assignment4-1

March 1, 2023

1 DAT405 Introduction to Data Science and AI

1.1 2022-2023, Reading Period 3

1.2 Assignment 4: Spam classification using Naïve Bayes

This assignmetn has three obligatory questions which will be grades as PASS/FAIL. Questions 4-5 are optional and will not be graded, but can be interesting for students aiming for higher grades.

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

[16]:
```
#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
```

```
'wget' is not recognized as an internal or external command,
```

```
operable program or batch file.
'wget' is not recognized as an internal or external command,
operable program or batch file.
'wget' is not recognized as an internal or external command,
operable program or batch file.
tar: Error opening archive: Failed to open '20021010_easy_ham.tar.bz2'
tar: Error opening archive: Failed to open '20021010_hard_ham.tar.bz2'
tar: Error opening archive: Failed to open '20021010_spam.tar.bz2'
```

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

[17]:
```
!ls -lah
```

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

1.2.1 1. Preprocessing:

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

[2]:
```
import pandas as pd
import os
import sklearn.model_selection as skm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import make_pipeline

# Read data from folder and sort it into a dataframe
def readData(folder):
    files = os.listdir(folder) # 'easy_ham'
    df = pd.DataFrame(columns=['file_name', 'data', 'label'])

    for i, file in enumerate(files):
        with open(folder + '/' + file, 'r', encoding="latin-1") as f:
            temp_df = pd.DataFrame({'file_name': file, 'data': f.read(),␣
 ↪'label': folder}, index=[i])
            df = pd.concat([df, temp_df])
    return df
```

```
# Read data
spam = readData('spam')
ham = readData('easy_ham')
hard_ham = readData('hard_ham')
```

1.2.2 2. Write a Python program that:

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

[6]:
```
df = pd.concat([spam, ham])

# Split data into training and testing
X_train, X_test, y_train, y_test = skm.train_test_split(df['data'],␣
 ↪df['label'], test_size=0.1)

model_Multi = make_pipeline(TfidfVectorizer(), MultinomialNB())
model_Bern = make_pipeline(TfidfVectorizer(), BernoulliNB())

model_Multi.fit(X_train, y_train)
model_Bern.fit(X_train, y_train)

y_pred_Multi = model_Multi.predict(X_test)
y_pred_Bern = model_Bern.predict(X_test)

print("Accuracy Multi: ", accuracy_score(y_test, y_pred_Multi))
print("Accuracy Bern: ", accuracy_score(y_test, y_pred_Bern))
print("-"*25)

# Find the confusion matrix
tn1, fp1, fn1, tp1 = confusion_matrix(y_test, y_pred_Multi).ravel()
tn2, fp2, fn2, tp2 = confusion_matrix(y_test, y_pred_Bern).ravel()

# Print the confusion matrix
print("True Positive Multi: " + str(tp1/(tp1+fp1)))
print("False Negative Multi: " + str((tn1/(tp1+fp1))/100))
```

```
print("True Positive Bern: " + str(tp2/(tp2+fp2)))
print("False Negative Bern: " + str((tn2/(tp2+fp2))/100))
```

```
Accuracy Multi:  0.8856209150326797
Accuracy Bern:  0.9281045751633987
-------------------------
True Positive Multi: 1.0
False Negative Multi: 0.13263157894736843
True Positive Bern: 0.9705882352941176
False Negative Bern: 0.0738235294117647
```

1.2.3 3.Run on hard ham:

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

After running both with ham and hard_ham data, we noticed that they were somewhat similar. The overall accuracy remained almost the same, but what differs is the accuracy of true postive/true negative. When using ham, the true positive is almost always 1, while the true negative is lower. When using hard_ham, the values flip, with true negative almost always equaling 1, with its true positive being lower.

As can be seen in the results, the multinomial classifier performs a bit better. The reason behind this is because a multinomial classifier keeps track of the number of occurances (of words) while the bernoulli classifier only keeps track of if a word appears at all. Because of this, the bernoulli classifier underperforms compared to the multinomial classifier when handling longer documents.

Both classifiers can be improved on by being trained on more data. The more data you have, the more accurate the classifier will be. This is because the more data you have, the more likely it is that the classifier will have seen the type of wording before, and thus be able to classify it correctly.

[22]:
```
df = pd.concat([spam, hard_ham])

# Split data into training and testing
X_train, X_test, y_train, y_test = skm.train_test_split(df['data'],␣
 ↪df['label'], test_size=0.1)

model_Multi = make_pipeline(TfidfVectorizer(), MultinomialNB())
model_Bern = make_pipeline(TfidfVectorizer(), BernoulliNB())

model_Multi.fit(X_train, y_train)
model_Bern.fit(X_train, y_train)

y_pred_Multi = model_Multi.predict(X_test)
y_pred_Bern = model_Bern.predict(X_test)

print("Accuracy Multi: ", accuracy_score(y_test, y_pred_Multi))
print("Accuracy Bern: ", accuracy_score(y_test, y_pred_Bern))
print("-"*25)
```

```
# Find the confusion matrix
tn1, fp1, fn1, tp1 = confusion_matrix(y_test, y_pred_Multi).ravel()
tn2, fp2, fn2, tp2 = confusion_matrix(y_test, y_pred_Bern).ravel()

# Print the confusion matrix
print("True Positive Multi: " + str(tn1/(tn1+fp1)))
print("True Negative Multi: " + str(tp1/(tp1+fn1)))
print("True Positive Bern: " + str(tn2/(tn2+fp2)))
print("True Negative Bern: " + str(tp2/(tp2+fn2)))
```

```
Accuracy Multi:  0.9333333333333333
Accuracy Bern:  0.9466666666666667
-------------------------
True Positive Multi: 0.7222222222222222
True Negative Multi: 1.0
True Positive Bern: 0.7777777777777778
True Negative Bern: 1.0
```

1.2.4 4. OPTIONAL - NOT MARKED:

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

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

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

[23]:
```
#Write your code here
```

1.2.5 5. OPTIONAL - NOT MARKED: 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? - What do you expect would happen if your training set were mostly spam messages while your test set were mostly ham messages or vice versa? - Look at the `fit_prior` parameter. What does this parameter mean? Discuss in what settings it can be helpful (you can also test your hypothesis).

[24]:
```
#Write your code here
```