (created it's dictionary). And the second two runs were on a directory full of unsorted mail that was all different than the mail which the program trained on.

The `spamsort()` was a slight alteration from the submitted script. It took one extra input, `filterType`. If `filterType == 1`, it behaved normally, and called the naive bayesian classifier to classify mail in the unsorted directory.

If `filterType == 0`, it used `prevProb` to classify mail. This process was simple. If `filterType == 0`, the program created an array in `range(0, len(mail))`. Each element in the list corresponded to an index in the mail list. Then, I called `shuffle()` on the list, and randomly reorganized the list of indexes. Then, with the list of random indexes of mail, I computed a `threshold` value, whereby `threshold = len(mail)*prevProb`. This

way, `threshold` = the number of items in mail that should be sorted as spam based on previous probability of an email being spam.

Using the list of random indexes, I sorted the mail from `0-threshold` as spam, and from `threshold-len(mail)` as ham.

Each time the experiment ran, I printed out the number of times and percentages each run of the experiment correctly classified spam and ham, given the data set.

These were the printed results:

---
same as data trained on / PrevProb

For this experiment, there were 3048 total emails.

Out of 2551 total known HAMS, spamsort classified 2127 correctly, or 83%.

Out of 497 total known SPAMS, spamsort classified 73 correctly, or 14%.
---
same as data trained on / Bayesian Filter

For this experiment, there were 3048 total emails.

Out of 2551 total known HAMS, spamsort classified 2333 correctly, or 91%.

Out of 497 total known SPAMS, spamsort classified 363 correctly, or 73%.

Not same as data trained on / PrevProb

For this experiment, there were 3898 total emails.

Out of 2501 total known HAMS, spamsort classified 2087 correctly, or 83%.

Out of 1397 total known SPAMS, spamsort classified 222 correctly, or 15%.

Not same as data trained on / Bayesian Filter

For this experiment, there were 3898 total emails.

Out of 2501 total known HAMS, spamsort classified 2294 correctly, or 91%.

Out of 1397 total known SPAMS, spamsort classified 439 correctly, or 31%.

A.

The mail sets were from the cs.mining website, and were, for each experiment:

Same as `trainingSet`:	Different from `trainingSet`
Spam: *20030288.spam*	Spam: *20050311.spam*
Ham: *20021010_easy_ham*	Ham: *20030255.ham*

For two of the runs of the experiment, the sets were the same as the `trainingSet`, and for the other two runs of the experiment, the sets were different than the ones trained on.

I chose these data sets because I wanted to see how the results varied when the mail was sorted with the exact same mail as it trained on, and an entirely different data set.

In the data which is the same as the `trainingSet`, there were 3048 emails, of which 2551 were ham and 497 were spam, or 84% ham, 16% spam. In the data which is different from the `trainingSet`, there were 3898 total emails, of which 2501 were ham and 1397 were spam, or 64% ham, 36% spam.

Looking through the spam emails in each data set, the emails are each fairly representative of the kinds of spam emails one might encounter in the real world. The type of email ranges the three levels of spam. The first level is just mean to clutter someone's inbox: which are just emails of random words, characters, and strings. The second level is spam that is just meant to try and trick people into thinking it a legitimate message like "refinance your mortgage" emails. And the third level is finicky ways of formatting the second level like "refINanCe ur m0rtgAg3." Looking through the spam emails in each data set, it looks like each of these type of spam email were represented throughout the emails in each set.

B.

After each run of the experiment, I looked through the directories which the filter sorted mail into spam or ham. I then checked each email in each sorted_spam or sorted_ham directory to see if the corresponding email was also in the known_spam or known_ham. If an email was in known_spam and was classified into sorted_spam, then that email was classified correctly. If it was in known_ham and was classified into sorted_ham, then that email was classified correctly. I went through both sorted_ham and sorted_spam, for each trial, and got a count of the correctly sorted hams and spams.

I did not chose to do cross validation. In this problem set, we are trying to gauge how well a learned dictionary classifies unsorted sets of emails into spams and hams. If we split up the data set, we could then potentially be separating types of spam emails into unequal groups, which would mean that some groups (especially when the data set is not the same as the trainingSet) could have undue low scores of classifying spam and ham, which would then

misrepresent the effectiveness of the spam filter over all the folds, because the likelihood that each fold have similar types of spam email would not be evenly distributed.

This is also why I chose to have my experiment run on two different data sets, once comparing the effectiveness of the spam filter on the set it trained on, and once comparing the effectiveness of the spam filter on an entirely different set of emails.

This made sure we got some sort of multi-set validation without improperly splitting up the data within sets, which would then cause the ultimate score of the subsets to be vastly different than the score of the overall sets because different spam emails had a much higher probability of being correctly processed by the spam filter.

With my methodology, the problem of how to divvy up different types of spam emails into each fold was avoided, while also providing insight on how the learned dictionary performed on the data set it learned on as well as a completely different data set.

C.

Here are the results of my experiment in a table:

	PrevProb	Bayes
Same as trained	83% ham / 14% spam	91% ham / 73% spam
Different from trained	83% ham / 15% spam	91% ham / 31% spam

The bayesian spam filter works ***much*** better than using PrevProb. This is true both when the mail to be sorted is the same as when it trained and different from the trainingset.

When the data was the same as the set the dictionary trained on, the Bayesian classifier correctly classified 73% of spam mail. On the other hand, prevProb only correctly classified 14%. Those numbers alone should convince anyone of the effectiveness of the bayesian filter.

But logically, the bayesian filter is a much more intelligent way of classifying emails. It constructs a dictionary of words in the training set's spam and ham emails. It then assign probabilities to each word for occurring in spam and ham. It then uses these probabilities to classify words in new emails as either spam or ham. Clearly this is a much more sophisticated and agile method of classifying emails vs. just randomly picking emails and using a prior probability to choose. 73% correctly classified is really good, especially when you consider that prevProb only classified 14% correctly.

While the data from when the testing set was different than the trainingset was less impressive, it still worked twice as well. Considering that the bayesian filter knew absolutely nothing about the set of emails it was looking through, and there is a good chance that there were plenty of words in spam emails (out of the 1497 spam emails) that could may not have been in the learned directory (considering it only trained on 497 spam emails) than it is understandable that the spam filter, as it is currently written, did not work extremely well on a data set different than the one it trained on. Still, with all that being said, the bayesian classifier

worked 2x as well as the prevProb when operated on a data set different than the trained set, so it's clearly a much better way of classifying emails.

So, in cases when the data set is the same as the trainngSet or different, the bayesian filter does better than using the prior probability of of spam.

Appendix

Python code for experiment with new functions and altered spamsort

```
#Starter code for spam filter assignment in EECS349 Machine Learning
#Author: Jeremy Midvidy, jam658.  EECS 349 HW#5

import sys
import numpy as np
import os
import shutil
import math
import csv
import scipy as sp, matplotlib as plt
import copy
import random


def parse(text_file):
	#This function parses the text_file passed into it into a set of words. Right now it
just splits up the file by blank spaces, and returns the set of unique strings used in the
file.
	content = text_file.read()
	return np.unique(content.split())

def writedictionary(dictionary, dictionary_filename):
	#Don't edit this function. It writes the dictionary to an output file.
	output = open(dictionary_filename, 'w')
	header = 'word\tP[word|spam]\tP[word|ham]\n'
	output.write(header)
	for k in dictionary:
		line = '{0}\t{1}\t{2}\n'.format(k, str(dictionary[k]['spam']),
str(dictionary[k]['ham']))
		output.write(line)


def makedictionary(spam_directory, ham_directory, dictionary_filename):
    #Making the dictionary.
    spam = [f for f in os.listdir(spam_directory) if
os.path.isfile(os.path.join(spam_directory, f))]
    ham = [f for f in os.listdir(ham_directory) if os.path.isfile(os.path.join(ham_directory,
f))]

    spam_prior_probability = len(spam)/float((len(spam) + len(ham)))

    words = {}

    #These for loops walk through the files and construct the dictionary. The dictionary,
words, is constructed so that words[word]['spam'] gives the probability of observing that
word, given we have a spam document P(word|spam), and words[word]['ham'] gives the probability
of observing that word, given a hamd document P(word|ham). Right now, all it does is
initialize both probabilities to 0. TODO: add code that puts in your estimates for
P(word|spam) and P(word|ham).
    for s in spam:
```

```
        for word in parse(open(spam_directory + s)):
            if word not in words:
                words[word] =  {'spam': 1, 'ham': 1}
                break
            else:
                words[word]['spam'] = words[word]['spam'] + 1
                break

    for h in ham:
        for word in parse(open(ham_directory + h)):
            if word not in words:
                words[word] =  {'spam': 1, 'ham': 1}
                break
            else:
                words[word]['ham'] = words[word]['ham'] + 1
                break

    #now have the number of raw times a word occurs in a document
    #need to convert to probabilities

    spam_denom = float(len(spam)) + 2
    ham_denom = float(len(ham)) + 2

    for key in words:
        s = words[key]['spam'] / spam_denom
        h = words[key]['ham'] / ham_denom
        words[key] = {'spam' : s, 'ham' : h}


    #Write it to a dictionary output file.
    writedictionary(words, dictionary_filename)

    return words, spam_prior_probability


def is_spamFilter(content, dictionary, spam_prior_probability):
       #TODO: Update this function. Right now, all it does is checks whether the
spam_prior_probability is more than half the data. If it is, it says spam for everything.
Else, it says ham for everything. You need to update it to make it use the dictionary and the
content of the mail. Here is where your naive Bayes classifier goes.


    Vj = spam_prior_probability

    contentHamDict = {}
    contentSpamDict = {}

    #make sure to test for edge case where word is not in the document
    #construc a dictionary of values where each key is the pS or pH of each word
    #in content
    for row in content:
        if row in dictionary:
            h = dictionary[row]['ham']
            s = dictionary[row]['spam']
            contentHamDict[row] = h
            contentSpamDict[row] = s
```

```
#edge case where none of the words in an email are in the dictionary
#then just classify isSpam based on prevProb
if (len(contentHamDict) == 0):
    if Vj > .5:
        return True
    else:
        return False


probsHam = []
probsSpam = []
#compute the log10 of probability in spam and ham
for row in contentHamDict:
    pH = math.log10(contentHamDict[row])
    pS = math.log10(contentSpamDict[row])

    probsHam.append(pH)
    probsSpam.append(pS)

#need to find the prob that a message is not in spam/ham.  1-
notpHam = []
notpSpam = []

for row in contentHamDict:
    pH = math.log10(1-contentHamDict[row])
    pS = math.log10(1-contentSpamDict[row])

    notpHam.append(pH)
    notpSpam.append(pS)

##########################################
#these are the probs that a message is IN the spam/ha,
pHam = probsHam[0] * notpSpam[0]
pSpam = probsSpam[0] * notpHam[0]



#summate all the probabilities together
for x in range(1, len(probsHam)):
    pHam = pHam * probsHam[x] * notpSpam[x]
for x in range(1, len(probsSpam)):
    pSpam = pSpam * probsSpam[x] * notpHam[x]

pHam = pHam * math.log10(1-Vj)
pSpam = pSpam * math.log10(Vj)


#return the larger probabilitiy, T for isSpam
if pSpam > pHam:
    return True
else:
    return False
```

```
def spamsort(filterType, mail_directory, spam_directory, ham_directory, dictionary,
spam_prior_probability):
    mail = [f for f in os.listdir(mail_directory) if
os.path.isfile(os.path.join(mail_directory, f))]

    #use bayesian filter
    if filterType == 1:
        for m in mail:
            content = parse(open(mail_directory + m))
            spam = is_spamFilter(content, dictionary, spam_prior_probability)
            if spam:
                shutil.copy(mail_directory + m, spam_directory)
            else:
                shutil.copy(mail_directory + m, ham_directory)


    else: #use prevProb
        #create a random list of indexes in mail
        x = range(0, len(mail))
        random.shuffle(x)

        #classify as Spam until threshold is reached where the rest
        #should be classified as ham.
        threshold = int(spam_prior_probability*len(mail))
        for r in range(0, len(x)):
            ind = x[r]
            if r < threshold:
                shutil.copy(mail_directory + mail[ind], spam_directory)
            else:
                shutil.copy(mail_directory + mail[ind], ham_directory)



def experiment(words, priorProb, filterType, experimentNum):

    expNum = experimentNum
    filType = filterType

    ##############################################################################

    #deleting files in sorted directories from old experiements
    Sorted_SpamFiles = "C:\Users\jmidv\Miniconda2\envs\eecs349\experiment_sorted_spam\\"
    folder = Sorted_SpamFiles
    for the_file in os.listdir(folder):
        file_path = os.path.join(folder, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
        except Exception as e:
            print(e)

    Sorted_HamFiles = "C:\Users\jmidv\Miniconda2\envs\eecs349\experiment_sorted_ham\\"
    folder = Sorted_HamFiles
    for the_file in os.listdir(folder):
        file_path = os.path.join(folder, the_file)
        try:
```

```
            if os.path.isfile(file_path):
                os.unlink(file_path)
        except Exception as e:
            print(e)



    ##############################################################################################

    #creating directiores depending on experiment number

    experiment_all_mail = "C:\Users\jmidv\Miniconda2\envs\eecs349\\"
    experiment_spam_knowns = "C:\Users\jmidv\Miniconda2\envs\eecs349\\"
    experiemnt_ham_knowns = "C:\Users\jmidv\Miniconda2\envs\eecs349\\"

    if expNum == 1: #not same as trained
        experiment_all_mail = experiment_all_mail + "EXP_1\EXP_1_All\\"
        experiment_spam_knowns = experiment_spam_knowns + "EXP_1\EXP_1_Spam\\"
        experiemnt_ham_knowns = experiemnt_ham_knowns +"EXP_1\EXP_1_Ham\\"
    else:
        experiment_all_mail = experiment_all_mail + "EXP_2\EXP_2_All\\"
        experiment_spam_knowns = experiment_spam_knowns + "EXP_2\EXP_2_Spam\\"
        experiemnt_ham_knowns = experiemnt_ham_knowns + "EXP_2\EXP_2_Ham\\"



    experiment_sorted_spam =  "C:\Users\jmidv\Miniconda2\envs\eecs349\experiment_sorted_ham\\"
    experiment_sorted_ham =  "C:\Users\jmidv\Miniconda2\envs\eecs349\experiment_sorted_spam\\"

    ##############################################################################################
    #filterType = 1 for bayesianFilter, 0 for prevProb

    spamsort(filType, experiment_all_mail, experiment_sorted_spam, experiment_sorted_ham,
words, priorProb)


    #need to see how many files in sorted are in known for both spam and ham


    ##############################################################################################
    #go through spam and ham directories and see how many were correclty classified

    spamKnown = [f for f in os.listdir(experiment_spam_knowns) if
os.path.isfile(os.path.join(experiment_spam_knowns, f))]
    spamTest = [f for f in os.listdir(experiment_sorted_spam) if
os.path.isfile(os.path.join(experiment_sorted_spam, f))]

    hamKnown = [f for f in os.listdir(experiemnt_ham_knowns) if
os.path.isfile(os.path.join(experiemnt_ham_knowns, f))]
    hamTest = [f for f in os.listdir(experiment_sorted_ham) if
os.path.isfile(os.path.join(experiment_sorted_ham, f))]

    #find how many of test are in each knowns
    numSpam = 0
    numHam = 0

    for row in hamTest:
```

```
        if row in hamKnown:
            numHam = numHam + 1

    for row in spamTest:
        if row in spamKnown:
            numSpam = numSpam + 1

    ##############################################################################################

    #print out results of experiment

    totalSpam = int (100 * (numSpam / float(len(spamKnown))))
    totalHam =  int (100 * (numHam  / float(len(hamKnown))))

    print("\n")
    print("-------------------------------------------------------")
    fil = ""
    if filType == 1:
        fil = "Bayesian Filter"
    else:
        fil = "PrevProb"

    data = ""
    if expNum == 1:
        data = "Not same as data trained on"
    else:
        data = "same as data trained on"

    print(data + " / "+ fil)
    print("\n")
    print("For this experiment, there were " + str(len(spamKnown) + len(hamKnown)) + " total
emails.")
    print("\n")
    print("Out of " + str(len(hamKnown)) + " total known HAMS, spamsort classified " +
str(numHam) + " correctly, or " + str(totalHam) + "%." )
    print("\n")
    print("Out of " + str(len(spamKnown)) + " total known SPAMS, spamsort classified " +
str(numSpam) + " correctly, or " + str(totalSpam) + "%." )
    print("\n")


def runExperiment(words, spp):

    expNum = [0,1]
    filType = [0,1]

    for row in expNum:
        for line in filType:
            experiment(words, spp, line, row)


if __name__ == "__main__":
   training_spam_directory = "C:\Users\jmidv\Miniconda2\envs\eecs349\spam\\"
   training_ham_directory = "C:\Users\jmidv\Miniconda2\envs\eecs349\ham\\"

   test_mail_directory = "C:\Users\jmidv\Miniconda2\envs\eecs349\mail\\"
```

```
   test_spam_directory = "C:\Users\jmidv\Miniconda2\envs\eecs349\sorted_spam\\"
   test_ham_directory = "C:\Users\jmidv\Miniconda2\envs\eecs349\sorted_ham\\"

   if not os.path.exists(test_spam_directory):
       os.mkdir(test_spam_directory)
   if not os.path.exists(test_ham_directory):
        os.mkdir(test_ham_directory)


   dictionary_filename = "dictionary.dict"
   dictionary, spam_prior_probability = makedictionary(training_spam_directory,
training_ham_directory, dictionary_filename)



   #run the experiemnt on the sorted data
   runExperiment(dictionary, spam_prior_probability)
```