DAT405_Assignment4_final

September 27, 2021

1 DAT405 Introduction to Data Science and AI

1.1 2021-2022, Reading Period 1

1.2 Assignment 4: Spam classification using Naïve Bayes

Authors: Lukas Martinsson, Sidner Magneli Time spent: Lukas 16h, Sidner 16h

There will be an overall grade for this assignment. To get a pass grade (grade 5), you need to pass items 1-3 below. To receive higher grades, finish items 4 and 5 as well.

The exercise takes place in a 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. Hints: You can execute certain linux shell commands by prefixing the command with `!`. You can insert Markdown cells and code cells. The first you can use for documenting and explaining your results the second you can use writing 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, with Windows you can use 7zip (https://www.7-zip.org/download.html) to decompress the data.

[ ]:
```
%%script false --no-raise-error

#Download and extract data
!mkdir datasets
%cd datasets
!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
```

```
%cd ..
```

```
Couldn't find program: 'false'
```

The data is now in the three folders `easy_ham`, `hard_ham`, and `spam`.

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

```
'ls' is not recognized as an internal or external command,
operable program or batch file.
```

1.2.1 1. Preprocessing:

1. 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. Further down (in the higher-grade part), you will be asked to filter out the headers and footers.

[ ]:
```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from email import message_from_string
import matplotlib.pyplot as plt
import pandas as pd
import email.policy
import os
```

[ ]:
```
def extract_emails(directories, class_name):
  rows = []
  for directory in directories:
    for file in os.listdir(directory):
      with open(os.path.join(directory,file), encoding='latin-1') as f:
        if f is not None:
          content = f.read()
          rows.append({'email': message_from_string(content, policy=email.
 ↪policy.default), 'content': content, 'class': class_name})
  return pd.DataFrame(rows)

# extract emails and add class information
df_easy_ham = extract_emails(['./datasets/easy_ham'], 'ham')
df_hard_ham = extract_emails(['./datasets/hard_ham'], 'ham')
df_spam = extract_emails(['./datasets/spam'], 'spam')

# join the two dataframes
df_combined = pd.concat([df_easy_ham, df_hard_ham, df_spam])

print('Easy ham:', + len(df_easy_ham))
print('Hard ham:', + len(df_hard_ham))
print('Spam:', + len(df_spam))
df_combined.sample(5)
```

```
Easy ham: 2551
Hard ham: 250
Spam: 501
```

[ ]:
```
                                                  email  \
2111  [Return-Path, Delivered-To, Received, Received…
1029  [Return-Path, Delivered-To, Received, Received…
76    [Return-Path, Delivered-To, Received, Received…
435   [Return-Path, Delivered-To, Received, Received…
128   [Return-Path, Received, Received, Date, From, …

                                                content class
2111  From rssfeeds@jmason.org  Thu Sep 26 16:43:29 …   ham
1029  From exmh-workers-admin@redhat.com  Mon Aug 26…   ham
76    From ilug-admin@linux.ie  Thu Aug 29 17:03:22 …   ham
435   From bmortgage@ig.com.br  Sun Sep 22 23:59:12 …  spam
128   Return-Path: <update@list.theregister.co.uk>\n…   ham
```

2. We don't want to train and test on the same data. Split the spam and the ham datasets in a training set and a test set. (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`)

[ ]:
```
# pre-processing code here
X = df_combined['content']
y = df_combined['class']

# split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
 ↪random_state=0)

# print shapes
print('df_combined:', df_combined.shape)
print('X_train:', X_train.shape)
print('X_test:', X_test.shape)
print('y_train:', y_train.shape)
print('y_test:', y_test.shape)
```

```
df_combined: (3302, 3)
X_train: (2641,)
X_test: (661,)
y_train: (2641,)
y_test: (661,)
```

1.2.2 2. Write a Python program that:

1. Uses four datasets (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`)
2. Trains a Naïve Bayes classifier (e.g. Sklearn) 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` to transform the email texts into vectors. Please note that there are different types of Naïve Bayes Classifier in SKlearn (Documentation here).

Test two of these classifiers that are well suited for this problem

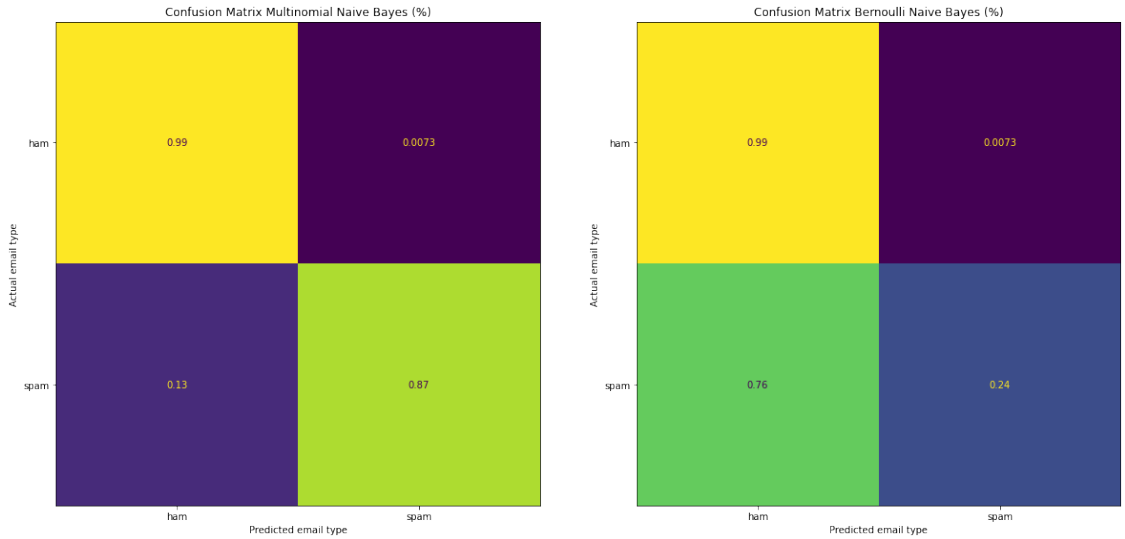
- Multinomial Naive Bayes
- Bernoulli Naive Bayes.

Please inspect the documentation to ensure input to the classifiers is appropriate. Discuss the differences between these two classifiers.

[ ]:
```
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import plot_confusion_matrix, confusion_matrix

def compare_naive_bayes(X_train, X_test, y_train, y_test, cv=None, fit_prior =␣
↪None):
  # instantiate a Count Vectorizer and fit for training data
  if cv is None:
    cv = CountVectorizer()

  X_train_vector = cv.fit_transform(X_train)
  X_test_vector = cv.transform(X_test)

  # fit models using training vector
  multinomial_naive_bayes = MultinomialNB().fit(X_train_vector, y_train)
  bernoulli_naive_bayes = BernoulliNB().fit(X_train_vector, y_train)

  # make predictions on the test data
  y_pred_multinomial = multinomial_naive_bayes.predict(X_test_vector)
  y_pred_bernoulli = bernoulli_naive_bayes.predict(X_test_vector)

  # calculate the true positive and false negatives
  tn1, fp1, fn1, tp1 = confusion_matrix(y_test,y_pred_multinomial).ravel()
  tn2, fp2, fn2, tp2 = confusion_matrix(y_test,y_pred_bernoulli).ravel()

  print ("True positive multinomial: " + str(tn1/(fp1+tn1)))
  print ("True negative multinomial " + str(tp1/(fn1+tp1)))
  print ("True positive bernoulli: " + str(tn2/(fp2+tn2)))
  print ("True negative bernoulli: " + str(tp2/(fn2+tp2)))

  # plot confusion matrices
  _, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(20,10))
  plot_confusion_matrix(multinomial_naive_bayes, X_test_vector, y_test, ax=ax1,␣
↪colorbar=False, normalize='true')
  ax1.set_title("Confusion Matrix Multinomial Naive Bayes (%)")
  ax1.set_xlabel("Predicted email type")
  ax1.set_ylabel("Actual email type")
  plot_confusion_matrix(bernoulli_naive_bayes, X_test_vector, y_test, ax=ax2,␣
↪colorbar=False, normalize='true')
  ax2.set_title('Confusion Matrix Bernoulli Naive Bayes (%)')
```

```
  ax2.set_xlabel("Predicted email type")
  ax2.set_ylabel("Actual email type")

compare_naive_bayes(X_train, X_test, y_train, y_test)
```

```
True positive multinomial: 0.9926605504587156
True negative multinomial 0.8706896551724138
True positive bernoulli: 0.9926605504587156
True negative bernoulli: 0.2413793103448276
```



As the confusion matrices show, the **Multinomial Naive Bayes** model outperforms the **Bernoulli Naive Bayes** model as it suffers from way less misclassifications, and consequently classifies new emails with a higher accuracy. Interestingly enough, both models seem to correctly classify "ham" or non-spam mail with similar accuracy. It's the false negative case, when the model classifies a spam mail as "ham", that separates the two models, with the **Multinomial Naive Bayes** model being more accurate here.

The big difference between **Multinomial Naive Bayes** and **Bernoulli Naive Bayes** is that **Multinomial** takes into account the frequency of the words whilst **Bernoulli** works binary and thus only checks if the word exist or not. This will make **Multinomial** more suited for this task since the a) the dataset is not to large and b) the frequency seems to matter. If the dataset were larger or a different question that is more binary suited then **Bernoulli** be better suited.

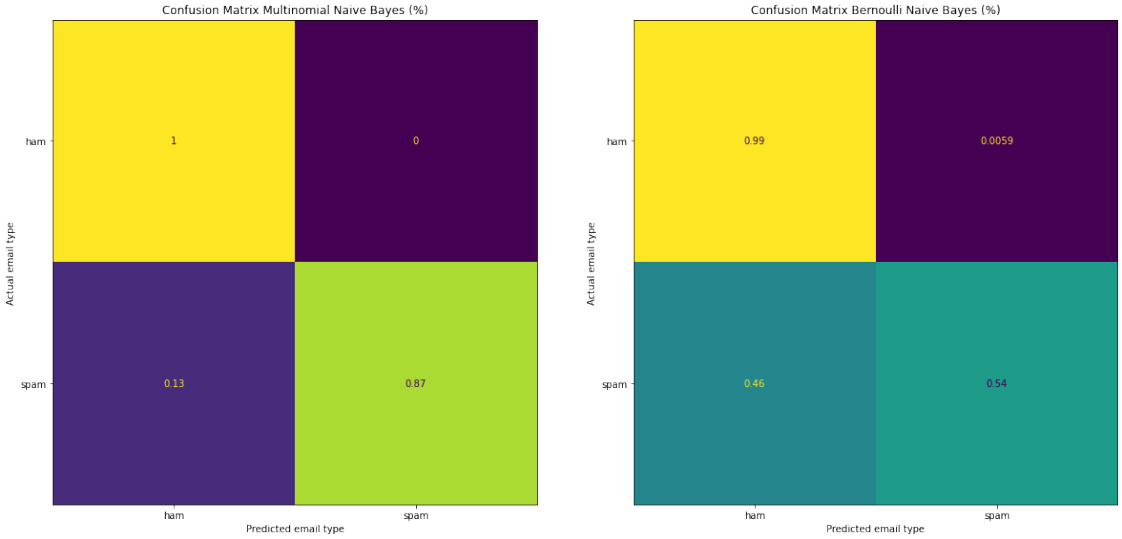
1.2.3 3.Run your program on

- Spam versus easy-ham

[ ]:
```
def spam_vs_easy_ham(cv=None, fit_prior = None):
  spam_vs_easy_ham = pd.concat([df_spam, df_easy_ham])
  X = spam_vs_easy_ham['content']
  y = spam_vs_easy_ham['class']
```

```
  X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
↪random_state=0)
  compare_naive_bayes(X_train, X_test, y_train, y_test, cv=cv)

spam_vs_easy_ham()
```

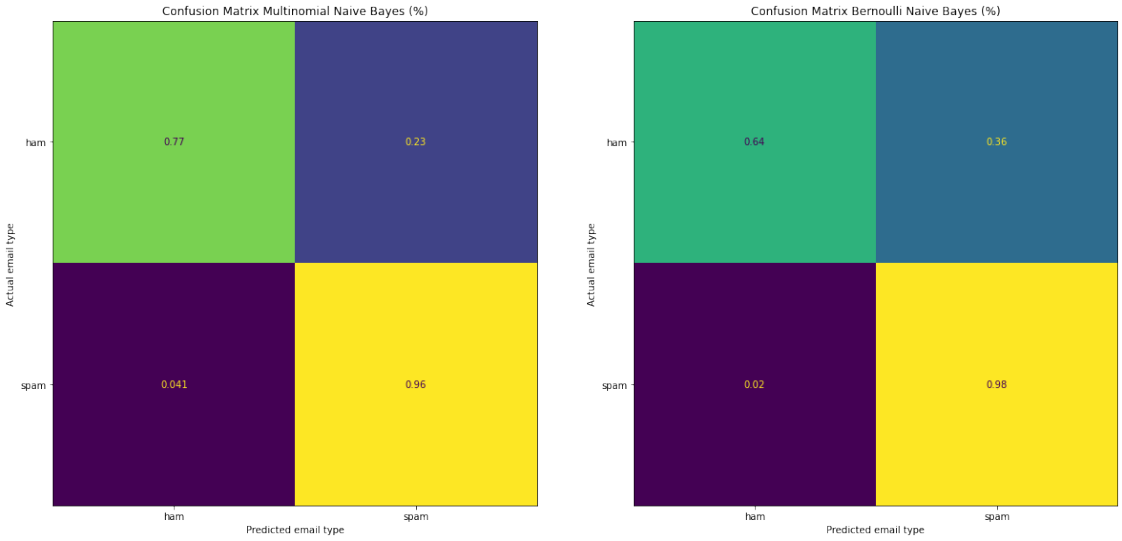
```
True positive multinomial: 1.0
True negative multinomial 0.8712871287128713
True positive bernoulli: 0.9941176470588236
True negative bernoulli: 0.5445544554455446
```



- Spam versus hard-ham.

[ ]:
```
def spam_vs_hard_ham(cv=None, fit_prior = None):
  spam_vs_hard_ham = pd.concat([df_spam, df_hard_ham])
  X = spam_vs_hard_ham['content']
  y = spam_vs_hard_ham['class']
  X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,␣
↪random_state=0)
  compare_naive_bayes(X_train, X_test, y_train, y_test, cv=cv)

spam_vs_hard_ham()
```

```
True positive multinomial: 0.7735849056603774
True negative multinomial 0.9591836734693877
True positive bernoulli: 0.6415094339622641
True negative bernoulli: 0.9795918367346939
```



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

a. Argue why this may be useful. Try finding the words that are too common/uncommon in the dataset.

Discarding common and uninformative words removes what could be considered as noise in the data, which might help improve the accuracy of the two classification models. For instance, looking at the 20 most common words below, one notice that most of these, like `com`, `the`, `to` and `from` are either very uninformative or used in the email headers and thus likely to be irrelevant for deciding if it's spam or ham.

[ ]:
```
def get_top_n_words(corpus, n=None):
    cv = CountVectorizer().fit(corpus)
    word_vector = cv.transform(corpus)
    word_sums = word_vector.sum(axis=0)
    words_freq = [(word, word_sums[0, idx]) for word, idx in cv.vocabulary_.
 ↪items()]
    words_freq = sorted(words_freq, key = lambda word: word[1], reverse=True)
    return words_freq[:n]

top_common_words = get_top_n_words(df_combined['content'])

#top_common_words
```

b. Use the parameters in Sklearn's `CountVectorizer` to filter out these words. Update the program from point 3 and run it on your data and report your results.

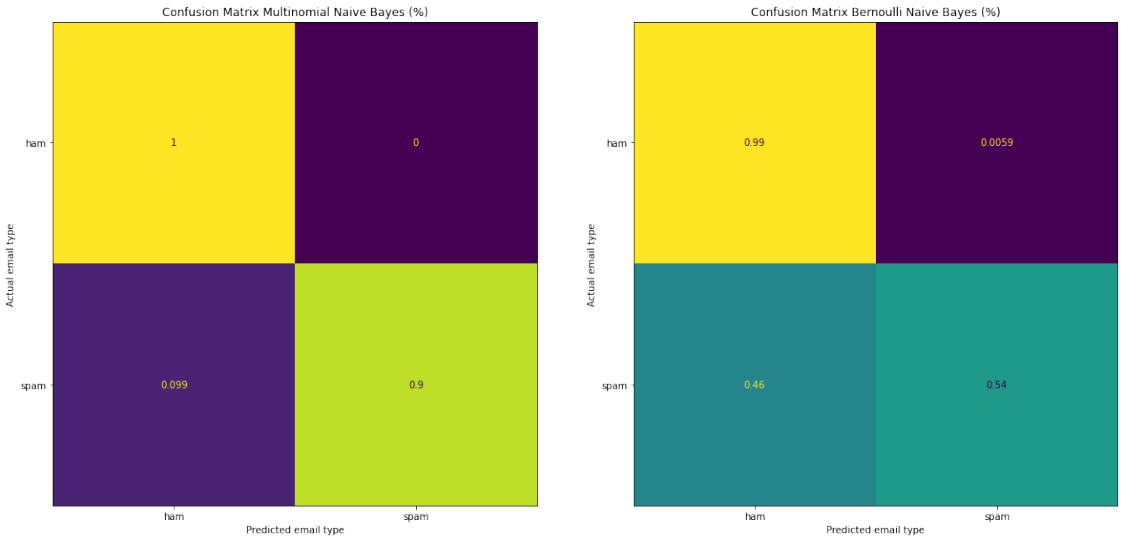
You have two options to do this in Sklearn: either using the words found in part (a) or letting Sklearn do it for you. Argue for your decision-making.

Using a custom word filter

Here one needs to specify the number of words to filter out, in order of word frequency.

[ ]:
```
custom_stop_words = [word[0] for word in top_common_words[:20]]
print(f'filtered out words: {custom_stop_words}')

# generate word count vectorizer with custom in filter
cv_custom_filter = CountVectorizer(stop_words=custom_stop_words)

spam_vs_easy_ham(cv=cv_custom_filter)
```

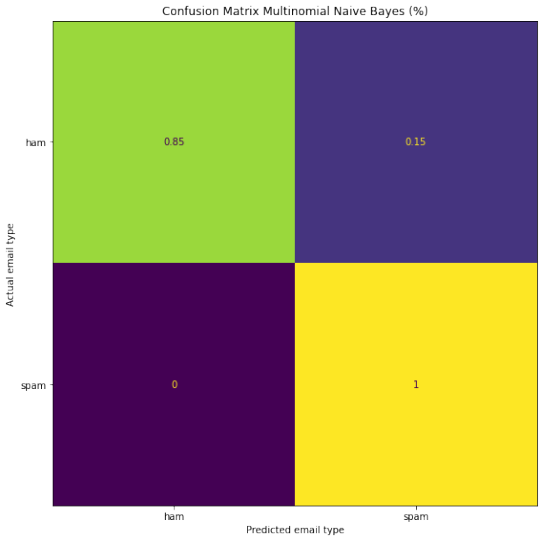
```
filtered out words: ['com', 'the', 'to', 'http', 'from', 'td', '2002', '3d',
'for', 'net', 'font', 'with', 'by', 'width', 'of', 'and', 'localhost', 'id',
'received', 'www']
True positive multinomial: 1.0
True negative multinomial 0.900990099009901
True positive bernoulli: 0.9941176470588236
True negative bernoulli: 0.5445544554455446
```

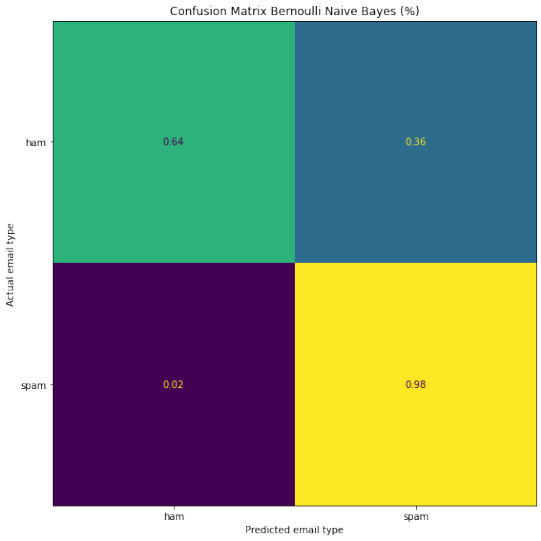


Comparing it with `spam vs hard ham` dataset

[ ]:
```
spam_vs_hard_ham(cv=cv_custom_filter)
```

```
True positive multinomial: 0.8490566037735849
True negative multinomial 1.0
True positive bernoulli: 0.6415094339622641
True negative bernoulli: 0.9795918367346939
```

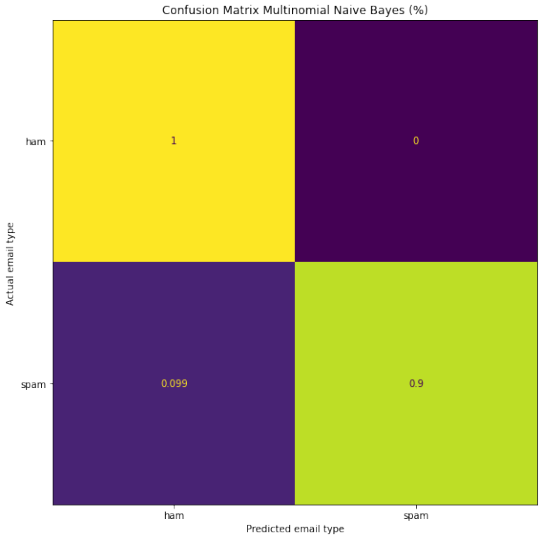


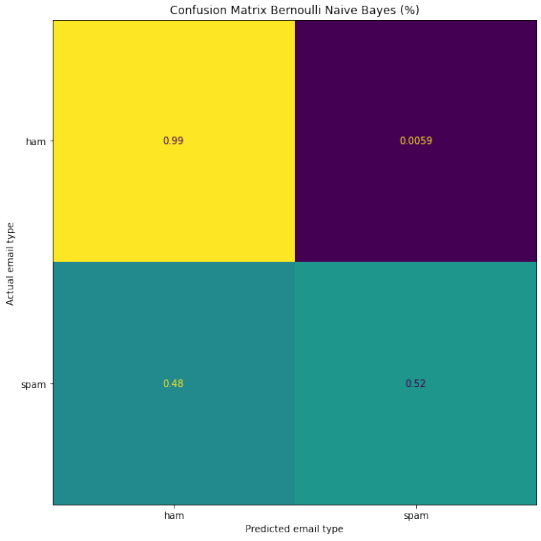


Using SKlearn's built in filtering

[ ]:
```
# generate word count vectorizer configured with built in filter
cv_built_in_filter = CountVectorizer(max_df=0.9, stop_words='english')

spam_vs_easy_ham(cv=cv_built_in_filter)
```

```
True positive multinomial: 1.0
True negative multinomial 0.900990099009901
True positive bernoulli: 0.9941176470588236
True negative bernoulli: 0.5247524752475248
```

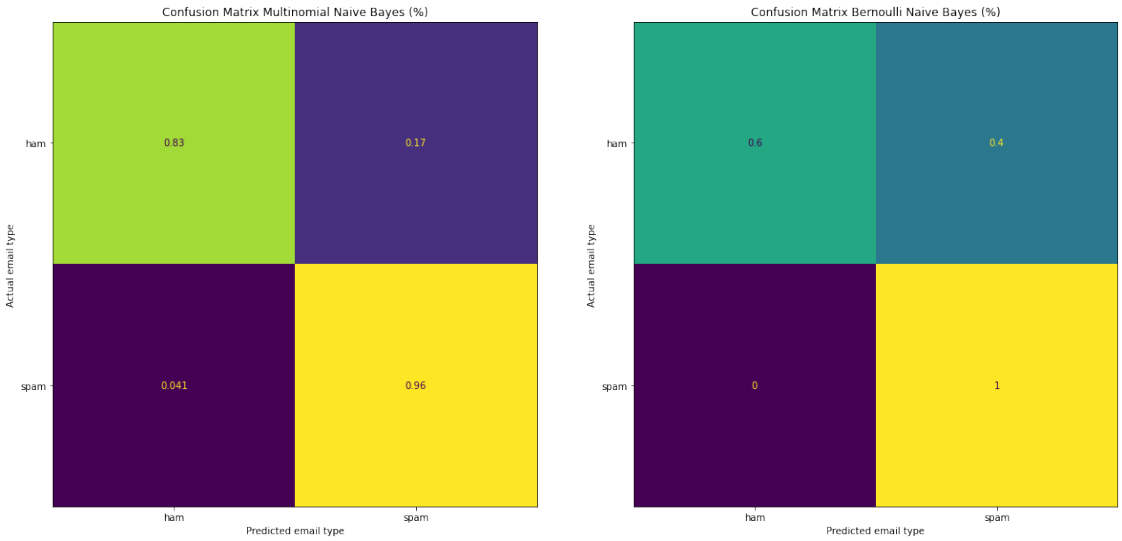




Comparing it with `spam vs hard ham` dataset

[ ]:
```
spam_vs_hard_ham(cv=cv_built_in_filter)
```

```
True positive multinomial: 0.8301886792452831
True negative multinomial 0.9591836734693877
True positive bernoulli: 0.6037735849056604
True negative bernoulli: 1.0
```



As mentioned in the question, one can either let `sklearn` filter out common words using the built-in stop word list for English, or one can manually specify this list. In the documentation, the authors of the `sklern` library mention there are several known issues with using the built in stop word list and that one should consider an alternative. A reason to manually specify the stop words is that this allows for experimenting with different configuration which might fit the particular dataset a lot better. On the other hand this might also increase the risk of overfitting the models with the consequence of poor accuracy when presented with new emails.

We decided to go with the built in method since it's been developed and tested over many years by more experienced engineers, as well as to avoid overfitting (The results suggested that our custom words performed slightly better then the built in method)

1.2.5 5. Eeking out further 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:

[ ]:
```
def extract_payload(email):
  if email.is_multipart():
    for part in email.get_payload():
      return part.get_payload()
```

```
  else:
    return email.get_payload()

df_combined['payload'] = df_combined.email.map(extract_payload)

for email_payload in df_combined.payload.head(3):
  print(email_payload)
```

```
    Date:         Wed, 21 Aug 2002 10:54:46 -0500
    From:         Chris Garrigues <cwg-dated-1030377287.06fa6d@DeepEddy.Com>
    Message-ID:   <1029945287.4797.TMDA@deepeddy.vircio.com>


  | I can't reproduce this error.

For me it is very repeatable… (like every time, without fail).

This is the debug log of the pick happening …

18:19:03 Pick_It {exec pick +inbox -list -lbrace -lbrace -subject ftp -rbrace
-rbrace} {4852-4852 -sequence mercury}
18:19:03 exec pick +inbox -list -lbrace -lbrace -subject ftp -rbrace -rbrace
4852-4852 -sequence mercury
18:19:04 Ftoc_PickMsgs {{1 hit}}
18:19:04 Marking 1 hits
18:19:04 tkerror: syntax error in expression "int …

Note, if I run the pick command by hand …

delta$ pick +inbox -list -lbrace -lbrace -subject ftp -rbrace -rbrace  4852-4852
-sequence mercury
1 hit

That's where the "1 hit" comes from (obviously).  The version of nmh I'm
using is …

delta$ pick -version
pick -- nmh-1.0.4 [compiled on fuchsia.cs.mu.OZ.AU at Sun Mar 17 14:55:56 ICT
2002]

And the relevant part of my .mh_profile …

delta$ mhparam pick
-seq sel -list


Since the pick command works, the sequence (actually, both of them, the
one that's explicit on the command line, from the search popup, and the
```

one that comes from .mh_profile) do get created.

kre

ps: this is still using the version of the code form a day ago, I haven't
been able to reach the cvs repository today (local routing issue I think).

Martin A posted:
Tassos Papadopoulos, the Greek sculptor behind the plan, judged that the
 limestone of Mount Kerdylio, 70 miles east of Salonika and not far from the
 Mount Athos monastic community, was ideal for the patriotic sculpture.

 As well as Alexander's granite features, 240 ft high and 170 ft wide, a
 museum, a restored amphitheatre and car park for admiring crowds are
planned

So is this mountain limestone or granite?
If it's limestone, it'll weather pretty fast.

Man Threatens Explosion In Moscow

Thursday August 22, 2002 1:40 PM
MOSCOW (AP) - Security officers on Thursday seized an unidentified man who
said he was armed with explosives and threatened to blow up his truck in
front of Russia's Federal Security Services headquarters in Moscow, NTV

```
television reported.
The officers seized an automatic rifle the man was carrying, then the man
got out of the truck and was taken into custody, NTV said. No other details
were immediately available.
The man had demanded talks with high government officials, the Interfax and
ITAR-Tass news agencies said. Ekho Moskvy radio reported that he wanted to
talk with Russian President Vladimir Putin.
Police and security forces rushed to the Security Service building, within
blocks of the Kremlin, Red Square and the Bolshoi Ballet, and surrounded the
man, who claimed to have one and a half tons of explosives, the news
agencies said. Negotiations continued for about one and a half hours outside
the building, ITAR-Tass and Interfax reported, citing witnesses.
The man later drove away from the building, under police escort, and drove
to a street near Moscow's Olympic Penta Hotel, where authorities held
further negotiations with him, the Moscow police press service said. The
move appeared to be an attempt by security services to get him to a more
secure location.

------------------------ Yahoo! Groups Sponsor ----------------------~-->
4 DVDs Free +s&p Join Now
http://us.click.yahoo.com/pt6YBB/NXiEAA/mG3HAA/7gSolB/TM
---------------------------------------------------------------------~->

To unsubscribe from this group, send an email to:
forteana-unsubscribe@egroups.com



Your use of Yahoo! Groups is subject to http://docs.yahoo.com/info/terms/
```

1.2.6 - Does the result improve from 3 and 4?

As can be seen in the output above, we managed to remove the headers and footers from the emails. However we ran into type errors while generating the word vector using the CountVectorizer, even though `type(df['payload'])` reported the same types. Unfortunately we could not solve this issue within reasonable time. However, unless the type of mail that was sent (html or normal), or any other header attribute is an indicator of spam or ham we would guess that the results would not improve. On the other hand, removing the header / footer, assuming these provide no insight on whether it's spam or ham, this might help unclutter the dataset, which in turn might improve the accuracy.

1.2.7 - The split of the data set into a training set and a test set can lead to very skewed results. Why is this, and do you have suggestions on remedies?

1.2.8 - What do you expect would happen if your training set were mostly spam messages while your test set were mostly ham messages?

If for example the ratio between ham and spam is 70% versus 30% in the total sample set and the ratio in the training set is 90%/10% and the test set is 50%/50% (this is to illustrate a point and not to discusses whether this is feasible) then they are not properly represented in each set. This will mean that training set will encounter more hams then possible and will be better at detecting them whilst worst at detecting spams. Furthermore is the fit_prior is set to True then it would guess that more emails are ham than it is (more discussion concerning this further down). Although this example is extreme, small changes occurring by chance could will also skew the results. The remedy for this would be utilizing some form of cross-validation on the data set since it would eliminated the random occurring missrepresented training/test sets.

Another thing that could lead to skewed results is the sample distribution of ham and spam. If the ham is higher then spam then it will be better at detecting it then spam. This problem will appear when/if the data gets validated, since it will still perform good on the test data. A remedy for this would probably be to balance out the sample size with the stratify parameter or ensure that the sample size is also representable of the reality (that ham appear more then spam). The second point is maybe not so good although we will not dive into it here since that would be a larger question to answer.

Lastly if the data has not been filtered so that common words are removed this could for example make it think that common words are more frequent amongst the ham. The remedy would for example be to remove the common words before training and testing.

1.3 Re-estimate your classifier using `fit_prior` parameter set to `false`, and answer the following questions:

1.3.1 - What does this parameter mean?

1.3.2 - How does this alter the predictions? Discuss why or why not.

The `fit_prior` argument considers the classes prior probabilities if set to TRUE. Otherwise, it uses assumes uniformity. This is what was explained on the lecture about farmers and librarians. If for example there is a larger number of farmers than librarians the odds of a shy and introverted farmer get picks is higher since the percentage of farmers in the sample is higher, even if the percentage amongst librarians who has this personality trait is higher.

In this case the amount of ham is higher than spam. If prior is set to true, there will a higher chance of predicting ham then spam. From the perspective of being set to true and then switching to false this will result in a higher probability of predicting spam then before. This could result in a lower number of false positives. However it could also lead to a higher number of false negatives.

In our case interestingly enough the data does not change which could indicate that the other factors are good enough indicators that the classes probability according to size does not matter much in this case.

Note: We tested using fit_prior to false but in the code changed it back afterwards

1.4 What to report and how to hand in.

- You will need to clearly report all results in the notebook in a clear and appropriate way, either using plots or code output (f.x. "print statements").
- The notebook must be reproducible, that means, we must be able to use the `Run all` function from the `Runtime` menu and reproduce all your results. **Please check this before handing in.**
- Save the notebook and share a link to the notebook (Press share in upper left corner, and use `Get link` option. **Please make sure to allow all with the link to open and edit.**
- Edits made after submission deadline will be ignored, graders will recover the last saved version before deadline from the revisions history.
- **Please make sure all cells are executed and all the output is clearly readable/visible to anybody opening the notebook.**