assignment-4

October 6, 2023

1 DAT565 Introduction to Data Science and AI

1.1 2023-2024, LP1

1.2 Assignment 4: Spam classification using Naïve Bayes

This assignment has three obligatory questions. Questions 4-5 are optional and will not be graded.

The exercise takes place in this notebook environment where you can chose to use Jupyter or Google Colabs. We recommend you use Google Colabs as it will facilitate remote group-work and makes the assignment less technical.

Tips: * You can execute certain Linux shell commands by prefixing the command with a `!`. * You can insert Markdown cells and code cells. The first you can use for documenting and explaining your results, the second you can use to write code snippets that execute the tasks required.

In this assignment you will implement a Naïve Bayes classifier in Python that will classify emails into spam and non-spam ("ham") classes. Your program should be able to train on a given set of spam and "ham" datasets.

You will work with the datasets available at https://spamassassin.apache.org/old/publiccorpus/. There are three types of files in this location: - easy-ham: non-spam messages typically quite easy to differentiate from spam messages. - hard-ham: non-spam messages more difficult to differentiate - spam: spam messages

Execute the cell below to download and extract the data into the environment of the notebook – it will take a few seconds.

If you chose to use Jupyter notebooks you will have to run the commands in the cell below on your local computer. Note that if you are using Windows, you can instead use (7zip)[https://www.7-zip.org/download.html] to decompress the data (you will have to modify the cell below).

What to submit: * Convert the notebook to a PDF file by compiling it, and submit the PDF file. * Make sure all cells are executed so all your code and its results are included. * Double-check that the PDF displays correctly before you submit it.

[2]:
```
# download and extract the data
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_hard_ham.tar.bz2
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.bz2
!tar -xjf 20021010_easy_ham.tar.bz2
!tar -xjf 20021010_hard_ham.tar.bz2
```

```
!tar -xjf 20021010_spam.tar.bz2
```

```
--2023-10-05 20:57:48--
https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2
    spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
   spamassassin.apache.org (spamassassin.apache.org)|151.101.2.132|:443…

   HTTP      … 200 OK
  1677144 (1.6M) [application/x-bzip2]
   : "20021010_easy_ham.tar.bz2"

20021010_easy_ham.t 100%[===================>]   1.60M  --.-KB/s    0.08s

2023-10-05 20:57:49 (19.0 MB/s) -   "20021010_easy_ham.tar.bz2"
[1677144/1677144])

--2023-10-05 20:57:49--
https://spamassassin.apache.org/old/publiccorpus/20021010_hard_ham.tar.bz2
    spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
   spamassassin.apache.org (spamassassin.apache.org)|151.101.2.132|:443…

   HTTP      … 200 OK
  1021126 (997K) [application/x-bzip2]
   : "20021010_hard_ham.tar.bz2"

20021010_hard_ham.t 100%[===================>] 997.19K  --.-KB/s    0.07s

2023-10-05 20:57:49 (14.3 MB/s) -   "20021010_hard_ham.tar.bz2"
[1021126/1021126])

--2023-10-05 20:57:49--
https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.bz2
    spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
   spamassassin.apache.org (spamassassin.apache.org)|151.101.2.132|:443…

   HTTP      … 200 OK
  1192582 (1.1M) [application/x-bzip2]
   : "20021010_spam.tar.bz2"

20021010_spam.tar.b 100%[===================>]   1.14M  --.-KB/s    0.08s

2023-10-05 20:57:49 (14.9 MB/s) -   "20021010_spam.tar.bz2" [1192582/1192582])
```

The data is now in the following three folders: `easy_ham`, `hard_ham`, and `spam`. You can confirm this via the following command:

[3]:
```
!ls -lah
```

```
total 7640
drwxr-xr-x    10 yuchuan.dong  staff   320B Sep 21 14:30 .
drwxr-xr-x    11 yuchuan.dong  staff   352B Sep 21 14:24 ..
drwxr-xr-x     3 yuchuan.dong  staff    96B Sep 21 14:25
.ipynb_checkpoints
-rw-r--r--     1 yuchuan.dong  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2
-rw-r--r--     1 yuchuan.dong  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2
-rw-r--r--     1 yuchuan.dong  staff   1.1M Jun 29  2004 20021010_spam.tar.bz2
-rw-r--r--@    1 yuchuan.dong  staff   8.5K Sep 21 14:29 assignment-4.ipynb
drwx--x--x  2553 yuchuan.dong  staff    80K Oct 10  2002 easy_ham
drwx--x--x   252 yuchuan.dong  staff   7.9K Dec 16  2004 hard_ham
drwxr-xr-x   503 yuchuan.dong  staff    16K Oct 10  2002 spam
```

1.2.1 1. Preprocessing:

Note that the email files contain a lot of extra information, besides the actual message. Ignore that for now and run on the entire text (in the optional part further down, you can experiment with filtering out the headers and footers). 1. We don't want to train and test on the same data (it might help to reflect on **why** ,if you don't recall). Split the spam and ham datasets into a training set and a test set. (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`). Use `easy_ham` for quesions 1 and 2.

[2]:
```
from sklearn.model_selection import train_test_split
import os
import sys
import numpy as np
import matplotlib.pyplot as plt
from bs4 import BeautifulSoup

os.chdir(sys.path[0])

def ReadEmails(directory):
    Emails=[]
    for directory in directory:
        for file in os.listdir(directory):
            with open(os.path.join(directory,file),encoding='latin-1') as␣
 ↪EmailFile:
                data = EmailFile.read()
                soup = BeautifulSoup(data, 'html.parser')
                Emails.append(soup.get_text())
            EmailFile.close()
    return Emails
Easy_ham = ReadEmails(['./easy_ham'])
Hard_ham = ReadEmails(['./hard_ham'])
Spam = ReadEmails(['./spam'])
Ham_Label=["ham" for i in range(len(Easy_ham))]
```

```
Spam_Label=["spam" for i in range(len(Spam))]
Hardham_Label=["hardham" for i in range(len(Hard_ham))]
hamtrain, hamtest, Hamtrain_labels, Hamtest_labels = train_test_split(Easy_ham,␣
↪Ham_Label, train_size=0.8, random_state=1)
spamtrain, spamtest, Spamtrain_labels, Spamtest_labels = train_test_split(Spam,␣
↪Spam_Label, train_size=0.8, random_state=1)
hardhamtrain,hardhamtest, Hardhamtrain_labels, Hardhamtest_labels =␣
↪train_test_split(Hard_ham, Hardham_Label, train_size=0.8, random_state=1)
```

1.2.2 2. Write a Python program that:

1. Uses the four datasets from Question 1 (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`).
2. Trains a Naïve Bayes classifier (use the scikit-learn library) on `hamtrain` and `spamtrain`, that classifies the test sets and reports True Positive and False Negative rates on the `hamtest` and `spamtest` datasets. You can use `CountVectorizer` (documentation here) to transform the email texts into vectors. Please note that there are different types of Naïve Bayes Classifiers available in *scikit-learn* (Documentation here). Here, you will test two of these classifiers that are well suited for this problem:

- Multinomial Naive Bayes
- Bernoulli Naive Bayes.

Please inspect the documentation to ensure input to the classifiers is appropriate before you start coding. You may have to modify your input.

[3]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import confusion_matrix, classification_report
# merge datasets
X_train=hamtrain+spamtrain
X_test=hamtest+spamtest
y_train=Hamtrain_labels+Spamtrain_labels
y_test=Hamtest_labels+Spamtest_labels
# transform the email texts into vectors
vectorizer=CountVectorizer(lowercase=False)

X_train_vector=vectorizer.fit_transform(X_train).toarray()
feature_names = vectorizer.get_feature_names_out()
X_test_vector=vectorizer.transform(X_test).toarray()
# train model
Multinomial_Naive_Bayes = MultinomialNB().fit(X_train_vector, y_train)
Bernoulli_Naive_Bayes = BernoulliNB().fit(X_train_vector, y_train)
# classify test sets
y_pred_multinomial = Multinomial_Naive_Bayes.predict(X_test_vector)
y_pred_bernoulli = Bernoulli_Naive_Bayes.predict(X_test_vector)
# print(len(y_test))
# print(len(y_pred_bernoulli))
# True Positive and False Negative rates
```

```python
confusion_multinomial = confusion_matrix(y_test,y_pred_multinomial)
confusion_bernoulli = confusion_matrix(y_test,y_pred_bernoulli)
print(confusion_multinomial)
print(confusion_bernoulli)
tn_multinomial, fp_multinomial, fn_multinomial, tp_multinomial = confusion_multinomial.ravel()
tn_bernoulli, fp_bernoulli, fn_bernoulli, tp_bernoulli = confusion_bernoulli.ravel()

true_positive_rate_multinomial = tp_multinomial / (tp_multinomial + fn_multinomial)
false_negative_rate_multinomial = fn_multinomial / (tp_multinomial + fn_multinomial)
true_positive_rate_bernoulli = tp_bernoulli / (tp_bernoulli + fn_bernoulli)
false_negative_rate_bernoulli = fn_bernoulli / (tp_bernoulli + fn_bernoulli)
print('true_positive_rate_multinomial:',true_positive_rate_multinomial)
print('false_negative_rate_multinomial:',false_negative_rate_multinomial)
print('true_positive_rate_bernoulli:',true_positive_rate_bernoulli)
print('false_negative_rate_bernoulli:',false_negative_rate_bernoulli)
```

```
[[511   0]
 [ 12  89]]
[[509   2]
 [ 62  39]]
true_positive_rate_multinomial: 0.8811881188118812
false_negative_rate_multinomial: 0.1188118811881188
true_positive_rate_bernoulli: 0.38613861386138615
false_negative_rate_bernoulli: 0.6138613861386139
```

Here is the analytical discussion about the models above

We observed that Multinominal Naive Bayes performed better than that of Bernouli Naive Bayes.

We look into the difference between these two methods from their principles.

For Multinominal Naive Bayes:

the frequency of feature i is

$$N_{yi} = \sum_{x \in X} x_i$$

the frequency of all features of label y is

$$N_y = \sum_{i=1}^{n} N_{yi}$$

So the probability of feature i in the label y is

$$P(X_i|Y) = \frac{N_{yi}}{N_y}$$

and for Bernouli Naive Bayes:

$x_i = 0$ or 1, which means that we consider each word obeys Bernouli distribution and we only consider whether a word appears or not and we don't consider the frequency of it. And the probability of feature i given by label y is:

$$P(X_i|Y) = P(X_i = 1|Y)X_i + P(X_i = 0|Y)(1 - X_i)$$

First, we want to see the words and their frequencies in spam emails and not in ham emails. Because these words can be considered as features that decide whether email is a spam or a ham. And frequencies of words can contribute to the value of $P(X_i|Y = "spam")$, which are used in the Multinominal Naive Bayes.

[4]:
```
y_train = np.array(y_train)

spam_index, ham_index = (y_train == "spam"), (y_train != "spam")
x_train_spam, x_train_ham = X_train_vector[spam_index],␣
↪X_train_vector[ham_index]

# sum the frequency of the word vectors
spam_word_frequency, ham_word_frequency =  np.sum(x_train_spam, axis=0), np.
↪sum(x_train_ham, axis=0)
spam_sort_index, ham_sort_index = np.argsort(spam_word_frequency), np.
↪argsort(ham_word_frequency)

# we choose the words, whose frequencies are in top 200.
topk = 200

spam_feature_names = [feature_names[spam_sort_index[-i]] for i in range(topk)]
ham_feature_names = [feature_names[ham_sort_index[-i]] for i in range(topk)]
spam_word_frequency = [spam_word_frequency[spam_sort_index[-i]] for i in␣
↪range(topk)]
ham_word_frequency = [ham_word_frequency[ham_sort_index[-i]] for i in␣
↪range(topk)]
spam_word = set([feature_names[spam_sort_index[-i]] for i in range(topk)])
ham_word = set([feature_names[ham_sort_index[-i]] for i in range(topk)])
spam_dict = {k:v for k, v in zip(spam_feature_names, spam_word_frequency)}
ham_dict = {k:v for k, v in zip(ham_feature_names, ham_word_frequency)}

intersection = spam_word - ham_word
result_dict = dict()
for key in intersection:
    result_dict[key] = spam_dict[key]
```

```python
print(result_dict)
```

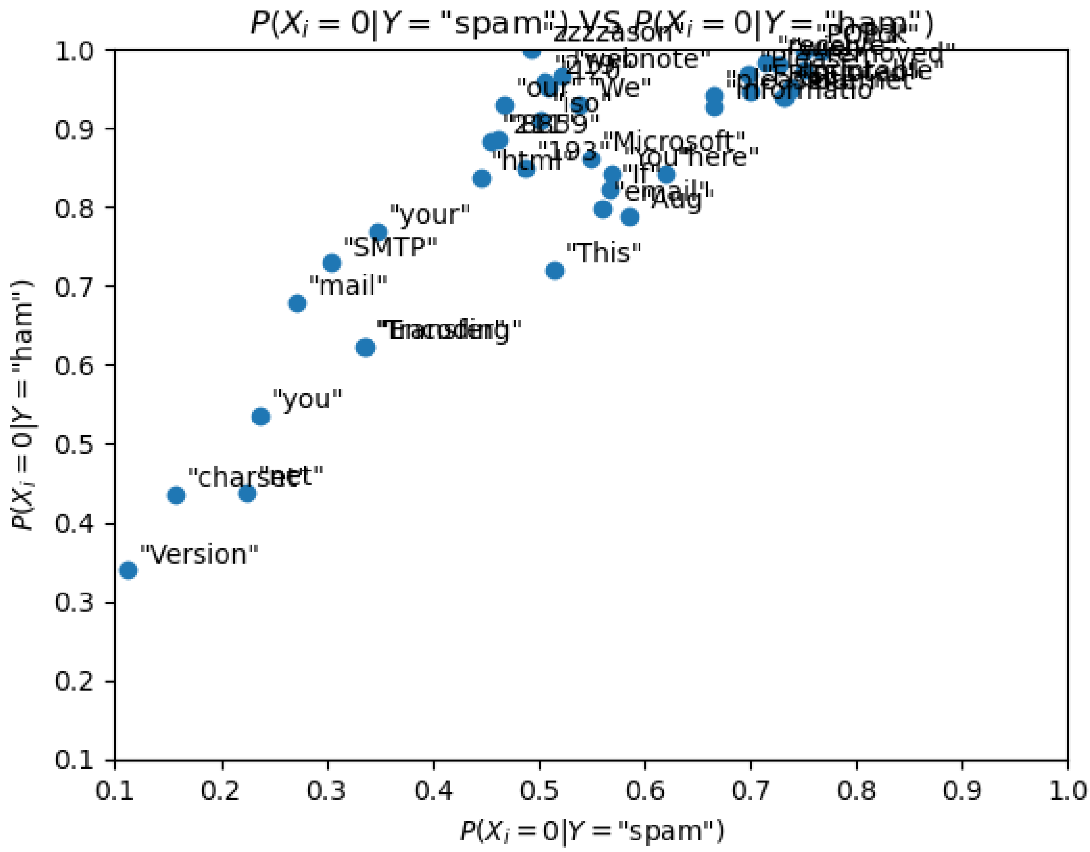
```
{'ilug': 211, 'Normal': 202, 'zzzz': 887, 'wish': 154, 'Internet': 186, 'our':
555, 'Sat': 225, 'me': 242, 'been': 160, 'tuatha': 218, 'If': 318, 'business':
170, '219': 209, 'webnote': 553, 'Microsoft': 276, 'Mailer': 203, '211': 256,
'Encoding': 312, 'We': 407, 'TO': 204, '193': 216, 'You': 431, 'name': 178,
'please': 206, 'address': 187, 'lugh': 250, '8859': 225, 'smtp': 159, 'send':
203, 'Your': 191, 'time': 241, 'who': 213, 'Please': 175, 'people': 271, 'FREE':
289, 'message': 218, 'want': 158, 'Sun': 178, 'html': 239, 'may': 185, 'email':
392, '000': 466, 'make': 210, 'no': 212, '45': 174, '3D': 196, 'information':
249, 'YOU': 165, 'OF': 161, 'iso': 213, 'Priority': 210, 'labs': 167,
'zzzzason': 215, '120': 200, 'yahoo': 164, 'any': 267, '100': 171, 'TNG': 0,
'here': 233, 'money': 354, 'only': 242, 'over': 176, 'THE': 182, 'receive': 241,
'Transfer': 315, 'COM': 168}
```

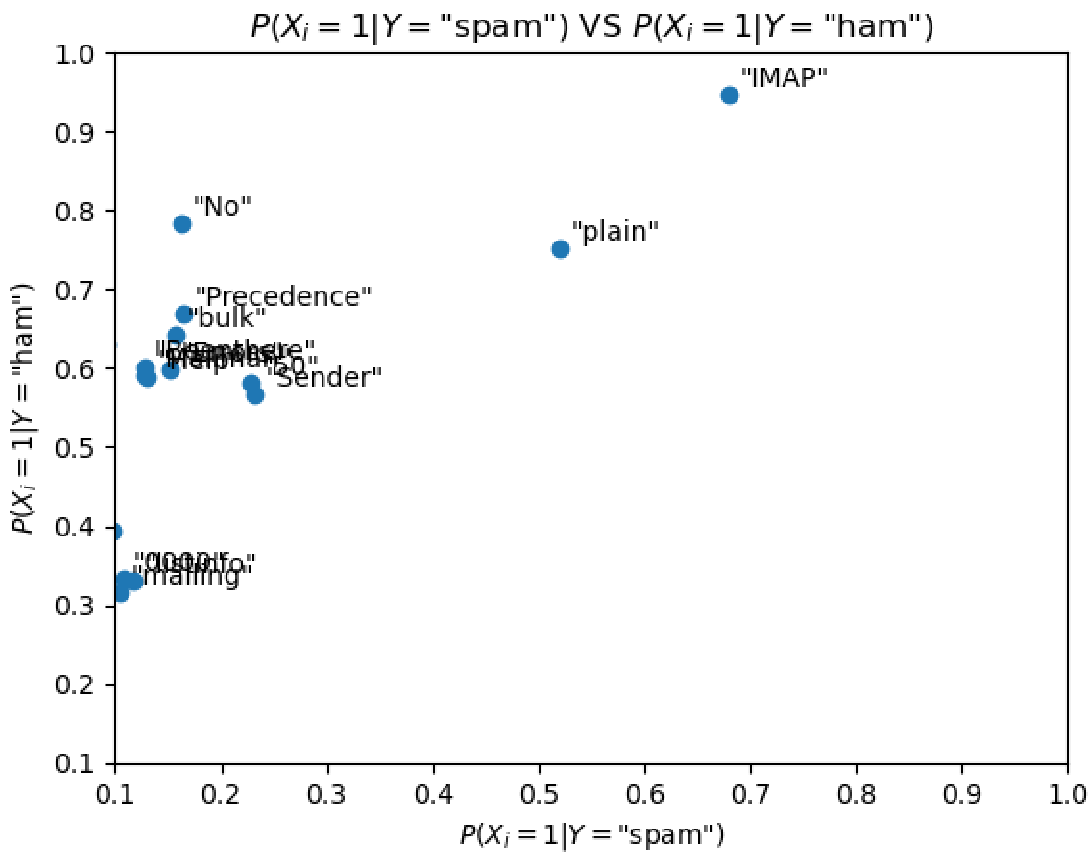
From what we have above, we can see that words such as Microsoft and FREE appearing in the spam email. So, the email containing the word Microsoft tends to be an advertisement from Microsoft company and the email containing the word FREE tend to be an advertisement that may promote something free. These kinds of words help the model, Multinominal Naive Bayes, to decide an email is spam or not.

Then, we want to see the values $P(X_i = 1|Y = "spam")$ versus $P(X_i = 1|Y = "ham")$ and $P(X_i = 0|Y = "spam")$ versus $P(X_i = 0|Y = "ham")$. Because, all these two pairs will make contribution to the model.

[13]:
```python
def versus_plot(spam_train, ham_train, x_i):
    spam_bool_vector = spam_train == x_i
    ham_bool_vector = ham_train == x_i
    # sum all the appearance
    spam_appear_vector, ham_appear_vector = spam_bool_vector.sum(axis=0),␣
 ↪ham_bool_vector.sum(axis=0)

    spam_word_prob = np.array(spam_appear_vector) / len(spam_train)
    ham_word_prob =  np.array(ham_appear_vector) / len(ham_train)

    selected_index = (ham_word_prob > (spam_word_prob+0.2))
    selected_feature = feature_names[selected_index]
    x, y = spam_word_prob[selected_index], ham_word_prob[selected_index]

    plt.scatter(x,y)
    for i in range(len(selected_feature)):
        if len(selected_feature[i]) > 10:
            plt.annotate(f"\"{selected_feature[i][:10]}\"", [x[i], y[i]], [x[i]␣
 ↪+ 0.01, y[i] + 0.01])
        else:
            plt.annotate(f"\"{selected_feature[i]}\"", [x[i], y[i]], [x[i] + 0.
 ↪01, y[i] + 0.01])
    plt.xlabel(f"$P(X_i={x_i}|Y=$\"spam\")")
```

```
    plt.ylabel(f"$P(X_i={x_i}|Y=$\"ham\")")
    plt.title(f"$P(X_i={x_i}|Y=$\"spam\") VS $P(X_i={x_i}|Y=$\"ham\")")
    plt.xlim([0.1,1])
    plt.ylim([0.1,1])
    plt.show()

versus_plot(x_train_spam, x_train_ham, 0)
versus_plot(x_train_spam, x_train_ham, 1)
```





Compared with the Multinominal Naive Bayes, Bernouli Naive Bayes consider the influence of the condition where $x_i = 0$ and we can find from the above graphs that there exists words(features) where $P(X_i = 0|Y = "spam") < P(X_i = 0|Y = "ham")$, which will have an influence for the model and make the latter model perform worse than the former one.

1.2.3 3. Run on hard ham:

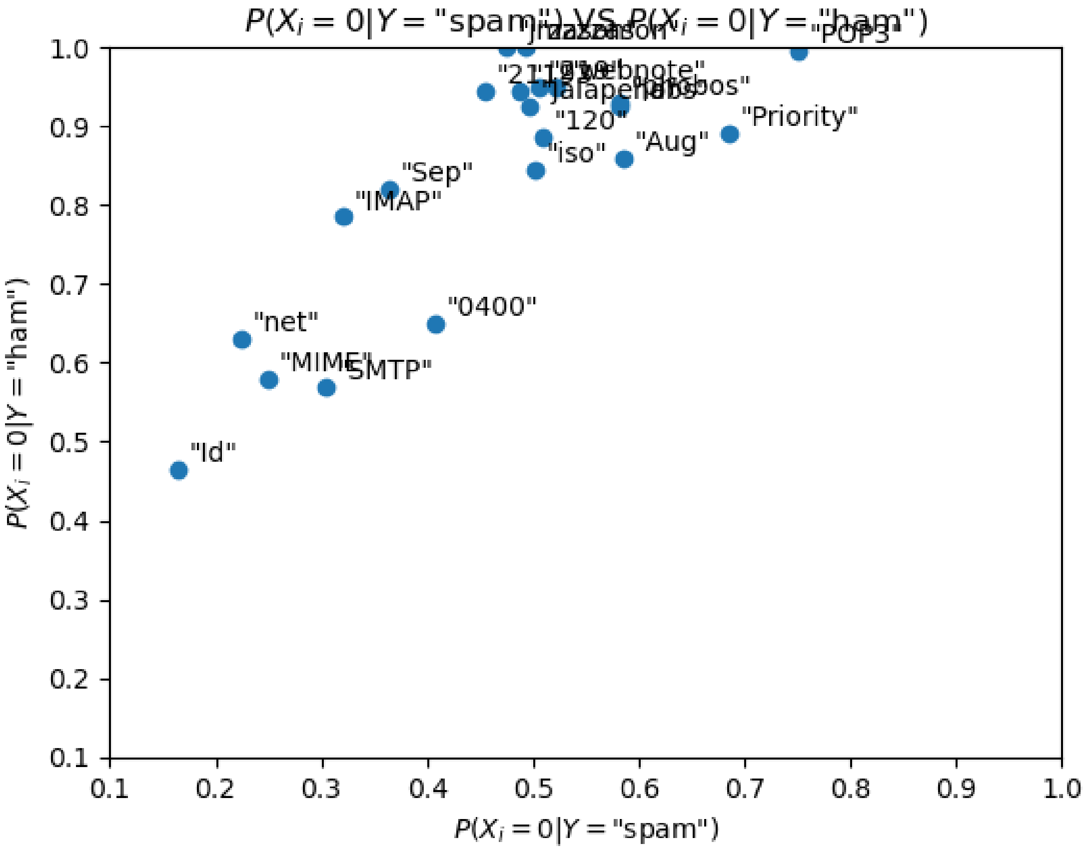
Run the two models from Question 2 on `spam` versus `hard-ham`, and compare to the `easy-ham` results.

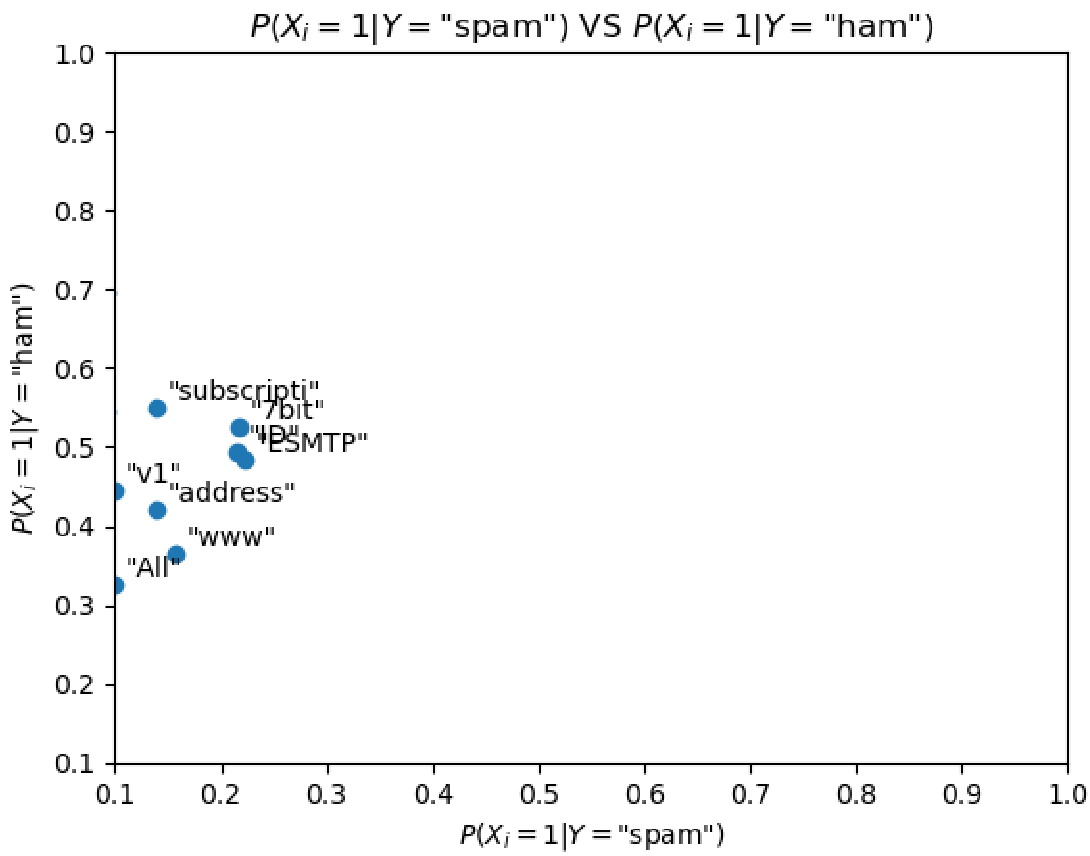
[16]:
```
# code to report results here
X_train=hardhamtrain+spamtrain
X_test=hardhamtest+spamtest
y_train=Hardhamtrain_labels+Spamtrain_labels
y_test=Hardhamtest_labels+Spamtest_labels

# transform the email texts into vectors
vectorizer=CountVectorizer(lowercase=False)
X_train_vector=vectorizer.fit_transform(X_train).toarray()
feature_names = vectorizer.get_feature_names_out()
```

```
X_test_vector=vectorizer.transform(X_test).toarray()
# train model
Multinomial_Naive_Bayes = MultinomialNB().fit(X_train_vector, y_train)
Bernoulli_Naive_Bayes = BernoulliNB().fit(X_train_vector, y_train)

# classify test sets
y_pred_multinomial = Multinomial_Naive_Bayes.predict(X_test_vector)
y_pred_bernoulli = Bernoulli_Naive_Bayes.predict(X_test_vector)
# print(len(y_test))
# print(len(y_pred_bernoulli))
# True Positive and False Negative rates
confusion_multinomial = confusion_matrix(y_test,y_pred_multinomial)
confusion_bernoulli = confusion_matrix(y_test,y_pred_bernoulli)
print(confusion_multinomial)
print(confusion_bernoulli)
tn_multinomial, fp_multinomial, fn_multinomial,␣
 ↪tp_multinomial,confusion_multinomial.ravel()
tn_bernoulli, fp_bernoulli, fn_bernoulli, tp_bernoulli = confusion_bernoulli.
 ↪ravel()
true_positive_rate_multinomial = tp_multinomial / (tp_multinomial +␣
 ↪fn_multinomial)
false_negative_rate_multinomial = fn_multinomial / (tp_multinomial +␣
 ↪fn_multinomial)
true_positive_rate_bernoulli = tp_bernoulli / (tp_bernoulli + fn_bernoulli)
false_negative_rate_bernoulli = fn_bernoulli / (tp_bernoulli + fn_bernoulli)
print('true_positive_rate_multinomial:',true_positive_rate_multinomial)
print('false_negative_rate_multinomial:',false_negative_rate_multinomial)
print('true_positive_rate_bernoulli:',true_positive_rate_bernoulli)
print('false_negative_rate_bernoulli:',false_negative_rate_bernoulli)
```

```
[[43  7]
 [ 2 99]]
[[34 16]
 [ 2 99]]
true_positive_rate_multinomial: 0.8811881188118812
false_negative_rate_multinomial: 0.1188118811881188
true_positive_rate_bernoulli: 0.9801980198019802
false_negative_rate_bernoulli: 0.019801980198019802
```

[17]:
```
y_train = np.array(y_train)

spam_index, ham_index = (y_train == "spam"), (y_train != "spam")
x_train_spam, x_train_ham = X_train_vector[spam_index],␣
 ↪X_train_vector[ham_index]
versus_plot(x_train_spam, x_train_ham, 0)
versus_plot(x_train_spam, x_train_ham, 1)
```


$P(X_i = 0|Y = \text{"spam"})$ VS $P(X_i = 0|Y = \text{"ham"})$
"POP3"
"Priority"
"Aug"
"iso"
"120"
"Sep"
"IMAP"
"0400"
"net"
"MIME"
"SMTP"
"Id"
$P(X_i = 0|Y = \text{"ham"})$
$P(X_i = 0|Y = \text{"spam"})$
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0



After changing the dataset, we find that although there exists words(features) where $P(X_i = 0|Y = \text{"}spam\text{"}) < P(X_i = 0|Y = \text{"}ham\text{"})$, but the number of those features is much less than the EASY-HAM dataset, which will cause less influence or the penalty in the Bernouli Naive Bayes.

1.2.4 4. OPTIONAL - NOT MARKED:

To avoid classification based on common and uninformative words, it is common practice to filter these out.

a. Think about why this may be useful. Show a few examples of too common and too uncommon words.

b. Use the parameters in *scikit-learn*'s `CountVectorizer` to filter out these words. Update the program from Question 2 and run it on `easy-ham` vs `spam` and `hard-ham` vs `spam`. Report your results.

[ ]:
```
# write your code here
```

1.2.5 5. OPTIONAL - NOT MARKED: Further improving performance

Filter out the headers and footers of the emails before you run on them. The format may vary somewhat between emails, which can make this a bit tricky, so perfect filtering is not required.

Run your program again and answer the following questions: - Does the result improve from those obtained in Questions 3 and 4? - What do you expect would happen if your training set consisted mostly of spam messages, while your test set consisted mostly of ham messages, or vice versa? - Look at the `fit_prior` parameter. What does this parameter mean? Discuss in what settings it can be helpful (you can also test your hypothesis).

[ ]:
```
# write your code here
```