oup-12-idsai-assignment4-sp4-22-23

April 25, 2023

Group 12

Anakha Krishnavilasom Gopalakrishnan - 14 hours

Daniel Juster - 14 hours

1 DAT405/DIT407 Introduction to Data Science and AI

1.1 2022-2023, Reading Period 4

1.2 Assignment 4: Spam classification using Naïve Bayes

The exercise takes place in this notebook environment. 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.

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

[134]:
```
#Download and extract data
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_hard_ham.tar.bz2
!wget https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.bz2
!tar -xjf 20021010_easy_ham.tar.bz2
!tar -xjf 20021010_hard_ham.tar.bz2
!tar -xjf 20021010_spam.tar.bz2
```

```
--2023-04-25 15:25:04--
https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132,
2a04:4e42::644
Connecting to spamassassin.apache.org
(spamassassin.apache.org)|151.101.2.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1677144 (1.6M) [application/x-bzip2]
Saving to: ‘20021010_easy_ham.tar.bz2.5’

20021010_easy_ham.t 100%[===================>]   1.60M  --.-KB/s    in 0.07s

2023-04-25 15:25:05 (23.1 MB/s) - ‘20021010_easy_ham.tar.bz2.5’ saved
[1677144/1677144]

--2023-04-25 15:25:05--
https://spamassassin.apache.org/old/publiccorpus/20021010_hard_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132,
2a04:4e42::644
Connecting to spamassassin.apache.org
(spamassassin.apache.org)|151.101.2.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1021126 (997K) [application/x-bzip2]
Saving to: ‘20021010_hard_ham.tar.bz2.5’

20021010_hard_ham.t 100%[===================>] 997.19K  --.-KB/s    in 0.06s

2023-04-25 15:25:05 (17.5 MB/s) - ‘20021010_hard_ham.tar.bz2.5’ saved
[1021126/1021126]

--2023-04-25 15:25:05--
https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132,
2a04:4e42::644
Connecting to spamassassin.apache.org
(spamassassin.apache.org)|151.101.2.132|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 1192582 (1.1M) [application/x-bzip2]
Saving to: ‘20021010_spam.tar.bz2.5’

20021010_spam.tar.b 100%[===================>]   1.14M  --.-KB/s    in 0.06s

2023-04-25 15:25:06 (17.9 MB/s) - ‘20021010_spam.tar.bz2.5’ saved
[1192582/1192582]
```

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

[135]:
```
!ls -lah
```

```
total 23M
drwxr-xr-x 1 root root 4.0K Apr 25 15:25 .
drwxr-xr-x 1 root root 4.0K Apr 25 11:58 ..
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.1
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.2
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.3
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.4
-rw-r--r-- 1 root root 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.5
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2.1
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2.2
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2.3
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2.4
-rw-r--r-- 1 root root 998K Dec 16  2004 20021010_hard_ham.tar.bz2.5
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.1
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.2
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.3
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.4
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.5
drwxr-xr-x 4 root root 4.0K Apr 21 13:37 .config
drwx--x--x 2  500  500 180K Oct 10  2002 easy_ham
drwx--x--x 2 1000 1000  20K Dec 16  2004 hard_ham
drwxr-xr-x 1 root root 4.0K Apr 21 13:38 sample_data
drwxr-xr-x 2  500  500  40K Oct 10  2002 spam
```

1.2.1 1. Preprocessing:

1.1 Look at a few emails from easy_ham, hard_ham and spam. Do you think you would be able to classify the emails just by inspection? How do you think a succesful model can learn the difference between the different classes of emails?

[136]:
```python
# Write your code for here for looking a few emails

# Start by learning where the folders are....
import os
os.getcwd()
```

[136]: '/content'

[137]:
```python
# They are in /content in the Colab cloud.
# Now, list /content
!ls /content
```

```
20021010_easy_ham.tar.bz2    20021010_hard_ham.tar.bz2.5
20021010_easy_ham.tar.bz2.1  20021010_spam.tar.bz2
```

```
20021010_easy_ham.tar.bz2.2  20021010_spam.tar.bz2.1
20021010_easy_ham.tar.bz2.3  20021010_spam.tar.bz2.2
20021010_easy_ham.tar.bz2.4  20021010_spam.tar.bz2.3
20021010_easy_ham.tar.bz2.5  20021010_spam.tar.bz2.4
20021010_hard_ham.tar.bz2    20021010_spam.tar.bz2.5
20021010_hard_ham.tar.bz2.1  easy_ham
20021010_hard_ham.tar.bz2.2  hard_ham
20021010_hard_ham.tar.bz2.3  sample_data
20021010_hard_ham.tar.bz2.4  spam
```

[138]:

```python
# We are dealing with emails, so we need the email library
import email

FOLDER_easy_ham = '/content/easy_ham'
FOLDER_hard_ham = '/content/hard_ham'
FOLDER_spam = '/content/spam'



FOLDER_test = '/content/easy_ham'
#FOLDER_test = '/content/hard_ham'
#FOLDER_test = '/content/spam'

# Have a look at the first email in FOLDER_test (=0)
filename = os.listdir(FOLDER_test)[0]
filepath = os.path.join(FOLDER_test, filename)

# Parse the email and print its content
with open(filepath, 'rb') as f:
    msg = email.message_from_binary_file(f)
    print(msg.as_string())
```

```
Return-Path: <rssfeeds@example.com>
Delivered-To: yyyy@localhost.example.com
Received: from localhost (jalapeno [127.0.0.1])
        by jmason.org (Postfix) with ESMTP id 6E3EB16F22
        for <jm@localhost>; Tue,  1 Oct 2002 10:36:40 +0100 (IST)
Received: from jalapeno [127.0.0.1]
        by localhost with IMAP (fetchmail-5.9.0)
        for jm@localhost (single-drop); Tue, 01 Oct 2002 10:36:40 +0100 (IST)
Received: from dogma.slashnull.org (localhost [127.0.0.1]) by
    dogma.slashnull.org (8.11.6/8.11.6) with ESMTP id g9181OK15626 for
    <jm@jmason.org>; Tue, 1 Oct 2002 09:01:24 +0100
Message-Id: <200210010801.g9181OK15626@dogma.slashnull.org>
To: yyyy@example.com
From: fark <rssfeeds@example.com>
Subject: Bush orders Sharon to obey UN resolutions so US can gather
    support for breaking of UN resolutions to punish Saddam for breaking UN
```

```
    resolutions
Date: Tue, 01 Oct 2002 08:01:24 -0000
Content-Type: text/plain; encoding=utf-8
X-Spam-Status: No, hits=-604.8 required=5.0
        tests=AWL
        version=2.50-cvs
X-Spam-Level:

URL: http://www.newsisfree.com/click/-1,8390119,1717/
Date: 2002-09-30T11:09:23+01:00

[IMG: http://www.newsisfree.com/Images/fark/smh.com.au.gif ([smh.com.au])]
```

Answer 1.1:

We tested by reading the first email from all three directories. We could clearly see that the easy_ham was a workrelated email, hard_ham was newsletter and spam was spam. Above we only keep easy_ham listed.

Regarding how to create a model, we think that one way would be to look at certain words that are classified as common within spam. If one or more are present, we could look at the probability of this being spam. This could be combined with speach recognition. If it is jibirish, it is more likely to be spam. Also the length of the email (number of characters) can indicate if it is spam or not.

So, given that our folders really have emails in them that are spam, not-spam and hard-to-decide-spam, we can use them to train a model.

1.2 Note that the email files contain a lot of extra information, besides the actual message. Ignore that for now and run on the entire text (in the optional part further down can experiment with filtering out the headers and footers). We don't want to train and test on the same data (it might help to reflect on why if you don't recall).

Split the spam and the ham datasets in a training set and a test set. (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`).

Use only the easy_ham part as ham data for quesions 1 and 2.

[139]:
```python
# Write your code for here for splitting the data



# We need some code to do the split
from sklearn.model_selection import train_test_split

# We also need to be able to load the files into datasets.
# We use a custom script to load our custom datasets.
def load_files(dir):
    files = []
```

```
    with os.scandir(dir) as it:
        for entry in it:
            with open(entry, 'r',  encoding='iso-8859-1') as f:
                data = f.read()
                files.append(data)
    return files


# Load the emails into datasets
DATASET_easy_ham = load_files('easy_ham')
DATASET_hard_ham = load_files('hard_ham')
DATASET_spam = load_files('spam')


# Add label so that we can identify ham-emails. Do this for easy_ham, hard_ham␣
 ↪and combined ham (=ham)
hard_ham_label = ['hardham']*(len(DATASET_hard_ham))
easy_ham_label = ['easyham']*len(DATASET_easy_ham)
ham_label = ['ham']*(len(DATASET_easy_ham + DATASET_hard_ham))

# Also add spam label so that we can identify spam-emails
spam_label = ['spam']*len(DATASET_spam)



# Split into hamtrain and hamtest datasets and put labels on
hard_hamtrain, hard_hamtest, hard_ham_label_train, hard_ham_label_test =␣
 ↪train_test_split(DATASET_hard_ham, hard_ham_label, test_size=0.3,␣
 ↪random_state=42)
easy_hamtrain, easy_hamtest, easy_ham_label_train, easy_ham_label_test =␣
 ↪train_test_split(DATASET_easy_ham, easy_ham_label, test_size=0.3,␣
 ↪random_state=42)
hamtrain, hamtest, ham_label_train, ham_label_test =␣
 ↪train_test_split(DATASET_easy_ham + DATASET_hard_ham, ham_label, test_size=0.
 ↪3, random_state=42)

# Split into spamtrain and spamtest datasets and put labels on
spamtrain, spamtest, spam_label_train, spam_label_test =␣
 ↪train_test_split(DATASET_spam, spam_label, test_size=0.3, random_state=42)




print("Size of dataset DATASET_easy_ham = ", len(DATASET_easy_ham))
print("Size of dataset DATASET_hard_ham = ", len(DATASET_hard_ham))
print("Size of labels  for combined ham = ", len(ham_label))
```

```python
print("Sum of size ham = ", len(DATASET_easy_ham) + len(DATASET_hard_ham))

print("--------------------")

print("Size of dataset hamtrain = ", len(hamtrain))
print("Size of dataset hamtest = ", len(hamtest))
print("Test of size = ", len(hamtrain) + len(hamtest))
print("--------------------")


print("Sum of size spam = ", len(DATASET_spam))
print("Size of dataset spamtrain = ", len(spamtrain))
print("Size of dataset spamtest = ", len(spamtest))
print("Test of size = ", len(spamtrain) + len(spamtest))
print("--------------------")
```

```
Size of dataset DATASET_easy_ham =  2551
Size of dataset DATASET_hard_ham =  250
Size of labels  for combined ham =  2801
Sum of size ham =  2801
--------------------
Size of dataset hamtrain =  1960
Size of dataset hamtest =  841
Test of size =  2801
--------------------
Sum of size spam =  501
Size of dataset spamtrain =  350
Size of dataset spamtest =  151
Test of size =  501
--------------------
```

1.2.2 2.1 Write a Python program that:

1. Uses the four datasets from Question 1 (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`)
2. Trains a Naïve Bayes classifier (use the scikit-learn library) on `hamtrain` and `spamtrain`, that classifies the test sets and reports True Positive and False Negative rates on the `hamtest` and `spamtest` datasets. Use `CountVectorizer` (Documentation here) to transform the email texts into vectors. Please note that there are different types of Naïve Bayes Classifier in scikit-learn (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 before you start coding.

[140]:
```
# First version: easy_ham and training


#Imports
from sklearn import datasets
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
from pprint import pprint



mnb = MultinomialNB()
bnb = BernoulliNB()
vectorizer = CountVectorizer()



  #Train the model using the training sets
train = vectorizer.fit_transform(easy_hamtrain + spamtrain)

test_ham = vectorizer.transform(easy_hamtest)
test_spam = vectorizer.transform(spamtest)
test_set = vectorizer.transform(easy_hamtest + spamtest)

#Predict the response for test dataset
mnb.fit(train, easy_ham_label_train + spam_label_train)
bnb.fit(train, easy_ham_label_train + spam_label_train)

pred_mnb_ham = mnb.predict(test_ham)
pred_bnb_ham = bnb.predict(test_ham)

pred_mnb_spam = mnb.predict(test_spam)
pred_bnb_spam = bnb.predict(test_spam)

mnb.score(test_set, easy_ham_label_test + spam_label_test), bnb.score(test_set,␣
 ↪easy_ham_label_test + spam_label_test)

# Show
unique, counts = np.unique(pred_mnb_ham, return_counts=True)
print(f"True positives (Multinomial): {dict(zip(unique,␣
 ↪counts))[easy_ham_label_test[0]]}")
unique, counts = np.unique(pred_bnb_ham, return_counts=True)
```

```
print(f"True positives (Bernoulli):   {dict(zip(unique,␣
↪counts))[easy_ham_label_test[0]]}")

print()

unique, counts = np.unique(pred_mnb_spam, return_counts=True)
print(f"True negatives (Multinomial): {dict(zip(unique,␣
↪counts))[spam_label_test[0]]}")
unique, counts = np.unique(pred_bnb_spam, return_counts=True)
print(f"True negatives (Bernoulli):   {dict(zip(unique,␣
↪counts))[spam_label_test[0]]}")

print()

print(f"Accuracy for multinomial test: {mnb.score(test_set, easy_ham_label_test␣
↪+ spam_label_test):,.2f}")
print(f"Accuracy for bernoulli test: {bnb.score(test_set, easy_ham_label_test +␣
↪spam_label_test):,.2f}")

print()
```

```
True positives (Multinomial): 762
True positives (Bernoulli):   761

True negatives (Multinomial): 127
True negatives (Bernoulli):   71

Accuracy for multinomial test: 0.97
Accuracy for bernoulli test: 0.91
```

1.2.3 2.2 Answer the following questions:

a) What does the CountVectorizer do? Answer 2.2.a

It concatenates our 2 datasets (ham and spam), converts the text to token counts puts the result in a matrix called "train". We need this so we can run machine learning algorithm on it.

b) What is the difference between Multinomial Naive Bayes and Bernoulli Naive Bayes Answer 2.2.b

The difference is **how** they model probabilities

Bernoulli is binary. This means that looking for the existence of words, like bitcoin - Yes or no. This is good to find e.g. spam.

Multinomial is frequency-based. This means "how often" doeas a word occur, like the word "friend". This is better to classify text (is it an informal or a formal email)

1.2.4 3.1 Run the two models:

Run (don't retrain) the two models from Question 2 on spam versus hard-ham. Does the performance differ compared to question 2 when the model was run on spam versus easy-ham? If so, why?

[141]:
```
# Second version: hard_ham and no-training

# make sure that we only train once
#mnb = MultinomialNB()
#print("MultinomialNB trained")
#bnb = BernoulliNB()
#print("BernoulliNB trained")
#vectorizer = CountVectorizer()
#print("vectorizer trained")



#Train the model using the training sets
train = vectorizer.fit_transform(hard_hamtrain + spamtrain)

test_ham = vectorizer.transform(hard_hamtest)
test_spam = vectorizer.transform(spamtest)
test_set = vectorizer.transform(hard_hamtest + spamtest)

#Predict the response for test dataset
mnb.fit(train, hard_ham_label_train + spam_label_train)
bnb.fit(train, hard_ham_label_train + spam_label_train)

pred_mnb_ham = mnb.predict(test_ham)
pred_bnb_ham = bnb.predict(test_ham)

pred_mnb_spam = mnb.predict(test_spam)
pred_bnb_spam = bnb.predict(test_spam)

mnb.score(test_set, hard_ham_label_test + spam_label_test), bnb.score(test_set,␣
 ↪hard_ham_label_test + spam_label_test)

# Show
unique, counts = np.unique(pred_mnb_ham, return_counts=True)
print(f"True positives (Multinomial): {dict(zip(unique,␣
 ↪counts))[hard_ham_label_test[0]]}")
unique, counts = np.unique(pred_bnb_ham, return_counts=True)
print(f"True positives (Bernoulli):   {dict(zip(unique,␣
 ↪counts))[hard_ham_label_test[0]]}")

print()
```

```
unique, counts = np.unique(pred_mnb_spam, return_counts=True)
print(f"True negatives (Multinomial): {dict(zip(unique,␣
↪counts))[spam_label_test[0]]}")
unique, counts = np.unique(pred_bnb_spam, return_counts=True)
print(f"True negatives (Bernoulli):   {dict(zip(unique,␣
↪counts))[spam_label_test[0]]}")

print()

print(f"Accuracy for multinomial test: {mnb.score(test_set, hard_ham_label_test␣
↪+ spam_label_test):,.2f}")
print(f"Accuracy for bernoulli test: {bnb.score(test_set, hard_ham_label_test +␣
↪spam_label_test):,.2f}")

print()
```

```
True positives (Multinomial): 58
True positives (Bernoulli):   35

True negatives (Multinomial): 147
True negatives (Bernoulli):   144

Accuracy for multinomial test: 0.91
Accuracy for bernoulli test: 0.79
```

Answer 3.1:

Yes, it does change! When using hard_ham, emails that are difficult to distinguish from spam, the Bernoulli-classifier finds less OK emails, probably because they do not have things like $, bitcoin etc., but they are still spam.

1.2.5 3.2 Retrain

Retrain new Multinomial and Bernolli Naive Bayes classifers on the combined (easy+hard) ham and spam. Now evaluate on spam versus hard-ham as in 3.1. Also evaluate on spam versus easy-ham. Compare the performance with question 2 and 3.1. What do you observe?

[142]:
```
# Third version: ham(combined) and training

mnb = MultinomialNB()
print("MultinomialNB trained")
bnb = BernoulliNB()
print("BernoulliNB trained")
vectorizer = CountVectorizer()
print("vectorizer trained")
```

```python
#Train the model using the training sets
train = vectorizer.fit_transform(hamtrain + spamtrain)

test_ham = vectorizer.transform(hamtest)
test_spam = vectorizer.transform(spamtest)
test_set = vectorizer.transform(hamtest + spamtest)

#Predict the response for test dataset
mnb.fit(train, ham_label_train + spam_label_train)
bnb.fit(train, ham_label_train + spam_label_train)

pred_mnb_ham = mnb.predict(test_ham)
pred_bnb_ham = bnb.predict(test_ham)

pred_mnb_spam = mnb.predict(test_spam)
pred_bnb_spam = bnb.predict(test_spam)

mnb.score(test_set, ham_label_test + spam_label_test), bnb.score(test_set, ham_label_test + spam_label_test)

# Show
unique, counts = np.unique(pred_mnb_ham, return_counts=True)
print(f"True positives (Multinomial): {dict(zip(unique, counts))[ham_label_test[0]]}")
unique, counts = np.unique(pred_bnb_ham, return_counts=True)
print(f"True positives (Bernoulli):   {dict(zip(unique, counts))[ham_label_test[0]]}")

print()

unique, counts = np.unique(pred_mnb_spam, return_counts=True)
print(f"True negatives (Multinomial): {dict(zip(unique, counts))[spam_label_test[0]]}")
unique, counts = np.unique(pred_bnb_spam, return_counts=True)
print(f"True negatives (Bernoulli):   {dict(zip(unique, counts))[spam_label_test[0]]}")

print()

print(f"Accuracy for multinomial test: {mnb.score(test_set, ham_label_test + spam_label_test):,.2f}")
print(f"Accuracy for bernoulli test: {bnb.score(test_set, ham_label_test + spam_label_test):,.2f}")

print()
```

```
MultinomialNB trained
BernoulliNB trained
vectorizer trained
True positives (Multinomial): 840
True positives (Bernoulli):   840

True negatives (Multinomial): 137
True negatives (Bernoulli):   39

Accuracy for multinomial test: 0.98
Accuracy for bernoulli test: 0.89
```

Answer 3.2:

Accuracy for ham (combined) with re-trained is highest of them all. Multinomial and bernoulli almost the same (although bernoulli is lower).

We can also see that the actual True positivs are very high and the same for both classfiers.

Re-training and combined is good, although bernoulli struggles when the spams gets more advanced.

1.2.6 3.3 Further improvements

Do you have any suggestions for how performance could be further improved? You don't have to implement them, just present your ideas.

Answer 3.3:

Some things to consider.

The length of the email, including code, could be an indication of spam. We could get the mean lenght and filter out the biggest ones.

It would be possible to remove all standard/common words, like "it", "at", "in", "cc", "reply" etc. This would bring down the total amount of words and make the precision for finding repetitive words in the emails.