Assignment4-1

November 29, 2022

1 DAT405 Introduction to Data Science and AI - Shivneshwar Velayutham (Chalmers)

1.1 2022-2023, Reading Period 2

1.2 Assignment 4: Spam classification using Naïve Bayes

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

The exercise takes place in a notebook environment where you can chose to use Jupyter or Google Colabs. We recommend you use Google Colabs as it will facilitate remote group-work and makes the assignment less technical. Hints: You can execute certain linux shell commands by prefixing the command with `!`. You can insert Markdown cells and code cells. The first you can use for documenting and explaining your results the second you can use writing code snippets that execute the tasks required.

In this assignment you will implement a Naïve Bayes classifier in Python that will classify emails into spam and non-spam ("ham") classes. Your program should be able to train on a given set of spam and "ham" datasets. You will work with the datasets available at https://spamassassin.apache.org/old/publiccorpus/. There are three types of files in this location: - easy-ham: non-spam messages typically quite easy to differentiate from spam messages. - hard-ham: non-spam messages more difficult to differentiate - spam: spam messages

Execute the cell below to download and extract the data into the environment of the notebook – it will take a few seconds. If you chose to use Jupyter notebooks you will have to run the commands in the cell below on your local computer, with Windows you can use 7zip (https://www.7-zip.org/download.html) to decompress the data.

[222]:
```
#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
```

```
--2022-11-29 23:51:47--
https://spamassassin.apache.org/old/publiccorpus/20021010_easy_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
```

```
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.bz2.8’

20021010_easy_ham.t 100%[===================>]   1.60M  --.-KB/s    in 0.09s

2022-11-29 23:51:48 (18.5 MB/s) - ‘20021010_easy_ham.tar.bz2.8’ saved
[1677144/1677144]

--2022-11-29 23:51:48--
https://spamassassin.apache.org/old/publiccorpus/20021010_hard_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
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.bz2.8’

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

2022-11-29 23:51:48 (13.4 MB/s) - ‘20021010_hard_ham.tar.bz2.8’ saved
[1021126/1021126]

--2022-11-29 23:51:48--
https://spamassassin.apache.org/old/publiccorpus/20021010_spam.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)… 151.101.2.132
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.bz2.8’

20021010_spam.tar.b 100%[===================>]   1.14M  --.-KB/s    in 0.07s

2022-11-29 23:51:48 (16.0 MB/s) - ‘20021010_spam.tar.bz2.8’ saved
[1192582/1192582]
```

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

[223]:
```
!ls -lah
```

```
total 68632
drwxr-xr-x    35 shivneshwarvelayutham  staff   1.1K Nov 29 23:51 .
drwxr-xr-x     9 shivneshwarvelayutham  staff   288B Nov 29 00:20 ..
-rw-r--r--@    1 shivneshwarvelayutham  staff   6.0K Nov 29 22:06 .DS_Store
```

```
drwxr-xr-x     3 shivneshwarvelayutham  staff    96B Nov 29 15:12
.ipynb_checkpoints
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.1
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.2
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.3
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.4
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.5
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.6
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.7
-rw-r--r--     1 shivneshwarvelayutham  staff   1.6M Jun 29  2004
20021010_easy_ham.tar.bz2.8
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.1
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.2
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.3
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.4
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.5
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.6
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.7
-rw-r--r--     1 shivneshwarvelayutham  staff   997K Dec 16  2004
20021010_hard_ham.tar.bz2.8
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.1
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.2
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.3
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.4
```

```
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.5
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.6
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.7
-rw-r--r--     1 shivneshwarvelayutham  staff   1.1M Jun 29  2004
20021010_spam.tar.bz2.8
-rw-r--r--@    1 shivneshwarvelayutham  staff    32K Nov 29 23:51
Assignment4-1.ipynb
drwx--x--x  2553 shivneshwarvelayutham  staff    80K Nov 29 23:51
easy_ham
drwx--x--x   252 shivneshwarvelayutham  staff   7.9K Nov 29 23:51
hard_ham
drwxr-xr-x   503 shivneshwarvelayutham  staff    16K Nov 29 23:51
spam
```

###1. Preprocessing: 1. Note that the email files contain a lot of extra information, besides the actual message. Ignore that for now and run on the entire text. Further down (in the higher-grade part), you will be asked to filter out the headers and footers. 2. We don't want to train and test on the same data. Split the spam and the ham datasets in a training set and a test set. (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`)

[224]:
```
import os
import email
import numpy as np
import pandas as pd
from pprint import pprint
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import accuracy_score, classification_report
```

[225]:
```
spam = []
easy_ham = []
hard_ham = []

spam_files = os.listdir('spam')
for spam_file in spam_files:
    with open(os.path.join('spam', spam_file), encoding="latin-1") as f:
        spam.append(f.read())

ham_files = os.listdir('easy_ham')
for ham_file in ham_files:
    with open(os.path.join('easy_ham', ham_file), encoding="latin-1") as f:
        easy_ham.append(f.read())
```

```
ham_files = os.listdir('hard_ham')
for ham_file in ham_files:
    with open(os.path.join('hard_ham', ham_file), encoding="latin-1") as f:
        hard_ham.append(f.read())
```

[226]:
```
def split_and_train(df, vectorizer, classifiers):
    X=df["Text"]
    y=df["Label"]

    print("\nNumber of spam/ham in entire set")
    print(y.value_counts())
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪shuffle=True , stratify=y)
    print("\nNumber of spam/ham in training set")
    print(y_train.value_counts())

    for classifier in classifiers:
        print("\nTrying " + str(classifier))
        pipe = Pipeline(steps=[('vectorize', vectorizer), ('classifier',␣
 ↪classifier)])
        pipe.fit(X_train, y_train)
        y_predict = pipe.predict(X_test)

        print("Accuracy score: " + str(accuracy_score(y_test, y_predict)))
        print(classification_report(y_test, y_predict))
```

###2. Write a Python program that: 1. Uses four datasets (`hamtrain`, `spamtrain`, `hamtest`, and `spamtest`) 2. Trains a Naïve Bayes classifier (e.g. Sklearn) on `hamtrain` and `spamtrain`, that classifies the test sets and reports True Positive and False Negative rates on the `hamtest` and `spamtest` datasets. You can use `CountVectorizer` to transform the email texts into vectors. Please note that there are different types of Naïve Bayes Classifier in SKlearn (Documentation here). Test two of these classifiers that are well suited for this problem - Multinomial Naive Bayes
- Bernoulli Naive Bayes.

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

[227]:
```
print("Training with both easy and hard ham")
df1 = pd.DataFrame (spam, columns = ['Text'])
df1['Label'] = 'spam'

df2 = pd.DataFrame (easy_ham + hard_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(strip_accents='ascii',␣
 ↪encoding="latin-1"), [MultinomialNB(), BernoulliNB()])
```

```
Training with both easy and hard ham
```

```
Number of spam/ham in entire set
ham     2801
spam     501
Name: Label, dtype: int64

Number of spam/ham in training set
ham     2240
spam     401
Name: Label, dtype: int64

Trying MultinomialNB()
Accuracy score: 0.9773071104387292
              precision    recall  f1-score   support

         ham       0.98      0.99      0.99       561
        spam       0.97      0.88      0.92       100

    accuracy                           0.98       661
   macro avg       0.97      0.94      0.95       661
weighted avg       0.98      0.98      0.98       661


Trying BernoulliNB()
Accuracy score: 0.8910741301059002
              precision    recall  f1-score   support

         ham       0.89      1.00      0.94       561
        spam       1.00      0.28      0.44       100

    accuracy                           0.89       661
   macro avg       0.94      0.64      0.69       661
weighted avg       0.90      0.89      0.86       661
```

Your discussion here

1.2.1 3.Run your program on

- Spam versus easy-ham
- Spam versus hard-ham.

[234]:
```
print("Training with only easy ham")
df2 = pd.DataFrame (easy_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(strip_accents='ascii',␣
 ↪encoding="latin-1"), [MultinomialNB(), BernoulliNB()])
```

```
print("Training with only hard ham")
df2 = pd.DataFrame (hard_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(strip_accents='ascii',␣
 ↪encoding="latin-1"), [MultinomialNB(), BernoulliNB()])
```

```
Training with only easy ham

Number of spam/ham in entire set
ham     2551
spam     501
Name: Label, dtype: int64

Number of spam/ham in training set
ham     2040
spam     401
Name: Label, dtype: int64

Trying MultinomialNB()
Accuracy score: 0.9656301145662848
              precision    recall  f1-score   support

         ham       0.96      1.00      0.98       511
        spam       0.99      0.80      0.88       100

    accuracy                           0.97       611
   macro avg       0.97      0.90      0.93       611
weighted avg       0.97      0.97      0.96       611


Trying BernoulliNB()
Accuracy score: 0.8723404255319149
              precision    recall  f1-score   support

         ham       0.87      0.99      0.93       511
        spam       0.84      0.27      0.41       100

    accuracy                           0.87       611
   macro avg       0.86      0.63      0.67       611
weighted avg       0.87      0.87      0.84       611

Training with only hard ham

Number of spam/ham in entire set
spam    501
```

```
ham     250
Name: Label, dtype: int64

Number of spam/ham in training set
spam    400
ham     200
Name: Label, dtype: int64

Trying MultinomialNB()
Accuracy score: 0.9072847682119205
              precision    recall  f1-score   support

         ham       0.88      0.84      0.86        50
        spam       0.92      0.94      0.93       101

    accuracy                           0.91       151
   macro avg       0.90      0.89      0.89       151
weighted avg       0.91      0.91      0.91       151


Trying BernoulliNB()
Accuracy score: 0.8741721854304636
              precision    recall  f1-score   support

         ham       0.97      0.64      0.77        50
        spam       0.85      0.99      0.91       101

    accuracy                           0.87       151
   macro avg       0.91      0.82      0.84       151
weighted avg       0.89      0.87      0.87       151
```

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

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

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

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

[229]:
```
vec = CountVectorizer(strip_accents='ascii', encoding="latin-1").fit(spam +␣
 ↪easy_ham + hard_ham)
bag_of_words = vec.transform(spam + easy_ham + hard_ham)
sum_words = bag_of_words.sum(axis=0)
```

```
words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.
 ↪items()]
words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
pprint(words_freq[:10])
```

```
[('com', 69898),
 ('the', 40814),
 ('to', 38179),
 ('http', 34048),
 ('from', 28715),
 ('td', 28399),
 ('2002', 28274),
 ('3d', 25415),
 ('for', 23845),
 ('net', 22839)]
```

1.3 Answer

1.3.1 4a)

This would help removing words that do not have any correlation to whether an email is spam or not, hence why they are called uninformative for our specific needs. The commons words are printed above and as we can see there are words like **the and to and from** which are uninformative words for whom there is no corellation to whether the email is spam or not and so we don't need to take this words into consideration.

1.3.2 4b)

I am letting sci kit learn do it since e can use already established knowledge of words that are uninformative and can be ignored and this way a mistake of leaving certain words by mistake does not occur.

[230]:
```
print("Training with both easy and hard ham and with stop words")
df1 = pd.DataFrame (spam, columns = ['Text'])
df1['Label'] = 'spam'

df2 = pd.DataFrame (easy_ham + hard_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(stop_words='english',␣
 ↪strip_accents='ascii', encoding="latin-1")
                , [MultinomialNB(), BernoulliNB()])
```

```
Training with both easy and hard ham and with stop words

Number of spam/ham in entire set
ham     2801
spam     501
Name: Label, dtype: int64
```

```
Number of spam/ham in training set
ham     2240
spam     401
Name: Label, dtype: int64

Trying MultinomialNB()
Accuracy score: 0.9863842662632375
              precision    recall  f1-score   support

         ham       0.99      1.00      0.99       561
        spam       0.98      0.93      0.95       100

    accuracy                           0.99       661
   macro avg       0.98      0.96      0.97       661
weighted avg       0.99      0.99      0.99       661


Trying BernoulliNB()
Accuracy score: 0.8910741301059002
              precision    recall  f1-score   support

         ham       0.89      1.00      0.94       561
        spam       1.00      0.28      0.44       100

    accuracy                           0.89       661
   macro avg       0.94      0.64      0.69       661
weighted avg       0.90      0.89      0.86       661
```

###5. Eeking out further performance Filter out the headers and footers of the emails before you run on them. The format may vary somewhat between emails, which can make this a bit tricky, so perfect filtering is not required. Run your program again and answer the following questions: - Does the result improve from 3 and 4? - The split of the data set into a training set and a test set can lead to very skewed results. Why is this, and do you have suggestions on remedies? - What do you expect would happen if your training set were mostly spam messages while your test set were mostly ham messages?

Re-estimate your classifier using `fit_prior` parameter set to `false`, and answer the following questions: - What does this parameter mean? - How does this alter the predictions? Discuss why or why not.

[253]:
```
spam = []
easy_ham = []
hard_ham = []

spam_files = os.listdir('spam')
for spam_file in spam_files:
```

```
    with open(os.path.join('spam', spam_file), encoding="latin-1") as f:
        email_message = email.message_from_file(f)
        body = ""
        if email_message.is_multipart():
            for part in email_message.get_payload():
                body += ''.join(map(str, part.get_payload()))
        else:
            body = email_message.get_payload()
        spam.append(body)


ham_files = os.listdir('easy_ham')
for ham_file in ham_files:
    with open(os.path.join('easy_ham', ham_file), encoding="latin-1") as f:
        email_message = email.message_from_file(f)
        body = ""
        if email_message.is_multipart():
            for part in email_message.get_payload():
                body += ''.join(map(str, part.get_payload()))
        else:
            body = email_message.get_payload()
        easy_ham.append(body)

ham_files = os.listdir('hard_ham')
for ham_file in ham_files:
    with open(os.path.join('hard_ham', ham_file), encoding="latin-1") as f:
        email_message = email.message_from_file(f)
        body = ""
        if email_message.is_multipart():
            for part in email_message.get_payload():
                body += ''.join(map(str, part.get_payload()))
        else:
            body = email_message.get_payload()
        hard_ham.append(body)

print("Training with both easy and hard ham without headers")

df1 = pd.DataFrame (spam, columns = ['Text'])
df1['Label'] = 'spam'

df2 = pd.DataFrame (easy_ham + hard_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(stop_words='english',␣
 ↪strip_accents='ascii', encoding="latin-1"), [MultinomialNB(), BernoulliNB()])
```

```
Training with both easy and hard ham without headers
```

```
Number of spam/ham in entire set
ham     2801
spam     501
Name: Label, dtype: int64

Number of spam/ham in training set
ham     2240
spam     401
Name: Label, dtype: int64

Trying MultinomialNB()
Accuracy score: 0.9803328290468987
              precision    recall  f1-score   support

         ham       0.98      0.99      0.99       561
        spam       0.97      0.90      0.93       100

    accuracy                           0.98       661
   macro avg       0.98      0.95      0.96       661
weighted avg       0.98      0.98      0.98       661


Trying BernoulliNB()
Accuracy score: 0.869894099848714
              precision    recall  f1-score   support

         ham       0.87      1.00      0.93       561
        spam       0.94      0.15      0.26       100

    accuracy                           0.87       661
   macro avg       0.90      0.57      0.59       661
weighted avg       0.88      0.87      0.83       661
```

1.4 Answer

1.4.1 5a)

There isn't in significant difference when the headers and footers are ignored and the accuracies are quite similar to before.

Sometimes splitting of data can result in vastly different training set and test set. This can especially happen if we have certain classes of data (ie. data is split based on certain rules.) Thus care must be taken to ensure that the training data has all the different classification/types of data. The remedy used in this notebook is with the help of the **stratify argument of train_test_split** which is able to ensure that the training data is similarly distributed as the original data.

If training data was mostly spam then the accuracy on the testing data would be quite low as it

might predict all the ham messages as spam. Thus it's quite important to run the classification multiple times to check if there are big oscillations of accuracy scores.

[248]:
```
print("Training with both easy and hard ham and fit_prior set to false")
df1 = pd.DataFrame (spam, columns = ['Text'])
df1['Label'] = 'spam'

df2 = pd.DataFrame (easy_ham + hard_ham, columns = ['Text'])
df2['Label'] = 'ham'

split_and_train(pd.concat([df1, df2]), CountVectorizer(strip_accents='ascii',␣
 ↪encoding="latin-1"), [MultinomialNB(fit_prior=False),␣
 ↪BernoulliNB(fit_prior=False)])
```

```
Training with both easy and hard ham and fit_prior set to false

Number of spam/ham in entire set
ham     2801
spam     501
Name: Label, dtype: int64

Number of spam/ham in training set
ham     2240
spam     401
Name: Label, dtype: int64

Trying MultinomialNB(fit_prior=False)
Accuracy score: 0.9682299546142209
              precision    recall  f1-score   support

         ham       0.97      0.99      0.98       561
        spam       0.96      0.82      0.89       100

    accuracy                           0.97       661
   macro avg       0.97      0.91      0.93       661
weighted avg       0.97      0.97      0.97       661


Trying BernoulliNB(fit_prior=False)
Accuracy score: 0.8714069591527988
              precision    recall  f1-score   support

         ham       0.87      1.00      0.93       561
        spam       0.89      0.17      0.29       100

    accuracy                           0.87       661
   macro avg       0.88      0.58      0.61       661
weighted avg       0.87      0.87      0.83       661
```

1.4.2 5b)

The fit_prior parameter decides whether to learn prior probablities of the different classes from the data or to use uniform prior probabilties (where the probablily of each class occuring is the same). When fit_prior is set to True then it takes into account the distribution of classes in the data during prediction. In our case, as we can have more ham than spam so the model will taken this account. When fit_prior is set to False then each class has an equal probablity of occurring (in our case 0.5 to ham, 0.5 to spam). This will have an impact in predictions since if the data contains a lot of examples of a specific class and fit_prior is set to True then it will be more likely that the model predicts the class which has a lot of examples in the data. This does not occur and each class is treated equally when fit_prior is set to False.