0.0.1 NLP Final Project

Syracuse University
IST 736 Text Mining
Fall 2021

0.0.2 Project Introduction

Natural language processing (NLP) is a field of study based on the concept of programming computers to process naturally occurring text in a human like way. Believe it or not, the idea of NLP dates back to the mid 1900s before computers were even invented. But this was just speculation at the time. Nobody was certain if or when it would become a reality. Things have gotten a lot clearer since then. A significant amount of progress has been made in NLP thanks to research efforts by leaders in innovation and the help of technology to make those efforts possible. Think about the mass production of text data that has been a byproduct of the internet, cell phones, audio transcription, and social media. With the next generation push towards internet of things, artificial intelligence, and the metaverse, there is plenty of opportunity for NLP to continue to grow.

NLP is a complicated topic that has many different aspects, but some of the foundational concepts that I have learned about in this course include corpus statistics, part of speech tagging, context free grammar, semantics, discourse and dialogue, and sentiment analysis among others. In this project, I will be applying my knowledge on a spam detection dataset. The dataset consists of a corpus of over 5,000 emails from a company called Enron. The dataset was published in 2006 alongside a paper that used the dataset to develop a spam classifier using naive bayes. More information on the dataset can be found here: http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/readme.txt.

Spam detection is such a standard task in email systems nowadays that we hardly even have to think about it. And part of the reason we do not have to think about it is because the spam detectors are so effective. But how do these spam detectors actually work? Well, what happens is that each email gets processed behind the scenes and is classified as either spam or non spam before being deposited in either the inbox folder or the spam folder. In this project, I will take a

look under the hood and try to build my own spam classifier using machine learning. The goal is to train a model to be able to predict if an email is spam or not spam based on the contents of the email.

Here is what is on the docket for the project. I will start off by importing the necessary Python packages and importing the data. I will then explore and process the email text data, including performing some cleansing operations and feature engineering. From there, I will run a series of experiments. Each experiment will involve training a classifier, but with a slightly different set of features each time. The first experiment will train a classifier with the raw email text without any modifications. This will act as a baseline. In each subsequent classifier, I will add in a modification, such as removing stopwords, adding or removing features, etc. Lastly, I will conclude the project by summarizing and interpreting the results.

0.0.3 Import Packages

I will start off by loading the necessary packages.

[1]:
```
# Import packages
import pandas
import nltk
import re
import os
import random
import matplotlib.pyplot as plt
from nltk.corpus import stopwords

# Scikit learn
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
```

0.0.4 Import Data

The data for this project can be downloaded from Kaggle via the following page: https://www.kaggle.com/wanderfj/enron-spam. There are 6 different corpuses, “enron1”, “en-

ron2", "enron3", "enron4", "enron5", and "enron6". When you download the data from the Kaggle page, it will be downloaded as a zip file called "archive.zip". The "archive.zip" folder contains each of the 6 corpuses. The "archive.zip" folder will need to be unzipped prior to running this code. This can take a while (took about 30 min on my PC). The unzipped folder is just called "archive" without the zip extension.

The following code will read the email data for the spam classification problem from the unzipped "archive" folder. The code assumes that the "archive" folder is located in the current working directory. For anyone trying to reproduce this work in their environment, either make sure that the "archive" folder is located in your current working directory or modify the path accordingly.

Inside the "archive.zip" folder are 6 subfolders for each of the corpuses. Inside each of these corpus subfolders there are 2 subfolders called "spam" and "ham". Inside the "spam" subfolder are individual text files with each text file representing a "spam" email. Inside the "ham" subfolder there are individual text files with each file representing a "ham" email. A "ham" email is a non spam email. Here are the number of emails that are contained within each corpus.

Corpus	Ham	Spam	Total
enron1	3,672	1,500	5,172
enron2	4,361	1,496	5,857
enron3	4,012	1,500	5,512
enron4	1,500	4,449*	6,000
enron5	1,500	3,675	5,175
enron6	1,500	4,500	6,000
total	16,545	17,171	33,716

[2]:
```
# Specify a list of the names of the enron corpuses
corpuses = ["enron1", "enron2", "enron3", "enron4", "enron5", "enron6"]

# Initiate lists for spam and ham email texts
ham_emails = []; spam_emails = []

# Save the pathway to the current working directory
cwd = os.getcwd()

# Iterate through the list of enron corpuses
for enron in corpuses:

    # Specify the file pathway within the "archive directory"
    # Assumes the "archive" folder is in the CWD
    dirPath = os.path.join(cwd, "archive", enron)

    # Change the CWD to the "archive" directory
    os.chdir(dirPath)
```

```
    # Initiate lists for spam and ham email texts
    ham_temp = []; spam_temp = []

    # Process all files in the spam directory
    for file in os.listdir("./spam"):

        # Check if the file extension is ".txt"
        if file.endswith(".txt"):

            # Open file for reading and read entire file into a string
            f = open("./spam/" + file, "r", encoding = "latin-1")
            spam_temp.append(f.read())
            f.close()

    # Process all files in the ham directory
    for file in os.listdir("./ham"):

        # Check if the file extension is ".txt"
        if file.endswith(".txt"):

            # Open file for reading and read entire file into a string
            f = open("./ham/"+file, 'r', encoding="latin-1")
            ham_temp.append(f.read())
            f.close()

    # Append to the main spam and ham lists
    spam_emails += spam_temp
    ham_emails += ham_temp

    # print number emails read
    print("Corpus: ", enron, "\n")
    print("Number of spam files: ", len(spam_temp))
    print("Number of ham files: ", len(ham_temp), "\n")
```

```
Corpus:  enron1

Number of spam files:  1500
Number of ham files:  3672

Corpus:  enron2

Number of spam files:  1496
Number of ham files:  4361

Corpus:  enron3

Number of spam files:  1500
```

```
Number of ham files:  4012

Corpus:  enron4

Number of spam files:  4499
Number of ham files:  1500

Corpus:  enron5

Number of spam files:  3675
Number of ham files:  1500

Corpus:  enron6

Number of spam files:  4500
Number of ham files:  1500
```

Note: I noticed that in the "enron4" corpus, there are supposed to be 4,500 "spam" emails (according to the documentation), but there are only 4,449. There is one email missing. This should not have any effect on the results of the analysis but I thought I would mention it as an FYI.

The previous code generated 2 lists. One list contains the raw text from the "spam" emails and the other list contains the raw text from the "ham" emails. Let's take a look at what some of the emails look like (each email truncated to the first 500 characters to save space).

[3]:
```
# Print out a few examples of ham emails
print(ham_emails[0][0:500], "\n")
print(ham_emails[1][0:500], "\n")
print(ham_emails[16544][0:500], "\n")
print(ham_emails[16543][0:500], "\n")
```

```
Subject: christmas tree farm pictures


Subject: vastar resources , inc .
gary , production from the high island larger block a - 1 # 2 commenced on
saturday at 2 : 00 p . m . at about 6 , 500 gross . carlos expects between 9 ,
500 and
10 , 000 gross for tomorrow . vastar owns 68 % of the gross production .
george x 3 - 6992
- - - - - - - - - - - - - - - - - - - - - - - - - forwarded by george weissman / hou /
ect on 12 / 13 / 99 10 : 16
am - - - - - - - - - - - - - - - - - - - - - - - - -
daren j farmer
12 / 10 / 99 10 : 38 am
```

```
to : car

Subject: fw : re ivanhoe e . s . d
fyi , kim .
- - - - - original message - - - - -
from : frazier , perry
sent : thursday , march 07 , 2002 2 : 25 pm
to : lebeau , randy ; watson , kimberly ; abdmoulaie , mansoor
subject : re : re ivanhoe e . s . d
just a couple of additional thoughts , the cost estimate for esd mods of $ 130 ,
000 will typically be , about 25 % more . the e abdmoulaie , mansoor ; frazier ,
perry
subject : fw : re ivanhoe e . s . d
here is an estimate for the upgrade to the iva

Subject: fw : tw question in amarillo
fyi , kim .
- - - - - original message - - - - -
from : lebeau , randy
sent : tuesday , march 05 , 2002 11 : 07 pm
to : watson , kimberly ; jordan , fred
cc : abdmoulaie , mansoor
subject : re : tw question in amarillo
thanks for the info .
fred , do you know how much it will take to upgrade the esd system ?
- - - - - original message - - - - -
from : watson , kimberly
sent : tuesday , march 05 , 2002 1 : 56 pm
to : lebeau , randy
cc : abdmoulaie , mansoor
s
```

[4]:
```
# Print out a few examples of spam emails
print(spam_emails[0][0:500], "\n")
print(spam_emails[1][0:500], "\n")
print(spam_emails[17169][0:500], "\n")
print(spam_emails[17168][0:500], "\n")
```

```
Subject: dobmeos with hgh my energy level has gone up ! stukm
introducing
doctor - formulated
hgh
human growth hormone - also called hgh
is referred to in medical science as the master hormone . it is very plentiful
when we are young , but near the age of twenty - one our bodies begin to produce
less of it . by the time we are forty nearly everyone is deficient in hgh ,
and at eighty our production has normally diminished at least 90 - 95 % .
```

```
advantages of hgh :
- increased muscle strength
- los

Subject: your prescription is ready . . oxwq s f e
low cost prescription medications
soma , ultram , adipex , vicodin many more
prescribed online and shipped
overnight to your door ! !
one of our us licensed physicians will write an
fda approved prescription for you and ship your
order overnight via a us licensed pharmacy direct
to your doorstep . . . . fast and secure ! !
click here !
no thanks , please take me off your list
ogrg z
lqlokeolnq
lnu

Subject: dear sir , i am interested in it
hi : )
do you need some softwares ? i can give you this link http : / / 6 zk . net .
softyourmeans . com
best site , more than 50 popular programs and all of them are very - very cheap
. you will no need to wait 2 - 3 week for cd delivery - you can download any
program at once you have purchased it ! lt

Subject: bloow in 5 - 10 times the time
learn how to last 5 - 10 times longer in
bed . . .
read here : plods . net
```

Some initial observations from looking at the emails.

(a) Each email consists of a subject line and email body. I think that it could be helpful to distinguish these when training a model because the subject lines of a "spam" email are likely different from those of a "ham" email.

(b) Some of the "ham" emails are forwarded emails and contain multiple emails chained together. The email chain in forwarded emails appear to contain a "to", a "from", a "cc", and timestamps in the email chain.

(c) Anecdotally, I think that "spam" emails seem to have more punctuation in them (for example excessive exclamation points), they tend to have more typos than "ham" emails (mispelled words, words stuck together, non words, etc.), and they can contain inappropriate content (swear words, etc.).

(d) The context of a "spam" email is different than the context of a "ham" email. For example,

"spam" emails are typically propaganda or trying to sell something.

(e) The text is already in lowercase format. The reason I point this out is because it might have been helpful to consider pronouns in the text, but that will not be an option if the capitalization is not present.

In the next bit of code, I will engineer some new features based on these observations such splitting up the subject line and email body, whether the email is a forwarded email or not, as well as a "cleaned" version of the emails. These features will be incorporated later on in the experiments.

0.0.5 Separate the Subject Line and Email Body

I will extract the subject line and the email body from each email. One of the experiments will incorporate this in the spam classifier. In the following code, I will iterate through every email and use regular expression to extract the subject line and the email body. I will first define 2 functions, one for extracting the subject line and one for extracting the email body, then I will use these functions on the emails.

[5]:
```
# Define a function to extract the subject

"""
The "get_subject" function takes in the raw text of an email,
turns it into a list with each component of the list being
a line from the email, checks each line, and the first line
that starts with "subject" or "Subject" (difference being
lowercase or uppercase) will be returned.
"""

def get_subject(text):
    textsplit = text.split("\n")
    for line in textsplit:
        if line.lower().startswith("subject:"):
            subject = re.findall("[sS]ubject: (.*)", line)[0]
            return subject
```

[6]:
```
# Define a function to extract the body

"""
The "get_body" function takes in the raw text of an email
and turns it into a list with each component of the list being
a line from the email. A new list is formed which comtains
all of the lines that are not the subject line. The new
list is then converted into a text string (with the line
breaks back in tact) and returns it.
```

```
"""

def get_body(text):
    textsplit = text.split("\n")
    textlist = []
    linecount = 0
    for line in textsplit:
        if not line.lower().startswith("subject:") and linecount != 0:
            textlist.append(line)
        linecount += 1
    return "\n".join(textlist)
```

[7]:
```
# Extract the subjects from the ham emails
ham_subjects = []
for email in ham_emails:
    ham_subjects.append(get_subject(email))
```

[8]:
```
# Extract the bodies from the ham emails
ham_bodies = []
for email in ham_emails:
    ham_bodies.append(get_body(email))
```

[9]:
```
# Print out a few examples of ham subject lines
print(ham_subjects[1000], "\n")
print(ham_subjects[2000], "\n")
print(ham_subjects[3000], "\n")
print(ham_subjects[4000], "\n")
print(ham_subjects[5000], "\n")
```

```
resume - gloria davis

enron / hpl nom for november 9 , 2000

re : nom / actual flow for april 4 th

invitation to sunday dinner with vince @ 6 . 30 pm

associate prc , martin lin
```

[10]:
```
# Extract the subjects from the ham emails
spam_subjects = []
for email in spam_emails:
    spam_subjects.append(get_subject(email))
```

[11]:
```
# Extract the bodies from the ham emails
spam_bodies = []
for email in spam_emails:
```

```
    spam_bodies.append(get_body(email))
```

[12]:
```
# Print out a few examples of spam subject lines
print(spam_subjects[1000], "\n")
print(spam_subjects[2000], "\n")
print(spam_subjects[3000], "\n")
print(spam_subjects[4000], "\n")
print(spam_subjects[5000], "\n")
```

```
get laid tonight : do you need someone to touch

great news from your bank

from mr . daniel mutade

reply : cute jenna leews soletn potohs

your expiring auto warranty
```

By looking at the subject lines in the "ham" emails and the subject lines in the "spam" emails, I can confirm my observation that the content in these subject lines are different. I think that the subject lines in the "ham" emails will be very domain specific. They will be related specifically to business at Enron or events that are happening at Enron. The "spam" emails will have a very random subject line that has nothing to do with Enron.

0.0.6 Check if an Email is Forwarded

As I made note of earlier, some of the "ham" emails are forwarded emails. You can tell this because they contain delimiters with a bunch of "-" characters and "forwarded" in the same line. I will use this to identify emails that are forwarded and emails that are not forwarded. This operation will be similar to what I did when separating the subject line and email body.

[13]:
```
# Define a function to check if an email is forwarded

"""
The "check_forward" function takes in the raw text of an email
and turns it into a list with each component of the list being
a line from the email. For each line, Two logical tests are
conducted. The first test checks if the line starts with a "-"
character. This is a sign that the line is a delimiter, and
delimiter lines are contained in forwarded emails. The second
```

```
test checks if the line contains the word "forwarded". This
is also a sign that the email is a forwarded email. If both
of these tests come back as true for any of the lines in the
email, then the email is classified as a forwarded email.
"""

def check_forward(text):
    textsplit = text.split("\n")
    isforwarded = False
    for line in textsplit:
        dashtest = line.startswith("-")
        fwdtest = bool(re.search("forwarded", line))
        if dashtest and fwdtest:
            isforwarded = True
    return isforwarded
```

[14]:
```
# Check all the ham emails if they were forwarded
ham_forwarded = []
for email in ham_bodies:
    ham_forwarded.append(check_forward(email))
```

[15]:
```
# How many of the ham emails were forwarded?
print(sum(ham_forwarded), " / ", len(ham_forwarded))
```

```
2629  /  16545
```

[16]:
```
# Check all the spam emails if they were forwarded
spam_forwarded = []
for email in spam_bodies:
    spam_forwarded.append(check_forward(email))
```

[17]:
```
# How many of the ham emails were forwarded?
print(sum(spam_forwarded), " / ", len(spam_forwarded))
```

```
0  /  17170
```

As expected, there are a number of "ham" emails that are forwarded emails (2,629 out of 16,545 emails) but there are not any "spam" emails that are forwarded emails (0 out of 17,170 emails).

0.0.7 Cleaned Version of Emails

Some of the emails (primarily forwarded emails it seems), contain "cc", "to", and "from" lines, along with timestamp lines. The content of these lines mainly contains names, symbols, numbers, etc. I think that this could be more like noise rather than any meaningful information that the classifier can learn from. Thus, my hypothesis is that removing these lines could improve the accuracy of the

spam classifier. That hypothesis will be tested later on in the experiments. Through observation, I also came across some other things that I will clean up. In the following code, I will create cleaned versions of the emails. But first, here is an overview of what I will be cleaning in this step.

(1) Lines that start with "cc". Indicates the names of people who are copied on the email.

(2) Lines that start with "subject". The initial subject line has already been extracted but this will take care of subject lines that are repeated in the email body. This occurs in forwarded emails.

(3) Lines that start with "to". Indicates the names of people who the email is sent to.

(4) Lines that start with "from". Indicates the name of the person who the email was sent from.

(5) Lines that start with a "/" character. Indicates that the line is a timestamp line.

(6) Lines that start with a "-" character. Indicates that the line is a delimiter line in a forwarded email.

(7) Lines that start with "am -". Indicates that the line is a delimiter line in a forwarded email.

(8) Lines that start with a "|" character. Indicates the start of a row in an an embedded table in the email.

(9) Lines that end with a ",". Indicate a greeting line in an email. For example, a lot of emails start with a line such as "Hi (enter name here),".

(10) Lines that start with a number. Indicates that the line is a timestamp line.

[18]:
```
# Define a function to clean an email

"""
The "clean_email" function takes in the raw text of an email
and turns it into a list with each component of the list being
a line from the email. For each line, 10 logical tests are
conducted. If any of the tests for a line come back true,
then that line is removed from the text. Only lines where
none of the tests are true will the line be kept. The function
returns the text with the removed lines.
"""

def clean_email(text):
    textsplit = text.split("\n")
    textlist = []
    for line in textsplit:
        t1 = bool(line.lower().startswith("cc"))
        t2 = bool(line.lower().startswith("subject"))
        t3 = bool(line.lower().startswith("to"))
        t4 = bool(line.lower().startswith("from"))
        t5 = bool(line.lower().startswith("/"))
        t6 = bool(line.lower().startswith("-"))
```

```
        t7 = bool(line.lower().startswith("am -"))
        t8 = bool(line.lower().startswith("|"))
        t9 = bool(line.lower().endswith(","))
        t10 = bool(re.search("^[0-9]", line))
        if sum([t1, t2, t3, t4, t5, t6, t7, t8, t9, t10]) == 0:
            textlist.append(line)
    return "\n".join(textlist)
```

[19]:
```
# Create a cleaned version of the "ham" emails
ham_cleaned = []
for email in ham_bodies:
    ham_cleaned.append(clean_email(email))
```

[20]:
```
# Create a cleaned version of the "spam" emails
spam_cleaned = []
for email in spam_bodies:
    spam_cleaned.append(clean_email(email))
```

[21]:
```
# Show an example of a raw email
print("EXAMPLE OF A RAW EMAIL", "\n")
print(ham_bodies[1][0:1000])
```

```
EXAMPLE OF A RAW EMAIL

gary , production from the high island larger block a - 1 # 2 commenced on
saturday at 2 : 00 p . m . at about 6 , 500 gross . carlos expects between 9 ,
500 and
10 , 000 gross for tomorrow . vastar owns 68 % of the gross production .
george x 3 - 6992
- - - - - - - - - - - - - - - - - - - - - - - - forwarded by george weissman / hou /
ect on 12 / 13 / 99 10 : 16
am - - - - - - - - - - - - - - - - - - - - - - - - -
daren j farmer
12 / 10 / 99 10 : 38 am
to : carlos j rodriguez / hou / ect @ ect
cc : george weissman / hou / ect @ ect , melissa graves / hou / ect @ ect
subject : vastar resources , inc .
carlos ,
please call linda and get everything set up .
i ' m going to estimate 4 , 500 coming up tomorrow , with a 2 , 000 increase
each
following day based on my conversations with bill fischer at bmar .
d .
- - - - - - - - - - - - - - - - - - - - - - - - forwarded by daren j farmer / hou /
ect on 12 / 10 / 99 10 : 34
am - - - - - - - - - - - - - - - - - - - - - - - - -
enron north ameri
```

[22]:
```
# Show the cleaned version of that email
print("CLEANED VERSION OF THE EMAIL", "\n")
print(clean_email(ham_bodies[1])[0:1000])
```

```
CLEANED VERSION OF THE EMAIL

gary , production from the high island larger block a - 1 # 2 commenced on
saturday at 2 : 00 p . m . at about 6 , 500 gross . carlos expects between 9 ,
500 and
george x 3 - 6992
daren j farmer
please call linda and get everything set up .
i ' m going to estimate 4 , 500 coming up tomorrow , with a 2 , 000 increase
each
following day based on my conversations with bill fischer at bmar .
d .
enron north america corp .
the attached appears to be a nomination from vastar resources , inc . for the
high island larger block a - 1 # 2 ( previously , erroneously referred to as the
# 1 well ) . vastar now expects the well to commence production sometime
control so she can provide notification of the turn - on tomorrow . linda ' s
numbers , for the record , are 281 . 584 . 3359 voice and 713 . 312 . 1689 fax .
would you please see that someone contacts linda and advises her how to
submit future nominations via e - mail , fax or voice ? thanks .
george x 3 - 6992
" linda harris " on 12 / 10 / 99
```

This is just one example of cleaning an email that I found depicts the idea well. There are many more examples of these kinds of emails in the dataset. The cleaned email is not perfect, for example the line that contains just "d ." could be removed or the line that contains "george x 3 - 6992" could be removed. But for now, I will leave these. I just wanted to point that out to get the idea across.

0.0.8 Experiments

As mentioned in the introduction, the experiments section will be comprised of a series of experiments which involve training spam classifiers. Each classifier will be trained and evaluated using a 3 fold cross validation. Models will be assessed using 4 metrics - accuracy, precision, recall, and F-measure.

Before we get started with the experiments, I will need to split the data into respective cross validation sets so that all of the experiments are consistent in that they use the same data. Otherwise, it would be like comparing apples to oranges and we want to be comparing apples to apples. A stratified sampling approach will be used to ensure that the ratio of spam to ham emails in each set is the same.

In the following code, I will conduct sampling to get the indices of each respective cross validation set. I will then pack everything that has been done so far (i.e. raw email, cleaned email, subject line, etc.) into a pandas dataframe. Having everything in one place will make it easy to grab the pieces that I need for each experiment and it will also reduce code redundancy.

[23]:
```
# Set the random seed for reproduceability
random.seed(42)

# Initiate empty lists to hold the indices of the cv sets
ham_set1 = []; spam_set1 = []
ham_set2 = []; spam_set2 = []
ham_set3 = []; spam_set3 = []

# Create a list of ham indices (0 through 3671)
ham_indices = list(range(0, len(ham_emails)))

# Create a list of spam indices (0 through 1499)
spam_indices = list(range(0, len(spam_emails)))

# Sample the ham indices
while len(ham_indices) > 0:
    for setnumber in [1, 2, 3]:
        sample = random.sample(list(range(0, len(ham_indices))), 1)[0]
        del ham_indices[sample]
        if setnumber == 1:
            ham_set1.append(sample)
        elif setnumber == 2:
            ham_set2.append(sample)
        else:
            ham_set3.append(sample)

# Sample the spam indices
# Have to remove one due to the missing email
while len(spam_indices) > 1:
    for setnumber in [1, 2, 3]:
        sample = random.sample(list(range(0, len(spam_indices))), 1)[0]
        del spam_indices[sample]
        if setnumber == 1:
            spam_set1.append(sample)
        elif setnumber == 2:
            spam_set2.append(sample)
        else:
            spam_set3.append(sample)
```

[24]:
```
# Save a list of the column names for the dataframe
columns = ["CVSet", "RawEmail", "RawEmailSubject",
```

```
"RawEmailBody", "IsForwarded", "CleanedEmailBody", "Label"]
```

[25]:
```
# Pack together set 1 ham emails
hamemaildf1 = pandas.DataFrame(list(zip(
    [1 for i in range(0, len(ham_set1))],
    [ham_emails[i] for i in ham_set1],
    [ham_subjects[i] for i in ham_set1],
    [ham_bodies[i] for i in ham_set1],
    [ham_forwarded[i] for i in ham_set1],
    [ham_cleaned[i] for i in ham_set1],
    ["ham" for i in range(0, len(ham_set1))])),
    columns = columns)
```

[26]:
```
# Pack together set 2 ham emails
hamemaildf2 = pandas.DataFrame(list(zip(
    [2 for i in range(0, len(ham_set2))],
    [ham_emails[i] for i in ham_set2],
    [ham_subjects[i] for i in ham_set2],
    [ham_bodies[i] for i in ham_set2],
    [ham_forwarded[i] for i in ham_set2],
    [ham_cleaned[i] for i in ham_set2],
    ["ham" for i in range(0, len(ham_set2))])),
    columns = columns)
```

[27]:
```
# Pack together set 3 ham emails
hamemaildf3 = pandas.DataFrame(list(zip(
    [3 for i in range(0, len(ham_set3))],
    [ham_emails[i] for i in ham_set3],
    [ham_subjects[i] for i in ham_set3],
    [ham_bodies[i] for i in ham_set3],
    [ham_forwarded[i] for i in ham_set3],
    [ham_cleaned[i] for i in ham_set3],
    ["ham" for i in range(0, len(ham_set3))])),
    columns = columns)
```

[28]:
```
# Pack together set 1 spam emails
spamemaildf1 = pandas.DataFrame(list(zip(
    [1 for i in range(0, len(spam_set1))],
    [spam_emails[i] for i in spam_set1],
    [spam_subjects[i] for i in spam_set1],
    [spam_bodies[i] for i in spam_set1],
    [spam_forwarded[i] for i in spam_set1],
    [spam_cleaned[i] for i in spam_set1],
    ["spam" for i in range(0, len(spam_set1))])),
    columns = columns)
```

[29]:
```
# Pack together set 2 spam emails
spamemaildf2 = pandas.DataFrame(list(zip(
    [2 for i in range(0, len(spam_set2))],
    [spam_emails[i] for i in spam_set2],
    [spam_subjects[i] for i in spam_set2],
    [spam_bodies[i] for i in spam_set2],
    [spam_forwarded[i] for i in spam_set2],
    [spam_cleaned[i] for i in spam_set2],
    ["spam" for i in range(0, len(spam_set2))])),
    columns = columns)
```

[30]:
```
# Pack together set 3 spam emails
spamemaildf3 = pandas.DataFrame(list(zip(
    [3 for i in range(0, len(spam_set3))],
    [spam_emails[i] for i in spam_set3],
    [spam_subjects[i] for i in spam_set3],
    [spam_bodies[i] for i in spam_set3],
    [spam_forwarded[i] for i in spam_set3],
    [spam_cleaned[i] for i in spam_set3],
    ["spam" for i in range(0, len(spam_set3))])),
    columns = columns)
```

[31]:
```
# Create one dataframe with everything
emaildf = pandas.concat([
    hamemaildf1,
    hamemaildf2,
    hamemaildf3,
    spamemaildf1,
    spamemaildf2,
    spamemaildf3
])
```

0.0.9 Experiment 1: Baseline Experiment

The baseline experiment will simply use the raw text from the emails to train a classifer. The text will be vectorized using the scikit learn CountVectorizer function with all of the default parameters (other than the encoding which is set to “latin-1”).

[32]:
```
# Initiate a vectorizer
vectorizer = CountVectorizer(encoding = "latin-1")
```

[34]:
```
# Iterate through fold 1, 2, and 3
for cv in [1, 2, 3]:
```

```
    # When cv = 1, fold 1 & and 2 are used for training and fold 3 is used for␣
↪testing
    if cv == 1:
        X_train = emaildf[emaildf.CVSet != 3].RawEmail
        X_test = emaildf[emaildf.CVSet == 3].RawEmail
        y_train = emaildf[emaildf.CVSet != 3].Label
        y_test = emaildf[emaildf.CVSet == 3].Label

    # When cv = 2, fold 2 & 3 are used for training and fold 1 is used for␣
↪testing
    elif cv == 2:
        X_train = emaildf[emaildf.CVSet != 1].RawEmail
        X_test = emaildf[emaildf.CVSet == 1].RawEmail
        y_train = emaildf[emaildf.CVSet != 1].Label
        y_test = emaildf[emaildf.CVSet == 1].Label

    # When cv = 3, fold 3 & 1 are used for training and fold 2 is used for␣
↪testing
    else:
        X_train = emaildf[emaildf.CVSet != 2].RawEmail
        X_test = emaildf[emaildf.CVSet == 2].RawEmail
        y_train = emaildf[emaildf.CVSet != 2].Label
        y_test = emaildf[emaildf.CVSet == 2].Label

    # Vectorize the train data
    X_train_vec = vectorizer.fit_transform(X_train)

    # Vectorize the test data
    X_test_vec = vectorizer.transform(X_test)

    # Initiate a multinomial naive bayes model
    model = MultinomialNB()

    # Fit the model on the train data
    model.fit(X_train_vec, y_train)

    # Make predictions based on the test data
    y_pred = model.predict(X_test_vec)

    # Print out the classification report for the current fold
    print("Fold ", cv, " Classification Report:", "\n")
    print(classification_report(y_test, y_pred, digits = 4), "\n")

    # Print out a confusion matrix for the current fold
    ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels =␣
↪["ham", "spam"]).plot()
```

```
    # Create a dataframe with ham and spam log probabilities by word
    feature_log_ratio = pandas.DataFrame(list(zip(
    model.feature_log_prob_[0],
    model.feature_log_prob_[1],
    vectorizer.get_feature_names_out())),
    columns = ["ham", "spam", "token"])

    # Add a column for the ratio
    feature_log_ratio["ratio"] = feature_log_ratio["ham"] /␣
↪feature_log_ratio["spam"]

    # Sort the dataframe by ratio in descending order
    feature_log_ratio.sort_values(by = "ratio", ascending = False, inplace =␣
↪True)

    # Print out the top 100
    print("Fold ", cv, " Top 100 Most Informative Features for Spam Emails",␣
↪"\n")
    print(list(feature_log_ratio.head(100).token), "\n")

    # Print out the bottom 100
    print("Fold ", cv, " Top 100 Most Informative Features for Ham Emails",␣
↪"\n")
    print(list(feature_log_ratio.tail(100).token), "\n")
```

```
Fold  1  Classification Report:

              precision    recall  f1-score   support

         ham     0.9866    0.9897    0.9881      5515
        spam     0.9900    0.9871    0.9885      5723

    accuracy                         0.9883     11238
   macro avg     0.9883    0.9884    0.9883     11238
weighted avg     0.9883    0.9883    0.9883     11238


Fold  1  Top 100 Most Informative Features for Spam Emails

['pills', 'td', 'nbsp', 'viagra', 'width', 'computron', 'cialis', 'href',
'meds', 'spam', 'src', 'xp', 'paliourg', 'voip', 'macromedia', 'photoshop',
'2005', 'adobe', '2004', 'div', 'drugs', 'bgcolor', 'ooking', 'wiil',
'pharmacy', 'prescription', 'br', 'xanax', 'height', 'logos', 'php', 'valium',
'lottery', 'penis', 'softwares', 'gr', 'pill', 'sex', 'eogi', '0310041',
'stationery', 'moopid', 'paypal', 'demokritos', 'iit', 'materia', 'dose',
'wysak', 'img', 'htmlimg', 'yap', 'ur', 'darial', 'fontfont', 'rolex', 'que',
```

```
'oniine', 'serial', 'ail', 'internationa', 'gra', 'resuits', 'vicodin', 'vnbl',
'1618', 'illustrator', 'couid', 'technoiogy', 'piease', 'newsietter',
'projecthoneypot', 'otcbb', 'oi', 'rnd', 'jebel', 'rocket', 'robotics', 'oniy',
'rfid', 'andmanyother', 'emerson', 'valign', 'lauraan', 'sofftwaares',
'profiled', 'nomad', 'pubiisher', 'squirrelmail', 'colspan', 'geec', 'potentia',
'erections', 'cellpadding', 'font', 'shouid', 'customerservice', 'um', 'tongue',
'wel', 'oo']
```

Fold 1 Top 100 Most Informative Features for Ham Emails

```
['shanbhogue', 'dbcaps', 'farmer', 'valero', 'unify', 'houston', 'neal', 'pefs',
'lamphier', 'dabhol', 'memo', 'nomination', 'lavorato', 'epmi', 'papayoti',
'rita', 'curves', 'clem', 'katy', 'hourahead', '2000', 'anjam', 'sherlyn',
'pat', 'ravi', 'cec', 'gcs', 'hplno', '853', 'midcon', 'mseb', 'vance', 'hplo',
'fyi', 'skilling', 'gco', 'herod', 'cera', 'weissman', 'wynne', 'luong',
'wellhead', 'nominations', 'calpine', 'pm', 'entex', 'txu', 'iferc',
'schedules', 'melissa', 'vasant', 'dpc', '2001', 'aep', 'lannou', 'zimin',
'equistar', 'enrononline', 'buyback', 'nom', 'ebs', 'cotten', 'tenaska',
'eastrans', 'ferc', 'eol', 'hplc', 'hsc', 'gibner', 'counterparty', 'crenshaw',
'volumes', 'fastow', 'actuals', 'clynes', 'aimee', 'cc', 'noms', 'teco', '713',
'louise', 'forwarded', 'chokshi', 'stinson', 'enronxgate', 'meter', 'pec',
'vince', 'dynegy', 'daren', 'ees', 'ena', 'sitara', 'xls', 'hou', 'hpl',
'mmbtu', 'kaminski', 'ect', 'enron']
```

Fold 2 Classification Report:

	precision	recall	f1-score	support
ham	0.9874	0.9909	0.9891	5515
spam	0.9912	0.9878	0.9895	5723
accuracy			0.9893	11238
macro avg	0.9893	0.9894	0.9893	11238
weighted avg	0.9893	0.9893	0.9893	11238

Fold 2 Top 100 Most Informative Features for Spam Emails

```
['nbsp', 'td', 'pills', 'viagra', 'width', 'computron', 'cialis', 'meds',
'href', 'src', 'paliourg', 'voip', 'photoshop', 'height', 'macromedia', 'adobe',
'font', 'xp', '0310041', 'bgcolor', 'ooking', 'wiil', 'oem', 'pharmacy', '2004',
'spam', 'sex', 'penis', 'php', 'xanax', 'valium', '000000', 'prescription',
'gr', 'paypal', 'rolex', 'br', 'div', 'wysak', 'tiscali', 'img', 'materia',
'htmlimg', 'pill', 'rnd', 'demokritos', 'iit', '2005', 'wifi', 'drugs', 'yap',
'stationery', 'cf', 'ail', 'cum', 'dose', 'darial', 'eogi', 'para',
'technoiogy', 'resuits', 'oniine', 'vicodin', 'colspan', 'internationa',
'foresee', 'lottery', 'piease', 'couid', 'vnbl', '1618', 'gra', 'newsietter',
'ur', 'illustrator', 'jebel', 'otcbb', 'fontfont', 'andmanyother', 'emerson',
```

'serial', 'projecthoneypot', 'valign', 'erections', 'softwares', 'pubiisher',
'lauraan', 'shouid', 'oniy', 'potentia', 'robotics', 'tr', 'rfid', 'oi',
'regalis', 'technoiogies', 'phentermine', 'oo', 'vi', 'geec']

Fold 2 Top 100 Most Informative Features for Ham Emails

['houston', 'dabhol', 'jackie', 'pefs', 'redeliveries', 'withers', 'pops',
'reinhardt', 'ami', 'wellhead', 'fyi', 'calger', 'ravi', 'outage', 'vance',
'853', 'cornhusker', 'gcs', 'cec', 'hourahead', 'clem', 'sherlyn', '2000',
'rita', 'curves', 'hplno', 'allocated', 'ljm', 'anjam', 'lavorato', 'skilling',
'dbcaps', 'mseb', 'luong', 'herod', 'hplo', 'cera', 'melissa', 'iferc',
'vasant', 'wynne', 'epmi', 'enw', 'equistar', 'calpine', 'zimin', 'dpc',
'buyback', 'txu', 'midcon', 'nom', 'gco', '2001', 'nominations', 'entex', 'aep',
'lannou', 'ebs', 'pm', 'hanks', 'cotten', 'volumes', 'eol', 'tenaska',
'crenshaw', 'gibner', 'enrononline', 'ferc', 'eastrans', 'stinson', 'clynes',
'hplc', '713', 'counterparty', 'actuals', 'hsc', 'noms', 'fastow', 'aimee',
'cc', 'forwarded', 'teco', 'meter', 'chokshi', 'louise', 'pec', 'enronxgate',
'dynegy', 'vince', 'ees', 'daren', 'ena', 'xls', 'sitara', 'hpl', 'mmbtu',
'hou', 'kaminski', 'ect', 'enron']

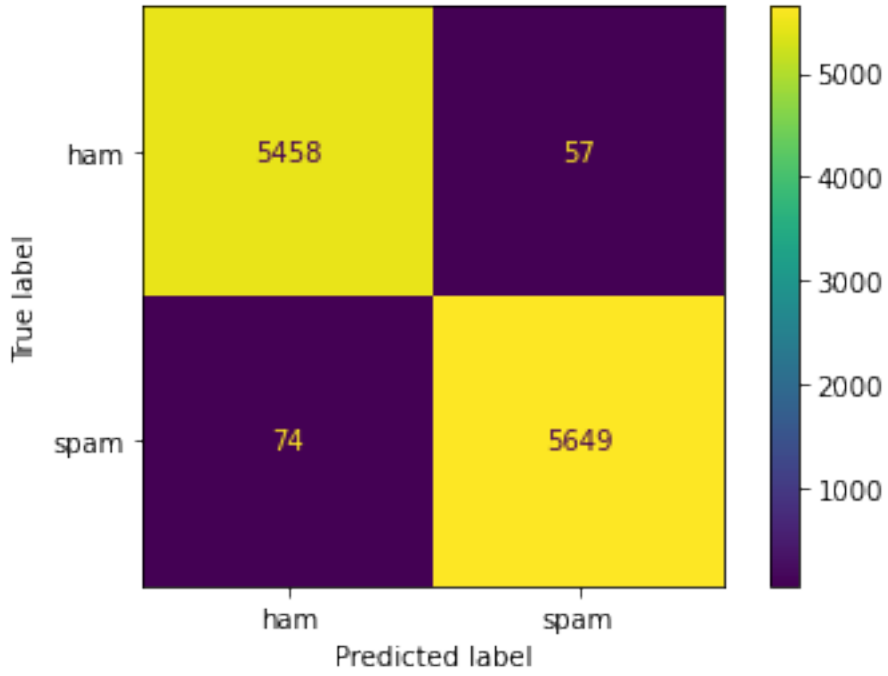
Fold 3 Classification Report:

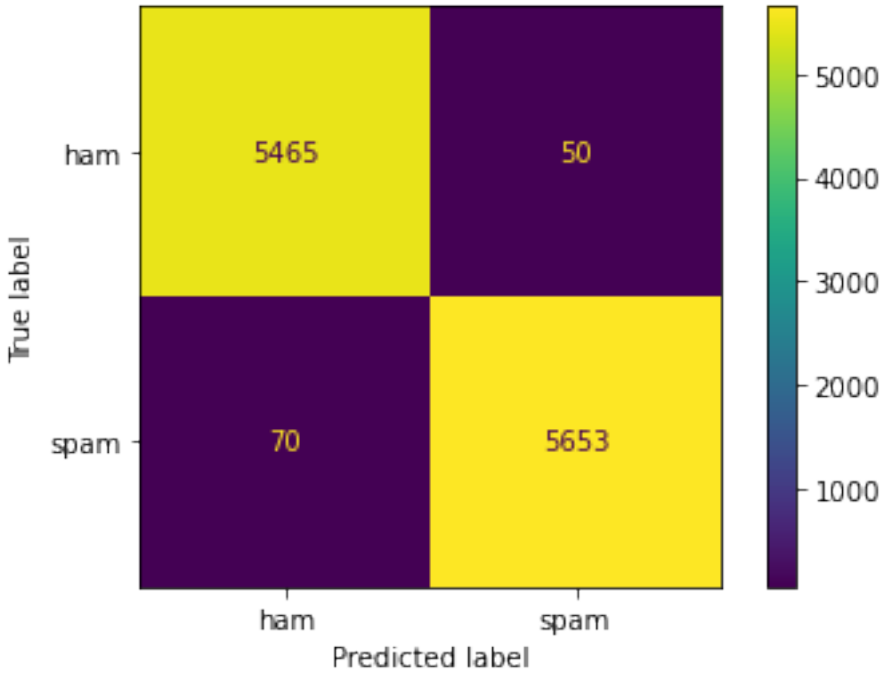
	precision	recall	f1-score	support
ham	0.9903	0.9860	0.9882	5515
spam	0.9866	0.9907	0.9887	5723
accuracy			0.9884	11238
macro avg	0.9885	0.9884	0.9884	11238
weighted avg	0.9884	0.9884	0.9884	11238

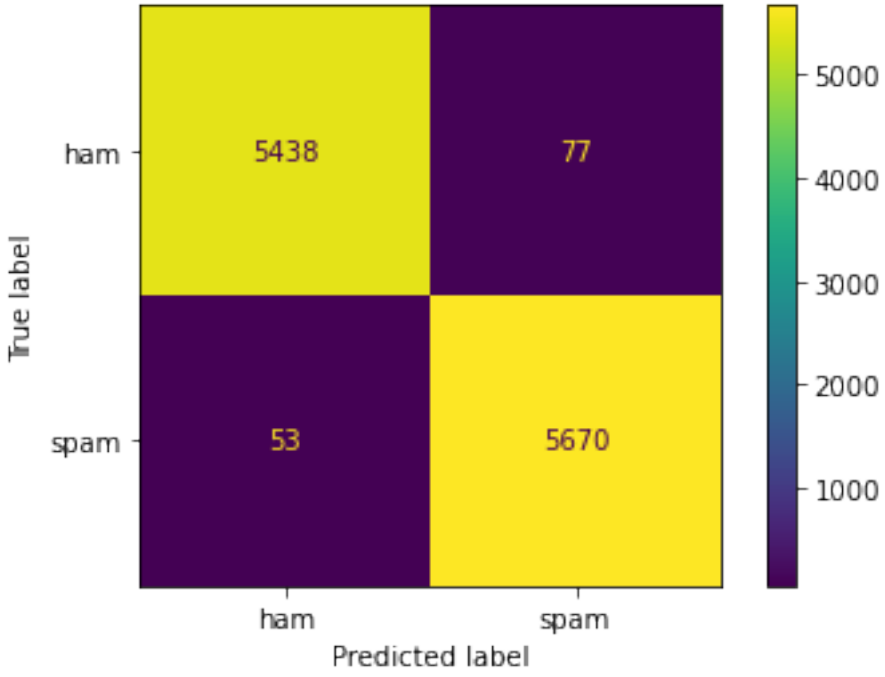
Fold 3 Top 100 Most Informative Features for Spam Emails

['pills', 'td', 'viagra', 'computron', 'width', 'nbsp', 'cialis', 'xp',
'height', 'meds', 'href', 'php', 'paliourg', 'src', 'prescription', 'photoshop',
'macromedia', 'voip', 'adobe', 'bgcolor', '0310041', '2004', 'sex', 'kin',
'spam', 'gr', 'penis', 'xanax', 'eogi', 'wiil', 'pharmacy', 'ooking', 'valium',
'drugs', 'pill', 'demokritos', 'iit', 'rolex', 'wysak', 'rnd', 'htmlimg',
'paypal', 'vicodin', 'div', 'stationery', 'wifi', '2005', 'emerson', 'materia',
'oniine', 'img', '1618', 'ur', 'dose', '8859', 'ail', 'yap', 'tiscali', 'jebel',
'discreet', 'fontfont', 'gra', 'serial', 'oi', 'darial', 'robotics', 'vnbl',
'resuits', 'lottery', 'ebay', 'phentermine', 'projecthoneypot', '1226030',
'internationa', 'technoiogy', 'alt', 'thousand', 'illustrator', 'speedway',
'aopen', '4176', 'customerservice', 'viewsonic', '1933', 'iomega', 'vinoble',
'8834464', '8834454', 'piease', 'targus', 'intellinet', 'potentia',
'andmanyother', 'erections', 'colspan', 'levitra', 'couid', 'rfid', 'corel',
'ambien']

```
Fold  3  Top 100 Most Informative Features for Ham Emails

['tejas', 'shanbhogue', 'shirley', 'redeliveries', 'counterparties',
'hourahead', 'dbcaps', 'epmi', 'cornhusker', 'withers', 'papayoti',
'responsibilities', 'cera', 'farmer', 'reinhardt', 'vance', 'calger',
'lavorato', 'lamphier', 'ravi', '2000', 'ljm', 'sherlyn', 'fyi', 'nomination',
'valero', 'pefs', 'gcs', 'clem', 'liquidity', 'cec', 'hplno', 'anjam',
'skilling', 'curves', 'hplo', '853', 'herod', 'vasant', 'wynne', 'iferc',
'luong', 'wellhead', 'gco', 'entex', 'equistar', 'txu', 'ebs', 'calpine',
'nominations', 'pat', '2001', 'aep', 'zimin', 'pm', 'midcon', 'lannou',
'volumes', 'buyback', 'tenaska', 'ferc', 'eol', 'eastrans', 'cotten', 'gibner',
'713', 'enrononline', 'hplc', 'stinson', 'actuals', 'counterparty', 'hsc',
'crenshaw', 'clynes', 'nom', 'noms', 'aimee', 'teco', 'louise', 'fastow', 'cc',
'melissa', 'chokshi', 'enronxgate', 'forwarded', 'pec', 'meter', 'ees', 'vince',
'dynegy', 'ena', 'daren', 'xls', 'sitara', 'hou', 'hpl', 'mmbtu', 'kaminski',
'ect', 'enron']
```




5465
50
70
5653
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000


5438
77
53
5670
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000

0.0.10 Experiment 1 Summary Table

Average accuracy = 98.87%

Class	Measure	Average
ham	precision	98.81%
spam	precision	98.93%
ham	recall	98.89%
spam	recall	98.85%
ham	f-measure	98.85%
spam	f-measure	98.89%

I was a little bit surprised at first to see that the accuracy was nearly 99% from using just the raw text from the emails. But by looking at the most informative words, it is evident that the model was able to pick up on the differences. With this model, there are only around 127 misclassified emails on average per 11,238 emails. Type I errors (false positive: thought it was a spam email but it was actually a ham email) make up for about 61 of those misclassifications. Type II errors (false negative: thought it was a ham email but it was actually a spam email) make up for about the other 66 misclassifications. So, the number of Type I errors and Type II errors are roughly the same. Also, the precision, recall, and f-measure scores are all close to each other. This means that the model is not swinging too heavily in one direction which is a good thing because it means that it is unbiased. But I would still to see if there is room for improvement.

0.0.11 Experiment 2: Using the Cleaned Emails Instead of Raw Emails

In experiment 2, I will be training the naive bayes classifier using the cleaned emails instead of the raw emails to see if the results are improved. The hypothesis is that the cleaned emails have less noise in them which could reduce the confusion in the model. If you need a refresher on what the cleaned emails are please refer back to that section.

[38]:
```
# Initiate a vectorizer
vectorizer = CountVectorizer(encoding = "latin-1")
```

[39]:
```
# Iterate through fold 1, 2, and 3
for cv in [1, 2, 3]:

    # When cv = 1, fold 1 & and 2 are used for training and fold 3 is used for␣
 ↪testing
    if cv == 1:
        X_train = emaildf[emaildf.CVSet != 3].CleanedEmailBody
        X_test = emaildf[emaildf.CVSet == 3].CleanedEmailBody
```

```
        y_train = emaildf[emaildf.CVSet != 3].Label
        y_test = emaildf[emaildf.CVSet == 3].Label

    # When cv = 2, fold 2 & 3 are used for training and fold 1 is used for␣
↪testing
    elif cv == 2:
        X_train = emaildf[emaildf.CVSet != 1].CleanedEmailBody
        X_test = emaildf[emaildf.CVSet == 1].CleanedEmailBody
        y_train = emaildf[emaildf.CVSet != 1].Label
        y_test = emaildf[emaildf.CVSet == 1].Label

    # When cv = 3, fold 3 & 1 are used for training and fold 2 is used for␣
↪testing
    else:
        X_train = emaildf[emaildf.CVSet != 2].CleanedEmailBody
        X_test = emaildf[emaildf.CVSet == 2].CleanedEmailBody
        y_train = emaildf[emaildf.CVSet != 2].Label
        y_test = emaildf[emaildf.CVSet == 2].Label

    # Vectorize the train data
    X_train_vec = vectorizer.fit_transform(X_train)

    # Vectorize the test data
    X_test_vec = vectorizer.transform(X_test)

    # Initiate a multinomial naive bayes model
    model = MultinomialNB()

    # Fit the model on the train data
    model.fit(X_train_vec, y_train)

    # Make predictions based on the test data
    y_pred = model.predict(X_test_vec)

    # Print out the classification report for the current fold
    print("Fold ", cv, " Classification Report:", "\n")
    print(classification_report(y_test, y_pred, digits = 4), "\n")

    # Print out a confusion matrix for the current fold
    ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels =␣
↪["ham", "spam"]).plot()

    # Create a dataframe with ham and spam log probabilities by word
    feature_log_ratio = pandas.DataFrame(list(zip(
    model.feature_log_prob_[0],
    model.feature_log_prob_[1],
    vectorizer.get_feature_names_out())),
```

```
    columns = ["ham", "spam", "token"])

    # Add a column for the ratio
    feature_log_ratio["ratio"] = feature_log_ratio["ham"] /␣
↪feature_log_ratio["spam"]

    # Sort the dataframe by ratio in descending order
    feature_log_ratio.sort_values(by = "ratio", ascending = False, inplace =␣
↪True)

    # Print out the top 100
    print("Fold ", cv, " Top 100 Most Informative Features for Spam Emails",␣
↪"\n")
    print(list(feature_log_ratio.head(100).token), "\n")

    # Print out the bottom 100
    print("Fold ", cv, " Top 100 Most Informative Features for Ham Emails",␣
↪"\n")
    print(list(feature_log_ratio.tail(100).token), "\n")
```

```
Fold  1  Classification Report:

              precision    recall  f1-score   support

         ham     0.9856    0.9837    0.9847      5515
        spam     0.9843    0.9862    0.9852      5723

    accuracy                         0.9850     11238
   macro avg     0.9850    0.9849    0.9850     11238
weighted avg     0.9850    0.9850    0.9850     11238


Fold  1  Top 100 Most Informative Features for Spam Emails

['td', 'nbsp', 'pills', 'width', 'viagra', 'computron', 'href', 'cialis', 'src',
'voip', 'spam', 'macromedia', 'meds', 'font', 'photoshop', 'xp', 'wi', 'adobe',
'bgcolor', 'br', 'ooking', 'wiil', '2005', 'height', 'div', 'drugs', '2004',
'php', '0310041', 'paliourg', 'eogi', 'materia', 'lottery', 'moopid',
'prescription', 'penis', 'htmlimg', 'logos', '000000', 'pharmacy', 'paypal',
'softwares', 'yap', 'dose', 'img', 'xanax', 'biz', 'wysak', 'valium',
'stationery', 'ur', 'resuits', 'que', 'darial', 'internationa', 'vnbl', 'gr',
'gif', 'ail', 'sex', 'pill', 'serial', 'newsietter', 'couid', 'piease',
'demokritos', 'illustrator', '1618', 'iit', 'rnd', 'rocket', 'technoiogy',
'andmanyother', 'oniy', 'sofftwaares', 'valign', 'lauraan', 'projecthoneypot',
'rfid', 'robotics', 'profiled', 'pubiisher', 'squirrelmail', 'colspan',
'cellpadding', 'emerson', 'otcbb', 'customerservice', 'potentia', '4176',
'nomad', 'jebel', 'viewsonic', 'thousand', 'targus', 'aopen', 'intellinet',
```

'iomega', 'oi', '161']

Fold 1 Top 100 Most Informative Features for Ham Emails

['liz', 'gcs', 'kcs', 'wynne', '4179', 'pm', 'metrics', 'pg', 'hourahead',
'meters', 'pops', 'hplno', 'shanbhogue', 'hplo', 'allocated', 'kevin', 'pathed',
'variances', 'withers', 'nominations', 'ravi', 'edu', 'liquidity', 'vance',
'clem', 'clynes', 'midcon', 'pjm', 'responsibilities', 'unify', '2001', 'enw',
'anjam', 'nom', 'counterparties', '77002', 'reliantenergy', 'ljm', 'curves',
'txu', '6353', 'memo', 'entex', 'katy', 'gco', 'equistar', 'lsk',
'redeliveries', 'aep', 'eastrans', 'noms', 'dabhol', 'enronxgate', 'zimin',
'epmi', 'dbcaps', 'vasant', 'wellhead', 'tenaska', 'cera', 'houston',
'skilling', 'gibner', 'crenshaw', 'aimee', 'mseb', 'buyback', 'louise',
'schedules', 'iferc', 'eol', 'hou', 'enrononline', 'dpc', 'hplc', 'ferc', 'ebs',
'stinson', '853', 'hsc', 'fyi', 'pec', '713', 'counterparty', 'fastow', 'meter',
'volumes', 'teco', 'vince', 'xls', 'ees', 'hpl', 'dynegy', 'sitara', 'ena',
'kaminski', 'daren', 'mmbtu', 'ect', 'enron']

Fold 2 Classification Report:

	precision	recall	f1-score	support
ham	0.9828	0.9837	0.9832	5515
spam	0.9843	0.9834	0.9838	5723
accuracy			0.9835	11238
macro avg	0.9835	0.9835	0.9835	11238
weighted avg	0.9835	0.9835	0.9835	11238

Fold 2 Top 100 Most Informative Features for Spam Emails

['nbsp', 'td', 'pills', 'width', 'viagra', 'computron', 'href', 'cialis', 'src',
'voip', 'height', 'meds', 'photoshop', 'adobe', '0310041', 'macromedia', 'font',
'bgcolor', 'ooking', 'br', 'wiil', 'xp', 'php', 'biz', 'spam', '000000',
'tiscali', 'paliourg', 'penis', 'htmlimg', 'materia', 'paypal', 'rnd', '2004',
'img', 'wysak', 'wifi', 'pharmacy', 'valium', 'xanax', 'cf', 'prescription',
'sex', 'eogi', 'gr', 'yap', 'resuits', 'ail', 'dose', 'div', 'colspan',
'stationery', 'oem', 'technoiogy', 'para', '2005', 'internationa', 'thousand',
'vnbl', 'cum', '971', 'drugs', 'darial', 'piease', 'newsietter', 'couid',
'indianapolis', 'andmanyother', 'demokritos', 'iit', 'nigeria', 'lottery',
'valign', '1618', 'pill', 'illustrator', 'lauraan', 'oniy', 'pubiisher', 'wi',
'emerson', 'rolex', 'rfid', 'potentia', 'robotics', 'projecthoneypot', 'ur',
'otcbb', 'mnei', 'customerservice', '4176', 'erections', '1226030', 'aopen',
'viewsonic', 'jebel', 'oniine', 'targus', 'shouid', 'leth']

Fold 2 Top 100 Most Informative Features for Ham Emails

['shanbhogue', 'hanks', 'mwh', 'meters', 'luong', 'gtc', 'ews', 'ravi', 'hplno',
'liquidity', 'lannou', 'cpr', 'variances', 'upenn', 'katy', 'hourahead', 'nom',
'maharashtra', 'hplo', 'withers', 'clynes', 'pathed', 'kevin', 'cdnow', 'pm',
'counterparties', 'clem', 'responsibilities', 'wellhead', 'equistar', 'outage',
'2001', 'anjam', 'reliantenergy', 'origination', '77002', 'memo', 'noms',
'6353', 'nominations', 'pops', 'aep', 'txu', 'midcon', 'dabhol', 'curves',
'redeliveries', 'houston', 'vasant', 'crenshaw', 'zimin', 'entex', 'eastrans',
'derivatives', '853', 'gibner', 'skilling', 'stinson', 'allocated', 'tenaska',
'ljm', 'cera', 'buyback', 'enronxgate', 'gco', 'dbcaps', 'mseb', 'eol', 'aimee',
'enw', 'iferc', 'epmi', 'louise', 'dpc', 'ebs', 'ferc', '713', 'hplc',
'enrononline', 'volumes', 'pec', 'meter', 'hou', 'counterparty', 'hsc',
'fastow', 'teco', 'xls', 'vince', 'fyi', 'ees', 'dynegy', 'sitara', 'hpl',
'ena', 'kaminski', 'daren', 'mmbtu', 'ect', 'enron']

Fold 3 Classification Report:

	precision	recall	f1-score	support
ham	0.9846	0.9833	0.9839	5515
spam	0.9839	0.9851	0.9845	5723
accuracy			0.9842	11238
macro avg	0.9843	0.9842	0.9842	11238
weighted avg	0.9843	0.9842	0.9842	11238

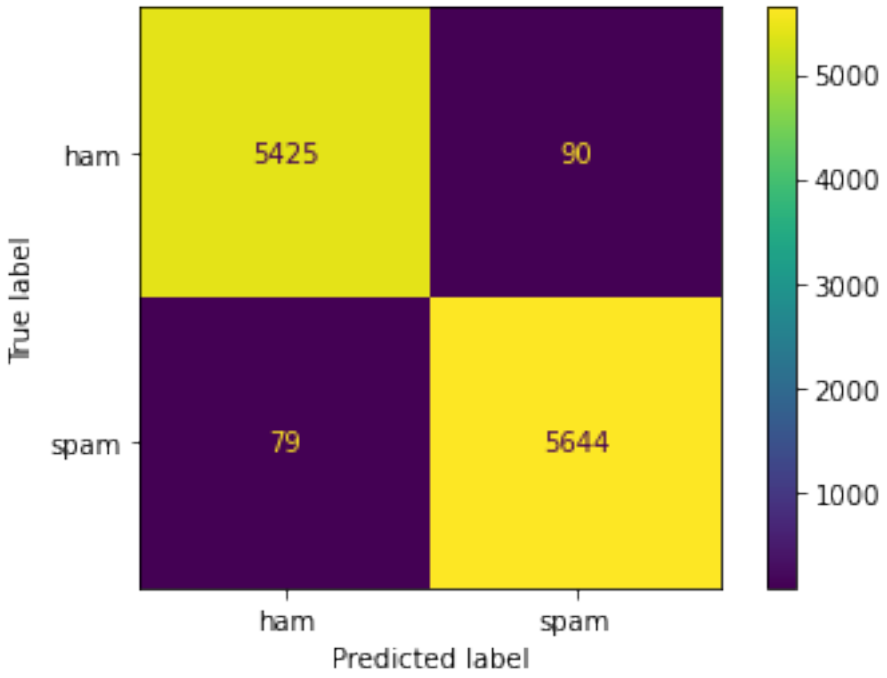
Fold 3 Top 100 Most Informative Features for Spam Emails

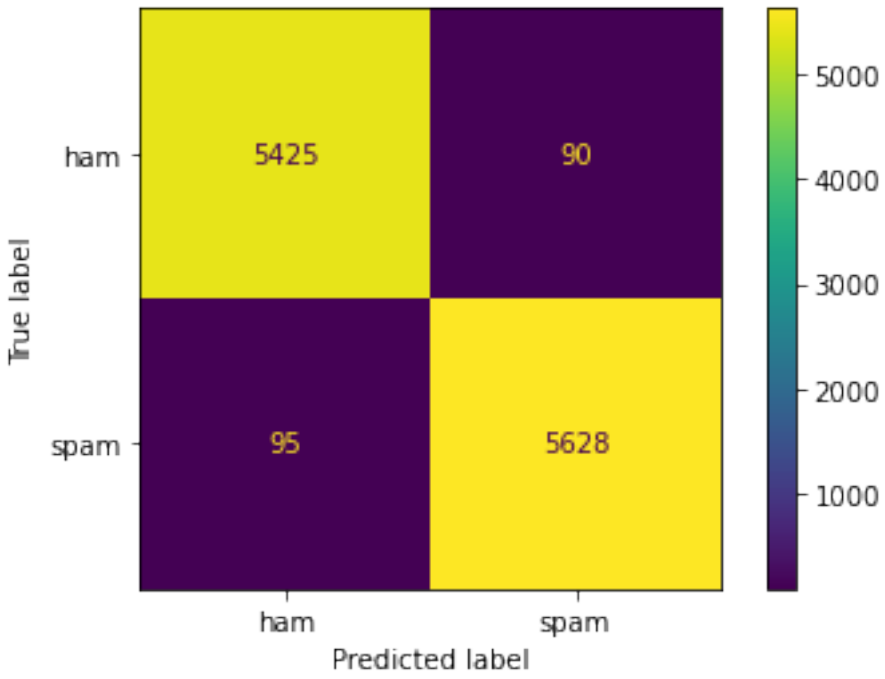
['pills', 'td', 'viagra', 'width', 'nbsp', 'computron', 'height', 'cialis',
'href', 'php', 'src', 'xp', 'photoshop', 'meds', 'macromedia', 'prescription',
'voip', 'adobe', 'bgcolor', '0310041', 'biz', 'font', 'kin', 'eogi', 'wiil',
'ooking', 'spam', 'paliourg', 'penis', '2004', '000000', 'xanax', 'htmlimg',
'rnd', 'sex', 'drugs', 'wysak', 'gr', 'valium', 'paypal', 'wifi', 'materia',
'ebay', 'pharmacy', 'img', 'tiscali', 'br', 'stationery', 'dose', 'emerson',
'1618', 'pill', 'ur', 'ail', 'yap', 'vnbl', 'robotics', 'resuits', '2005',
'demokritos', 'div', '1226030', 'iit', 'wi', 'speedway', 'alt', '4176', 'aopen',
'rolex', 'customerservice', 'viewsonic', 'jebel', 'iomega', 'targus',
'intellinet', 'internationa', 'andmanyother', 'projecthoneypot', 'vicodin',
'potentia', 'colspan', 'illustrator', 'serial', 'vinoble', 'piease', 'thousand',
'darial', '1933', 'technoiogy', 'rfid', 'discreet', 'lottery', 'couid', 'oniy',
'oi', 'oniine', 'emirates', 'cf', '161', 'levitra']

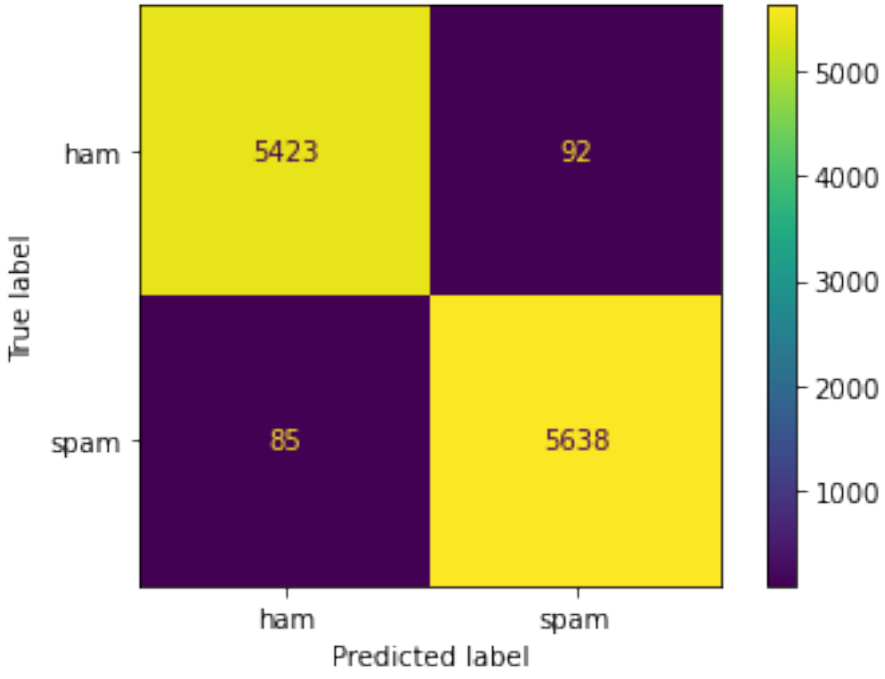
Fold 3 Top 100 Most Informative Features for Ham Emails

['cpr', 'newpower', 'kevin', 'kcs', 'downgraded', 'gtc', 'valero', 'variances',
'cec', 'wynne', 'reinhardt', 'lannou', 'upenn', 'meters', 'hplno', 'mseb',
'katy', 'ravi', 'ratings', 'hplo', 'withers', 'enw', 'pat', 'cera',
'nominations', 'dpc', 'pm', 'dabhol', 'pathed', 'clynes', 'cdnow',

```
'origination', 'enronxgate', 'allocated', 'entex', '2001', 'memo', 'equistar',
'clem', 'anjam', 'reliantenergy', 'responsibilities', '77002', 'redeliveries',
'vasant', '6353', 'noms', 'counterparties', 'houston', 'eastrans', 'txu',
'epmi', 'dbcaps', 'aep', 'midcon', 'tenaska', 'ljm', 'gco', 'gibner', 'zimin',
'curves', 'wellhead', 'nom', '853', 'stinson', 'skilling', 'liquidity',
'melissa', 'crenshaw', 'louise', 'buyback', 'aimee', 'eol', 'ebs', 'iferc',
'713', 'hou', 'ferc', 'volumes', 'hplc', 'enrononline', 'pec', 'fyi', 'hsc',
'counterparty', 'teco', 'ees', 'meter', 'xls', 'fastow', 'vince', 'ena', 'hpl',
'sitara', 'kaminski', 'dynegy', 'daren', 'mmbtu', 'ect', 'enron']
```




5425
90
95
5628
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000


5423
92
85
5638
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000

0.0.12 Experiment 2 Summary Table

Average accuracy = 98.42%

Class	Measure	Average
ham	precision	98.43%
spam	precision	98.42%
ham	recall	98.36%
spam	recall	98.49%
ham	f-measure	98.39%
spam	f-measure	98.45%

I was expecting that the accuracy from using the cleaned emails vs the raw emails would have been better, but that is not the case. The accuracy went down by roughly 0.40%. Using the cleaned emails, there are about 177 misclassifications per 11,238 emails (about 50 more misclassifications than experiment 1). It does not sound like much but it is actually almost 50% more misclassifications than experiment 1. This is a sign that there was lost information from cleaning the emails. One possible reason for this is that I went too aggressive in my email cleaning methods. In the next experiment, I will try removing names, numbers, and stopwords before training the model.

0.0.13 Experiment 3: Removing Names, Numbers, and Stop Words

In experiment 3, I will try removing names, numbers, and stop words from the email text prior to training the model. What do I mean by names, numbers, and stop words? More on that soon. The idea here is similar to the idea from experiment 2 in that it should reduce the noise in the data hence improve the results. But we will soon test that theory.

[40]:
```
# Get a list of peoples names by looking at "to", "from", and "cc" lines

"""
The "get_names" function takes in the raw text of an email
and turns it into a list with each component of the list being
a line from the email. For each line, 3 logical tests are
conducted to see if the line is a "to", "from", or "cc" line.
If yes, then the content from the line will be extracted and
tokenized. The result is a list of names of people. This is
then appended to a main list and the main list is returned
after all iterations are complete.

"""

def get_names(text):
```

```
    textsplit = text.split("\n")
    nameslist = []
    for line in textsplit:
        linelower = line.lower()
        t1 = bool(linelower.startswith("cc :"))
        t2 = bool(linelower.startswith("to :"))
        t3 = bool(linelower.startswith("from :"))
        if t1:
            names = re.findall("cc(.*)", linelower)[0]
        elif t2:
            names = re.findall("to(.*)", linelower)[0]
        elif t3:
            names = re.findall("from(.*)", linelower)[0]
        else:
            continue
        nametokens = nltk.word_tokenize(names)
        nameslist += nametokens
    return nameslist
```

[41]:
```
# Use the get_names function on the ham emails
names = []
for email in ham_emails:
    names += get_names(email)
```

[42]:
```
# Make the list of names unique
names = list(set(names))
```

[43]:
```
# How many names were captured?
print(len(names))
```

```
6646
```

[44]:
```
# Print out some examples of the names
print(names[0:200])
```

```
['myers', 'tamez', 'talvitie', 'tseng', 'sr', 'mulligan', 'enserch', 'elf',
'takriti', 'danilov', 'cbwhitta', 'elafandi', 'ranjan', 'opriskl', '53',
'unseen', 'vtaylor', 'g', 'ram', 'kohli', 'mercy', 'goben', 'watkins',
'covington', 'janet', 'kskinner', 'bangle', 'farley', 'sewell', 'cingular',
'mao', 'plemons', 'cobs', 'nanci', 'flood', 'scada', 'jbouill', 'mirant',
'marathonoil', '915', 'weatherstone', 'nina', 'fotiou', 'abercrombie', 'lamas',
'radu', 'casari', 'mara', 'jubran', 'resources', 'accounting', 'castell',
'fpresto', 'valery', 'thomas', 'trevino', 'cdeepak', '5', 'hemmeline', 'clynes',
'ring', 'sankey', 'fels', 'tamma', 'approximation', 'dreiner', 'pdq',
'zionette', 'fischer', 'jesus', 'welsch', 'cousino', 'dan', 'nielsen',
'dtmusselman', 'cowan', 'lucy', 'couret', 'jorgev', 'drillock', 'anguel',
'eddy', 'reblitz', '=', 'mancini', 'lisbet', 'weaman', 'albrecht', 'hurst',
'tonelli', 'dincerler', 'hoog', 'jacob', 'lerea', 'accenture', 'dhar', 'belen',
```

```
'dayvault', 'markharris', 'reves', 'eber', '15', 'brooks', 'madrid', 'garg',
'karkour', 'elliott', 'laad', 'deb', 'sheila', 'todd', 'pickover', 'cao', 'do',
'avistacorp', 'epsa', 'aa', 'bcfisher', 'shikhar', 'mcmills', 'tgary',
'security', 'hrice', 'watkiss', 'dubash', 'hayes', 'inc', 'sampson', 'romano',
'simunek', 'marzena', 'langmade', 'scampbell', 'spradlin', 'bilberry',
'infante', 'mathew', 'dipak', 'ingersoll', 'conley', 'dennis', 'hankamer',
'robyn', 'jspell', 'donald', 'sacerdote', 'jarrow', 'clerichrd', 'cordomr',
'mario', 'ekrapels', 'dcorrig', 'darren', 'quality', 'hr', 'law', 'kunkel',
'gil', 'rachel', 'akllp', 'laurent', 'govenar', '32', 'leroux', 'sherrick',
'mireya', 'same', 'blancke', 'cohen', 'ondreko', 'sigrid', 'hotmail', 'eprime',
'julie', 'deepa', 'gmasson', 'gmprescott', 'charles', 'misha', 'houston',
'mansfield', '00', 'nicholie', 'wiesepape', 'cohrs', 'shields', 'sheidun',
'amador', 'orourke', 'bymank', 'champion', 'plachy', 'mauldin', 'wells',
'johansen', 'billy', 'tapscott', 'bertone', 'poffenroth', 'col']
```

There were a total of 6,646 names that were captured in this process. But note that not all of them are necessarily names. For example, looking at first 200 that I printed out above, you can see things such as “0008”, “0”, “46”, “ea”, “es”, “training”, etc., are not names. But the large majority of them seem to be valid names so I am not too concerned about it.

[45]:
```
# Get a list of numbers from looking through the emails

"""
The "get_numbers" function takes in the raw text of an email
and searches for all of the words that are made up of only
numbers and returns them in a list.
"""

def get_numbers(text):
    return re.findall("[0-9]+", text)
```

[46]:
```
numbers = []
for email in ham_emails:
    numbers += get_numbers(email)
```

[47]:
```
# Make the list of numbers unique
numbers = list(set(numbers))
```

[48]:
```
# How many numbers were captured?
print(len(numbers))
```

```
9620
```

[49]:
```
# Print out some examples of the numbers
print(numbers[0:200])
```

```
['20787', '8802', '3564', '226611', '602', '7019', '2002020202', '0802', '5221',
'4877', '980388', '3595', '26962', '26681', '2002010618', '2110', '4055',
'892261', '2002013123', '1462', '2001122217', '692725', '6939', '31539',
'27267', '957', '866', '500681', '00834', '40346', '1928', '1641', '2002012412',
'4035', '35522', '29320', '53', '24636', '680210', '3138', '2029', '072700',
'6906', '97201', '8886', '3302', '142002', '1163', '1006', '1707', '959',
'1905', '8774', '1450', '1212', '10545', '000000000031067', '7928', '0505',
'6067', '2002012013', '094', '2001122303', '7901', '4607', '1129', '194', '867',
'96022367', '421598', '2002013007', '704426', '39196', '4023', '387', '915',
'463', '1864', '8222', '90706', '10501', '9875', '8692', '1038590', '3278',
'530687', '6599', '438', '8007', '0421', '691156', '3918', '1770', '133344',
'7493', '4431', '2001122309', '1300', '4705', '33396', '6161', '0521', '5',
'2126', '10018', '703009', '830513', '40000', '24881', '430', '981511',
'986673', '9854', '4745', '377169', '96029028', '982', '0348', '271595', '9523',
'741976', '2002011314', '2002013110', '986581', '22042', '2002020105', '018',
'7240', '0820', '617161', '544392', '1394', '133169', '380', '132976', '5031',
'8945', '1998', '1361763', '286538', '1357', '693081', '18690', '8339', '6327',
'275', '6969504', '7563', '9883', '96046959', '2002012809', '5995', '1249',
'4279', '4420', '26884', '3050', '15', '54763', '3434', '416019', '2002011608',
'3911', '0477', '90316789627', '77019', '6146', '4938', '986831', '8405',
'724368', '250', '40547', '0102', '36100', '1385', '3072', '378608', '6119',
'0733', '749907', '255326837', '0121', '2001122005', '8863', '4809', '121',
'131054', '1880', '5603', '46994', '1231', '39807', '78712', '6245', '1952',
'0741', '6347', '9376', '30396']
```

There were a total of 9,620 numbers that were captured in this process. These numbers could represent employee ids, a quantity of something, or a phone number, for example. As such, it is likely that most of these numbers are associated with one particular correspondence and only appear once or a small handful of times in the email corpus. Thus, since they appear so sparsely, it is likely that they will be noise rather than meaningful information for the model to learn from.

[50]:
```
# List of nltk stopwords
nltkstopwords = list(nltk.corpus.stopwords.words("english"))
print(nltkstopwords)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
"you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's",
'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what',
'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is',
'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
```

```
'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now',
'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn',
"couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn',
"hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't",
'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
"shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn',
"wouldn't"]
```

[51]:
```
# Put together the nltk stopwords and my stopwords
stopwords = names + nltkstopwords + numbers
```

The total list of stop words includes the names, numbers, and the standard english stop words from the nltk library. These stop words will be passed to the vectorizer which removes them during the vectorization process.

[52]:
```
# Initiate a vectorizer
vectorizer = CountVectorizer(encoding = "latin-1", stop_words =␣
↪frozenset(stopwords))
```

[54]:
```
# Iterate through fold 1, 2, and 3
for cv in [1, 2, 3]:

    # When cv = 1, fold 1 & and 2 are used for training and fold 3 is used for␣
↪testing
    if cv == 1:
        X_train = emaildf[emaildf.CVSet != 3].RawEmail
        X_test = emaildf[emaildf.CVSet == 3].RawEmail
        y_train = emaildf[emaildf.CVSet != 3].Label
        y_test = emaildf[emaildf.CVSet == 3].Label

    # When cv = 2, fold 2 & 3 are used for training and fold 1 is used for␣
↪testing
    elif cv == 2:
        X_train = emaildf[emaildf.CVSet != 1].RawEmail
        X_test = emaildf[emaildf.CVSet == 1].RawEmail
        y_train = emaildf[emaildf.CVSet != 1].Label
        y_test = emaildf[emaildf.CVSet == 1].Label

    # When cv = 3, fold 3 & 1 are used for training and fold 2 is used for␣
↪testing
    else:
        X_train = emaildf[emaildf.CVSet != 2].RawEmail
        X_test = emaildf[emaildf.CVSet == 2].RawEmail
```

```
        y_train = emaildf[emaildf.CVSet != 2].Label
        y_test = emaildf[emaildf.CVSet == 2].Label

    # Vectorize the train data
    X_train_vec = vectorizer.fit_transform(X_train)

    # Vectorize the test data
    X_test_vec = vectorizer.transform(X_test)

    # Initiate a multinomial naive bayes model
    model = MultinomialNB()

    # Fit the model on the train data
    model.fit(X_train_vec, y_train)

    # Make predictions based on the test data
    y_pred = model.predict(X_test_vec)

    # Print out the classification report for the current fold
    print("Fold ", cv, " Classification Report:", "\n")
    print(classification_report(y_test, y_pred, digits = 4), "\n")

    # Print out a confusion matrix for the current fold
    ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels =␣
↪["ham", "spam"]).plot()

    # Create a dataframe with ham and spam log probabilities by word
    feature_log_ratio = pandas.DataFrame(list(zip(
    model.feature_log_prob_[0],
    model.feature_log_prob_[1],
    vectorizer.get_feature_names_out())),
    columns = ["ham", "spam", "token"])

    # Add a column for the ratio
    feature_log_ratio["ratio"] = feature_log_ratio["ham"] /␣
↪feature_log_ratio["spam"]

    # Sort the dataframe by ratio in descending order
    feature_log_ratio.sort_values(by = "ratio", ascending = False, inplace =␣
↪True)

    # Print out the top 100
    print("Fold ", cv, " Top 100 Most Informative Features for Spam Emails",␣
↪"\n")
    print(list(feature_log_ratio.head(100).token), "\n")

    # Print out the bottom 100
```

```
    print("Fold ", cv, " Top 100 Most Informative Features for Ham Emails",␣
↪"\n")
    print(list(feature_log_ratio.tail(100).token), "\n")
```

```
Fold  1  Classification Report:

              precision    recall  f1-score   support

         ham     0.9604    0.9940    0.9769      5515
        spam     0.9940    0.9605    0.9770      5723

    accuracy                         0.9770     11238
   macro avg     0.9772    0.9773    0.9770     11238
weighted avg     0.9775    0.9770    0.9770     11238


Fold  1  Top 100 Most Informative Features for Spam Emails

['pills', 'td', 'nbsp', 'viagra', 'width', 'computron', 'cialis', 'href',
'meds', 'spam', 'src', 'xp', 'paliourg', 'voip', 'macromedia', 'photoshop',
'adobe', 'div', 'drugs', 'bgcolor', 'ooking', 'wiil', 'pharmacy',
'prescription', 'xanax', 'height', 'logos', 'php', 'valium', 'lottery', 'penis',
'softwares', 'gr', 'pill', 'sex', 'eogi', '0310041', 'stationery', 'paypal',
'moopid', 'demokritos', 'iit', 'materia', 'dose', 'wysak', 'img', 'htmlimg',
'yap', 'ur', 'darial', 'fontfont', 'rolex', 'que', 'oniine', 'serial', 'ail',
'internationa', 'gra', 'resuits', 'vicodin', 'vnbl', 'illustrator', '1618',
'couid', 'technoiogy', 'piease', 'projecthoneypot', 'newsietter', 'otcbb', 'oi',
'rnd', 'jebel', 'rocket', 'robotics', 'oniy', 'rfid', 'andmanyother', 'valign',
'emerson', 'lauraan', 'sofftwaares', 'profiled', 'nomad', 'pubiisher',
'squirrelmail', 'colspan', 'erections', 'geec', 'potentia', 'cellpadding',
'shouid', 'font', 'um', 'tongue', 'wel', 'rolete', 'phentermine', 'oo', '4176',
'viewsonic']

Fold  1  Top 100 Most Informative Features for Ham Emails

['derivatives', 'cdnow', 'reuters', 'neon', 'mwh', 'afx', 'goliad', 'garven',
'deregulation', 'rto', 'newswires', 'crore', 'westdesk', 'tap', 'deliveries',
'cpr', 'dwr', 'hesco', 'lsu', 'pg', 'citigroup', 'reveffo', 'mtbe', 'ehronline',
'revised', 'egmnom', 'fma', 'socal', 'meeting', 'outage', 'origination',
'internship', 'flowed', 'gtc', 'mtr', 'revisions', 'panenergy', 'ews', 'tetco',
'garp', 'waha', 'cleburne', 'pager', 'variances', 'encina', 'deal', 'kcs',
'allocations', 'allocated', 'pathed', 'pops', 'meters', 'liquidity', 'hplnl',
'tejas', 'counterparties', 'lsk', 'responsibilities', 'ljm', 'redeliveries',
'cornhusker', 'dbcaps', 'dabhol', 'memo', 'epmi', 'unify', 'curves', 'katy',
'hourahead', 'nomination', 'hplno', 'midcon', 'mseb', 'hplo', 'nominations',
'fyi', 'entex', 'wellhead', 'iferc', 'schedules', 'dpc', 'equistar',
'enrononline', 'buyback', 'ebs', 'tenaska', 'nom', 'eastrans', 'hplc', 'hsc',
'counterparty', 'actuals', 'volumes', 'noms', 'teco', 'forwarded', 'meter',
```

```
'sitara', 'xls', 'mmbtu']

Fold  2  Classification Report:

              precision    recall  f1-score   support

         ham     0.9560    0.9958    0.9755      5515
        spam     0.9958    0.9558    0.9754      5723

    accuracy                         0.9754     11238
   macro avg     0.9759    0.9758    0.9754     11238
weighted avg     0.9763    0.9754    0.9754     11238


Fold  2  Top 100 Most Informative Features for Spam Emails

['nbsp', 'td', 'pills', 'viagra', 'width', 'computron', 'cialis', 'meds',
'href', 'src', 'paliourg', 'voip', 'photoshop', 'height', 'macromedia', 'adobe',
'font', 'xp', '0310041', 'bgcolor', 'ooking', 'wiil', 'oem', 'pharmacy', 'spam',
'sex', 'penis', 'php', 'xanax', 'valium', 'prescription', 'gr', 'paypal',
'rolex', 'div', 'wysak', 'tiscali', 'img', 'htmlimg', 'pill', 'materia', 'rnd',
'demokritos', 'iit', 'wifi', 'drugs', 'stationery', 'yap', 'cf', 'cum', 'ail',
'darial', 'dose', 'eogi', 'para', 'technoiogy', 'resuits', 'oniine', 'vicodin',
'colspan', 'internationa', 'foresee', 'lottery', 'couid', 'piease', 'vnbl',
'1618', 'gra', 'newsietter', 'ur', 'illustrator', 'otcbb', 'jebel', 'fontfont',
'andmanyother', 'emerson', 'serial', 'projecthoneypot', 'valign', 'erections',
'softwares', 'pubiisher', 'lauraan', 'oniy', 'shouid', 'potentia', 'robotics',
'rfid', 'tr', 'oi', 'regalis', 'technoiogies', 'phentermine', 'vi', 'oo',
'geec', 'mnei', 'viewsonic', 'prozac', '4176']

Fold  2  Top 100 Most Informative Features for Ham Emails

['dth', 'garven', 'newswires', 'westdesk', 'interconnect', 'imbalance',
'avails', 'schedules', 'attached', 'ppa', 'interview', 'cilco', 'revision',
'ercot', 'dwr', 'tap', 'crore', 'rto', 'neon', 'egmnom', 'deregulation',
'krishna', 'lsu', 'mtr', 'meeting', 'mwh', 'pager', 'vols', 'henwood',
'reveffo', 'internship', 'socal', 'waha', 'encina', 'cpr', 'revisions',
'liquidity', 'tetco', 'maharashtra', 'gtc', 'variances', 'ews', 'allocations',
'memo', 'cleburne', 'kcs', 'panenergy', 'meters', 'deal', 'counterparties',
'hplnl', 'unify', 'derivatives', 'cdnow', 'pathed', 'katy', 'responsibilities',
'tejas', 'origination', 'dabhol', 'nomination', 'redeliveries', 'pops',
'wellhead', 'outage', 'cornhusker', 'fyi', 'hourahead', 'hplno', 'curves',
'ljm', 'allocated', 'dbcaps', 'mseb', 'hplo', 'iferc', 'epmi', 'equistar',
'dpc', 'buyback', 'midcon', 'nominations', 'entex', 'ebs', 'nom', 'tenaska',
'enrononline', 'volumes', 'eastrans', 'hplc', 'counterparty', 'actuals', 'hsc',
'noms', 'teco', 'forwarded', 'meter', 'xls', 'sitara', 'mmbtu']

Fold  3  Classification Report:
```

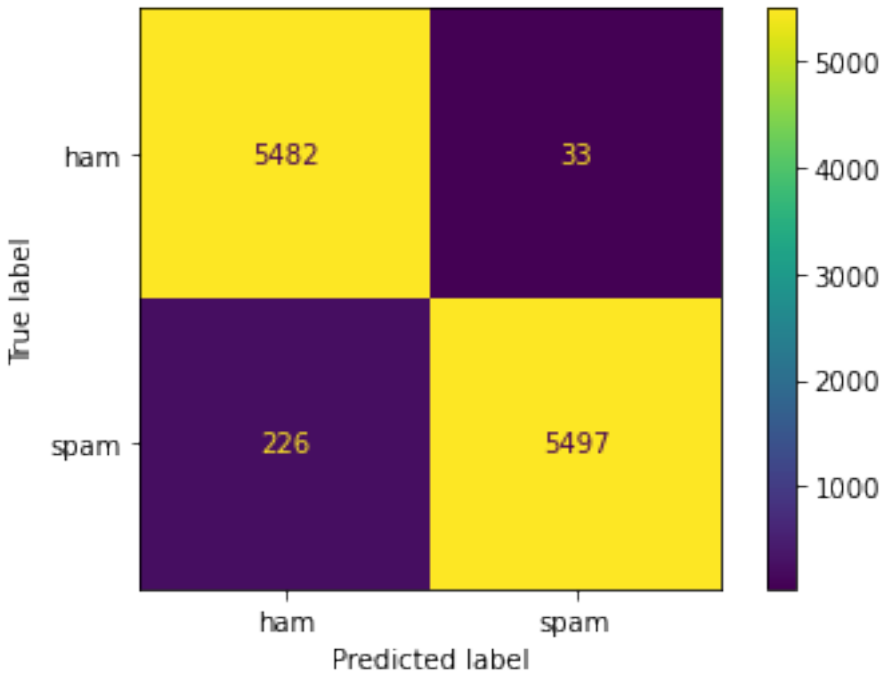
	precision	recall	f1-score	support
ham	0.9644	0.9917	0.9778	5515
spam	0.9917	0.9647	0.9780	5723
accuracy			0.9779	11238
macro avg	0.9781	0.9782	0.9779	11238
weighted avg	0.9783	0.9779	0.9779	11238

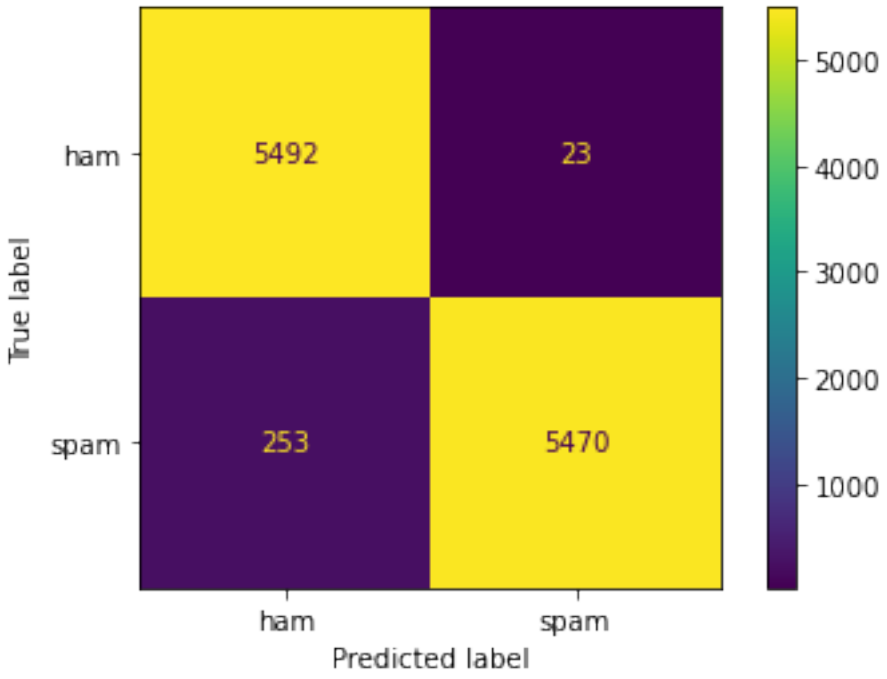
Fold 3 Top 100 Most Informative Features for Spam Emails

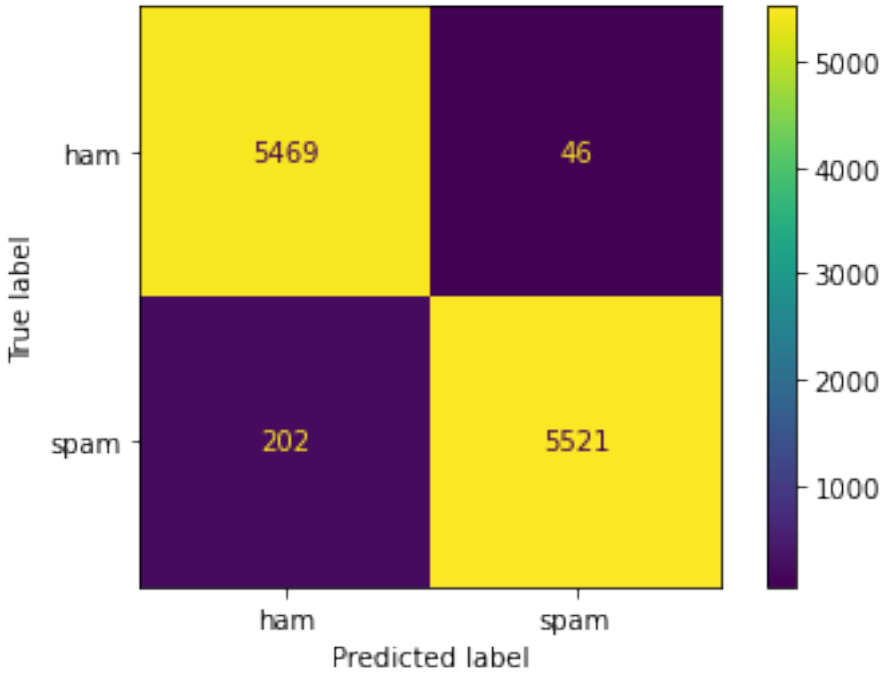
['pills', 'td', 'viagra', 'computron', 'width', 'nbsp', 'cialis', 'xp',
'height', 'meds', 'href', 'php', 'paliourg', 'src', 'prescription', 'photoshop',
'macromedia', 'voip', 'adobe', 'bgcolor', '0310041', 'sex', 'spam', 'gr',
'penis', 'xanax', 'eogi', 'wiil', 'pharmacy', 'ooking', 'valium', 'drugs',
'pill', 'demokritos', 'iit', 'rolex', 'wysak', 'htmlimg', 'rnd', 'paypal',
'vicodin', 'div', 'stationery', 'wifi', 'emerson', 'oniine', 'img', 'materia',
'1618', 'ur', 'dose', 'ail', 'yap', 'tiscali', 'jebel', 'discreet', 'fontfont',
'gra', 'serial', 'oi', 'darial', 'robotics', 'vnbl', 'resuits', 'lottery',
'ebay', 'phentermine', 'projecthoneypot', '1226030', 'internationa',
'technoiogy', 'alt', 'thousand', 'speedway', 'illustrator', 'aopen', '4176',
'viewsonic', 'iomega', '8834454', '8834464', 'vinoble', 'targus', 'intellinet',
'piease', 'andmanyother', 'potentia', 'erections', 'colspan', 'levitra',
'couid', 'rfid', 'corel', 'ambien', 'wel', 'oniy', 'oo', 'pubiisher', 'otcbb',
'mrs']

Fold 3 Top 100 Most Informative Features for Ham Emails

['gtv', 'duffie', 'schedulers', 'avails', 'marketpoint', 'variance',
'deregulation', 'unaccounted', 'ene', 'neon', 'garp', 'pager', 'epgt', 'hesco',
'afx', 'henwood', 'tap', 'cilco', 'flowed', 'ews', 'mmbtus', 'desks', 'egmnom',
'downgraded', 'cpr', 'outage', 'intrastate', 'reuters', 'vols', 'newswires',
'variances', 'socal', 'koch', 'meeting', 'mtr', 'panenergy', 'mseb',
'revisions', 'dwr', 'memo', 'pops', 'gtc', 'internship', 'ratings',
'allocations', 'invoices', 'meters', 'kcs', 'encina', 'unify', 'cleburne',
'dpc', 'waha', 'dabhol', 'deal', 'hplnl', 'cdnow', 'origination', 'pathed',
'katy', 'allocated', 'tejas', 'redeliveries', 'counterparties', 'dbcaps',
'hourahead', 'epmi', 'cornhusker', 'responsibilities', 'ljm', 'liquidity',
'hplno', 'fyi', 'nomination', 'curves', 'hplo', 'iferc', 'entex', 'wellhead',
'equistar', 'ebs', 'nominations', 'midcon', 'buyback', 'tenaska', 'eastrans',
'volumes', 'enrononline', 'hplc', 'actuals', 'counterparty', 'hsc', 'noms',
'nom', 'teco', 'forwarded', 'meter', 'xls', 'sitara', 'mmbtu']


5482
33
226
5497
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000


5492
23
253
5470
ham
spam
True label
ham
spam
Predicted label
5000
4000
3000
2000
1000



0.0.14 Experiment 3 Summary Table

Average accuracy = 97.68%

Class	Measure	Average
ham	precision	96.03%
spam	precision	99.38%
ham	recall	99.38%
spam	recall	96.03%
ham	f-measure	97.67%
spam	f-measure	97.68%

The accuracy in this model decreased a little bit more than the model from experiment 2. Even though the accuracy is lower, the results might be more desireable. The number of Type I misclassifications (about 34 on average) is about half as much as the number of type I misclassifications in experiment 1 (about 61 on average). Why is this a good thing? Well, if a legit email goes into your spam folder, you might never see it. On the other hand, if a spam email goes into your inbox, it is not that big of a deal you just move it into your spam folder. So, in this model, we see that removing the stopwords improved the Type I error but at the expense of the Type II error. In other words, with this model, you can be more confident that your emails are not accidently going into the spam folder, but you can expect to be seeing more spam in your inbox.

0.0.15 Experiment 4: Part of Speech (POS) Tagging

In experiment 4, I will shift gears away from training the model using the text of the emails and instead train the model using part of speech tags. I would venture to say that the grammar in spam emails is different than ham emails. For example, maybe spam emails tend to use a bunch of adjectives since they are trying to hype up their product or whatever it is that they are trying to get you to do. Will it be enough to beat the accuracy scores of the other models? We will find out next.

[56]:
```
# Tokenize the emails
email_tokens = []
for email in emaildf.RawEmail:
    email_tokens.append(nltk.word_tokenize(email))
```

[57]:
```
# POS tag the tokenized emails
email_tags = []
for token_list in email_tokens:
    email_tags.append(nltk.pos_tag(token_list))
```

[64]:
```
# Extract just the POS tags
tags_only = []
for pos_list in email_tags:
    tag_list = []
    for pos_pair in pos_list:
        tag_list.append(pos_pair[1])
    tags_only.append(tag_list)
```

[68]:
```
# Flatten the lists of POS tags into a strings
pos_strings = []
for pos_list in tags_only:
    pos_strings.append(" ".join(pos_list))
```

[69]:
```
# Print out an example of a POS string
print(pos_strings[1000], "\n")
```

```
JJ : NN NNS TO VB NN NN , NN CC NN VBD RP IN DT NNS IN DT IN NN : JJ NN NN CC NN
NN CD . CD NN NN NN CC NN CD . CD NN NN NN VBD CD NN NN CD NN NN CD . CD NN NN
CD . CD NN NN CD . CD NN JJ CD NN
```

[70]:
```
# Add the POS string list to the email df
emaildf["POS"] = pos_strings
```

[71]:
```
# Initiate a vectorizer
vectorizer = CountVectorizer(encoding = "latin-1")
```

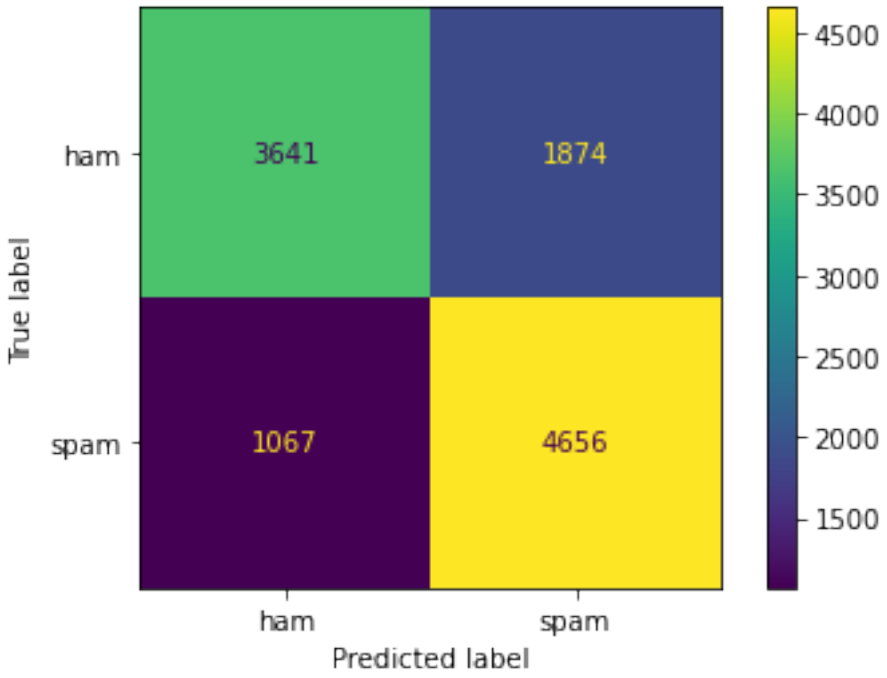
[72]:
```
# Iterate through fold 1, 2, and 3
for cv in [1, 2, 3]:

    # When cv = 1, fold 1 & and 2 are used for training and fold 3 is used for␣
 ↪testing
    if cv == 1:
        X_train = emaildf[emaildf.CVSet != 3].POS
        X_test = emaildf[emaildf.CVSet == 3].POS
        y_train = emaildf[emaildf.CVSet != 3].Label
        y_test = emaildf[emaildf.CVSet == 3].Label

    # When cv = 2, fold 2 & 3 are used for training and fold 1 is used for␣
 ↪testing
    elif cv == 2:
        X_train = emaildf[emaildf.CVSet != 1].POS
        X_test = emaildf[emaildf.CVSet == 1].POS
        y_train = emaildf[emaildf.CVSet != 1].Label
        y_test = emaildf[emaildf.CVSet == 1].Label

    # When cv = 3, fold 3 & 1 are used for training and fold 2 is used for␣
 ↪testing
    else:
        X_train = emaildf[emaildf.CVSet != 2].POS
        X_test = emaildf[emaildf.CVSet == 2].POS
        y_train = emaildf[emaildf.CVSet != 2].Label
        y_test = emaildf[emaildf.CVSet == 2].Label

    # Vectorize the train data
    X_train_vec = vectorizer.fit_transform(X_train)

    # Vectorize the test data
    X_test_vec = vectorizer.transform(X_test)

    # Initiate a multinomial naive bayes model
    model = MultinomialNB()

    # Fit the model on the train data
    model.fit(X_train_vec, y_train)

    # Make predictions based on the test data
    y_pred = model.predict(X_test_vec)

    # Print out the classification report for the current fold
    print("Fold ", cv, " Classification Report:", "\n")
```

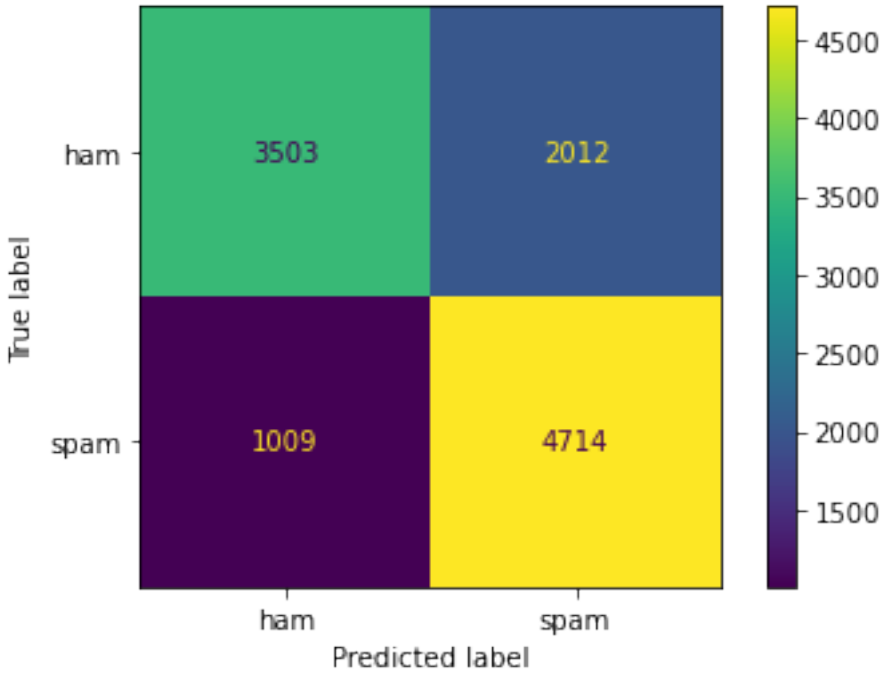
```
    print(classification_report(y_test, y_pred, digits = 4), "\n")

    # Print out a confusion matrix for the current fold
    ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred), display_labels =␣
↪["ham", "spam"]).plot()

    # Create a dataframe with ham and spam log probabilities by word
    feature_log_ratio = pandas.DataFrame(list(zip(
    model.feature_log_prob_[0],
    model.feature_log_prob_[1],
    vectorizer.get_feature_names_out())),
    columns = ["ham", "spam", "token"])

    # Add a column for the ratio
    feature_log_ratio["ratio"] = feature_log_ratio["ham"] /␣
↪feature_log_ratio["spam"]

    # Sort the dataframe by ratio in descending order
    feature_log_ratio.sort_values(by = "ratio", ascending = False, inplace =␣
↪True)

    # Print out the top 100
    print("Fold ", cv, " Top 100 Most Informative Features for Spam Emails",␣
↪"\n")
    print(list(feature_log_ratio.head(100).token), "\n")

    # Print out the bottom 100
    print("Fold ", cv, " Top 100 Most Informative Features for Ham Emails",␣
↪"\n")
    print(list(feature_log_ratio.tail(100).token), "\n")
```

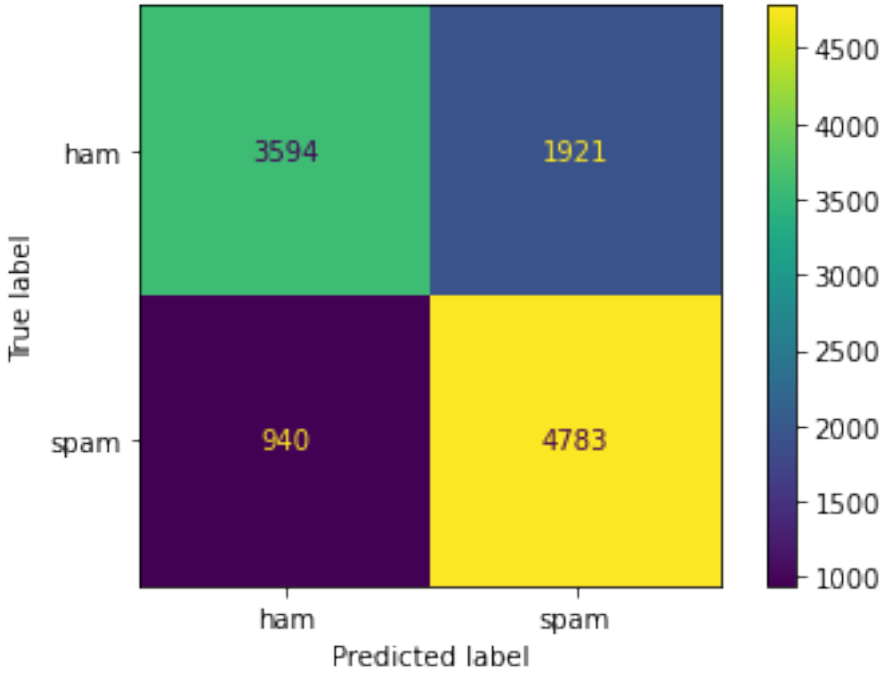
```
Fold  1  Classification Report:

              precision    recall  f1-score   support

         ham     0.7734    0.6602    0.7123      5515
        spam     0.7130    0.8136    0.7600      5723

    accuracy                         0.7383     11238
   macro avg     0.7432    0.7369    0.7361     11238
weighted avg     0.7426    0.7383    0.7366     11238


Fold  1  Top 100 Most Informative Features for Spam Emails

['jj', 'jjs', 'rbs', 'jjr', 'fw', 'prp', 'sym', 'pdt', 'nns', 'rb', 'cc', 'rbr',
'uh', 'nn', 'vbz', 'vbg', 'vbp', 'wrb', 'wdt', 'nnps', 'dt', 'wp', 'vbn', 'vbd',
```

```
'in', 'rp', 'vb', 'pos', 'to', 'ex', 'md', 'nnp', 'cd', 'ls']

Fold  1  Top 100 Most Informative Features for Ham Emails

['jj', 'jjs', 'rbs', 'jjr', 'fw', 'prp', 'sym', 'pdt', 'nns', 'rb', 'cc', 'rbr',
'uh', 'nn', 'vbz', 'vbg', 'vbp', 'wrb', 'wdt', 'nnps', 'dt', 'wp', 'vbn', 'vbd',
'in', 'rp', 'vb', 'pos', 'to', 'ex', 'md', 'nnp', 'cd', 'ls']

Fold  2  Classification Report:

              precision    recall  f1-score   support

         ham     0.7764    0.6352    0.6987      5515
        spam     0.7009    0.8237    0.7573      5723

    accuracy                         0.7312     11238
   macro avg     0.7386    0.7294    0.7280     11238
weighted avg     0.7379    0.7312    0.7286     11238


Fold  2  Top 100 Most Informative Features for Spam Emails

['jj', 'jjs', 'rbs', 'jjr', 'fw', 'prp', 'sym', 'nns', 'pdt', 'rb', 'cc', 'rbr',
'nn', 'vbg', 'vbz', 'wrb', 'vbp', 'uh', 'wdt', 'dt', 'vbn', 'vbd', 'in', 'nnps',
'wp', 'rp', 'vb', 'pos', 'to', 'ex', 'md', 'ls', 'cd', 'nnp']

Fold  2  Top 100 Most Informative Features for Ham Emails

['jj', 'jjs', 'rbs', 'jjr', 'fw', 'prp', 'sym', 'nns', 'pdt', 'rb', 'cc', 'rbr',
'nn', 'vbg', 'vbz', 'wrb', 'vbp', 'uh', 'wdt', 'dt', 'vbn', 'vbd', 'in', 'nnps',
'wp', 'rp', 'vb', 'pos', 'to', 'ex', 'md', 'ls', 'cd', 'nnp']

Fold  3  Classification Report:

              precision    recall  f1-score   support

         ham     0.7927    0.6517    0.7153      5515
        spam     0.7135    0.8358    0.7698      5723

    accuracy                         0.7454     11238
   macro avg     0.7531    0.7437    0.7425     11238
weighted avg     0.7523    0.7454    0.7430     11238


Fold  3  Top 100 Most Informative Features for Spam Emails

['jj', 'jjs', 'fw', 'rbs', 'jjr', 'prp', 'pdt', 'sym', 'nns', 'rb', 'cc', 'rbr',
'nn', 'vbg', 'vbz', 'uh', 'vbp', 'wrb', 'wdt', 'dt', 'wp', 'vbn', 'in', 'vbd',
```

```
'rp', 'vb', 'pos', 'nnps', 'to', 'ex', 'md', 'ls', 'cd', 'nnp']

Fold  3  Top 100 Most Informative Features for Ham Emails

['jj', 'jjs', 'fw', 'rbs', 'jjr', 'prp', 'pdt', 'sym', 'nns', 'rb', 'cc', 'rbr',
'nn', 'vbg', 'vbz', 'uh', 'vbp', 'wrb', 'wdt', 'dt', 'wp', 'vbn', 'in', 'vbd',
'rp', 'vb', 'pos', 'nnps', 'to', 'ex', 'md', 'ls', 'cd', 'nnp']
```




3503
2012
1009
4714
ham
spam
True label
ham
spam
Predicted label
4500
4000
3500
3000
2500
2000
1500


3594
1921
940
4783
ham
spam
True label
ham
spam
Predicted label
4500
4000
3500
3000
2500
2000
1500
1000

0.0.16 Experiment 4 Summary Table

Average accuracy = 73.83%

Class	Measure	Average
ham	precision	78.08%
spam	precision	70.91%
ham	recall	64.90%
spam	recall	82.44%
ham	f-measure	70.88%
spam	f-measure	76.24%

The POS tags were not as informative as I thought they would be. As you can see by looking at the most informative features that were printed out, both the spam emails and the ham emails had a lot of the same most informative features. This explains why there was confusion in the model. The model was not able to classify spam emails very well based solely on the part of speech tags. It did learn something, however, so maybe this knowledge could be combined with the email text to build an even better model, but I will save that exercise for another day.

0.0.17 Project Conclusion

A lot has been covered in this project. To recap, I started off by exploring the email spam data, processed it in different ways, trained several different naive bayes classifiers, and interpreted their results. There were a total of 4 experiments conducted, with each experiment taking on a different approach. Throughout the experiments, I tested my hypotheses and took a detailed look at the results of each trained model.

The margins of accuracy were razor thin, with only slight differences from model to model. Precision, Recall, and F1 scores helped to highlight where the errors occurred in the models. By analyzing the results in this way, I learned that models can be trained in different ways to be tailored for a specific outcome. For example, it might make sense to use a model that has a slightly lower accuracy score overall but performs better on Type I errors. I make this argument in the spam detection task because it is more important to ensure that legit emails are not accidently sent to the spam folder rather than spam emails accidently getting sent to the inbox folder.

While I am pleased with the results of the project, I still feel that there is some unfinished business. Unfortunately, due to time constraints, I was not able to run anymore experiments. One experiment that I would have liked to run would be to put everything together into one classifier (i.e. the email text, the email subject, the pos tags, and other features). It would be interesting to see if combining the features would lead to better results or if it would just cause the curse of dimensionality and end up hindering the results. With that being said, I feel that there is definitely room for improvement.

This project just scratches the surface on how spam detection models are built. In the real world, there is much more that would go into it. For example, huge tech companies that provide email

services such as Google (gmail) or Microsoft (outlook) would try out different models, tune the hyperparameters in the models, and ensure that the models run extremely fast. Not to mention, spammers are constantly trying to figure out new ways to trick the system, so the models need to be smart enough to adapt.