nb_upg1-3

February 16, 2022

1 Assignment 4: Spam classification using Naïve Bayes

1.1 DAT405 Introduction to Data Science and AI

1.1.1 By Pauline Nässlander and Albin Ekström

Hours spent on the assignment:

- Pauline Nässlander: 12 hours
- Albin Ekström: 12 hours

1.2 Question 1, 2 & 3

[186]:
```
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import os
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn import metrics
```

[187]:
```
# defining functions

def read_data(file):
    with open(file, 'rb') as f:
        data = f.read().decode(errors='replace')
        return data

def clean_data(data):
    words = ["content-transfer-encoding", "precedence", "content-type",␣
 ↪"mailman", "x-mailer", "to"] # common words in header
    match = next((x for x in words if x in data.lower()), False)
    if match:
        first_index = data.lower().find(match)

    last = "\n\n\n"
    last_index = data.lower().find(last)

    data = data[first_index:last_index]
```

```python
    return data


def gen_df(path, label, clean):
    di = {"label": [], "text": []}
    for file in os.listdir(path):
        file_data = read_data(path + file)
        if clean:
            file_data = clean_data(file_data)
        di["label"].append(label)
        di["text"].append(file_data)
    df = pd.DataFrame.from_dict(di)
    return df

def into_vec(self, w_filter):
    vectorizer = CountVectorizer(max_df = (1.0 - w_filter), min_df = w_filter)
    X = vectorizer.fit_transform(self)
    return X.toarray()

def multinominal_naive_bayes(Xs_train, ys_train, ys_hamtest, ys_spamtest,␣
 ↪Xs_hamtest, Xs_spamtest):
    # Multinomial Naive# concatenate training

    mnb = MultinomialNB()

    mnb.fit(Xs_train, ys_train)
    ham_pred = mnb.predict(Xs_hamtest)
    spam_pred = mnb.predict(Xs_spamtest)

    print("Multinomial Naive Bayes")
    print("Ham Accuracy:", metrics.accuracy_score(ys_hamtest, ham_pred))
    print("Spam Accuracy:", metrics.accuracy_score(ys_spamtest, spam_pred))


def bernoulli_naive_bayes(Xs_train, ys_train, ys_hamtest, ys_spamtest,␣
 ↪Xs_hamtest, Xs_spamtest):
    # Bernoulli Naive Bayes

    bnb = BernoulliNB(binarize=0.0)

    bnb.fit(Xs_train, ys_train)
    ham_pred = bnb.predict(Xs_hamtest)
    spam_pred = bnb.predict(Xs_spamtest)

    print("Bernoulli Naive Bayes")
    print("Ham Accuracy:", metrics.accuracy_score(ys_hamtest, ham_pred))
    print("Spam Accuracy:", metrics.accuracy_score(ys_spamtest, spam_pred))
```

```
    # This function is taken from user "mtrw" on StackOverflow:
    # https://stackoverflow.com/questions/4601373/
 ↪better-way-to-shuffle-two-numpy-arrays-in-unison
def unison_shuffled_copies(a, b):
    assert len(a) == len(b)
    p = np.random.permutation(len(a))
    return a[p], b[p]

def make_shity_work(mail, df_ham, df_spam, filter):
    mail_vec = into_vec(mail, filter)

    # categorize ham and spam of vectorized object
    ham = mail_vec[:len(df_ham['text']), :]
    spam = mail_vec[len(df_ham['text']):, :]

    # split the data into hamtrain, spamtrain, hamtest, and spamtest
    X_hamtrain, X_hamtest, y_hamtrain, y_hamtest = train_test_split(ham,␣
 ↪df_ham['label'], test_size=0.3)
    X_spamtrain, X_spamtest, y_spamtrain, y_spamtest = train_test_split(spam,␣
 ↪df_spam['label'], test_size=0.3)

    # concatenate training data
    X_train = np.concatenate((X_hamtrain, X_spamtrain))
    y_train = np.concatenate((y_hamtrain, y_spamtrain))

    # shuffel
    X, y = unison_shuffled_copies(X_train, y_train)

    return X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest
```

[188]:
```
# generate dataframes for spam and ham

df_easy_ham = gen_df('data/easy_ham/', "ham", False)
df_hard_ham = gen_df('data/hard_ham/', "ham", False)
df_spam = gen_df('data/spam/', "spam", False)
```

[189]:
```
# concatenate and vectorize all mails

easy_mail = np.concatenate((df_easy_ham['text'], df_spam['text']))
hard_mail = np.concatenate((df_hard_ham['text'], df_spam['text']))
```

1.3 EASY HAM

[190]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(easy_mail,
    df_easy_ham, df_spam, 0)
```

[191]:
```
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Multinomial Naive Bayes
Ham Accuracy: 0.9973890339425587
Spam Accuracy: 0.9
```

[192]:
```
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Bernoulli Naive Bayes
Ham Accuracy: 0.9947780678851175
Spam Accuracy: 0.5
```

1.4 HARD HAM

[193]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(hard_mail,
    df_hard_ham, df_spam, 0)
```

[194]:
```
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Multinomial Naive Bayes
Ham Accuracy: 0.8
Spam Accuracy: 0.98
```

[195]:
```
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Bernoulli Naive Bayes
Ham Accuracy: 0.64
Spam Accuracy: 0.9666666666666667
```

1.4.1 2b

The large difference between multinomial naive bayes and bernoulli naive bayes is that the bernoulli model actively penalizes the absence of a word that indicates a classification while the multinomial model only ignores this. That is the Bernoulli model says whether a word has occurred or not while the multinomial one counts how often words occur.

1.5 Question 4 & 5

[196]:
```
mails = np.concatenate((df_easy_ham['text'], df_hard_ham['text'],
    df_spam['text']))
```

[197]:
```
vectorizer = CountVectorizer().fit(mails)
mails_vec = vectorizer.transform(mails)
```

[198]:
```python
sum = mails_vec.sum(axis=0)
freq_words = [(word, sum[0, i]) for word, i in vectorizer.vocabulary_.items()]
least_freq = sorted(freq_words, key = lambda x: x[1], reverse = False)
most_freq = sorted(freq_words, key = lambda x: x[1], reverse = True)

print("Most frequent: ", most_freq[:20])
print("Least frequent: ", least_freq[:20])
```

```
Most frequent:  [('com', 69898), ('the', 40824), ('to', 38179), ('http', 34049),
('from', 28715), ('td', 28399), ('2002', 28278), ('3d', 25415), ('for', 23847),
('net', 22839), ('font', 22609), ('with', 22181), ('by', 21436), ('width',
20932), ('of', 20336), ('and', 20232), ('localhost', 18916), ('id', 18226),
('received', 17800), ('www', 17481)]
Least frequent:  [('ae79816f16', 1), ('8381145', 1), ('philanthropist', 1),
('gangster', 1), ('cahill', 1), ('04d8916f3d', 1), ('g8366uz07156', 1),
('8c84f2940b3', 1), ('1a407294099', 1), ('1783', 1), ('g8365sb03549', 1),
('transhumantech', 1), ('0209030804570', 1), ('2c3858', 1), ('2c4489999', 1),
('2c00', 1), ('spells', 1), ('overburdened', 1), ('repeater', 1), ('relaying',
1)]
```

1.5.1 4a

This is useful because otherwise the model will adapt to words that may not be useful to determine if the mail is spam or ham. If the most common words, e.g. From, Return-Path, Delivered-To etc. that is in every mail and the most uncommon words that aren't typical for a spam or ham mail where removed, the model can hopefully become more accurate.

By manually searching the mails and looking for very common words we found that words such as the, that, and, on, of etc. occured in most mails. Also words that are related to the headers and footers such as received, from, path etc. occured in almost every email and were therefore very uninformative.

After doing this manual quick overview search we then used countvectorizer to print the 20 most common words and the 20 least common words in the lists above.

[199]:
```python
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(easy_mail,␣
    ↪df_easy_ham, df_spam, 0.05)
print("EASY HAM WITH FILTER 0.05")
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
EASY HAM WITH FILTER 0.05
Multinomial Naive Bayes
Ham Accuracy: 0.9960835509138382
Spam Accuracy: 0.8466666666666667

Bernoulli Naive Bayes
Ham Accuracy: 0.9112271540469974
Spam Accuracy: 0.98
```

[200]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(easy_mail,␣
 ↪df_easy_ham, df_spam, 0.3)
print("EASY HAM WITH FILTER 0.3")
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
EASY HAM WITH FILTER 0.3
Multinomial Naive Bayes
Ham Accuracy: 0.9138381201044387
Spam Accuracy: 0.8133333333333334

Bernoulli Naive Bayes
Ham Accuracy: 0.8903394255874674
Spam Accuracy: 0.9333333333333333
```

[201]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(hard_mail,␣
 ↪df_hard_ham, df_spam, 0.05)
print("HARD HAM WITH FILTER 0.05")
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
HARD HAM WITH FILTER 0.05
Multinomial Naive Bayes
Ham Accuracy: 0.72
Spam Accuracy: 0.98

Bernoulli Naive Bayes
Ham Accuracy: 0.6933333333333334
Spam Accuracy: 0.96
```

[202]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(hard_mail,␣
 ↪df_hard_ham, df_spam, 0.3)
print("HARD HAM WITH FILTER 0.3")
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
HARD HAM WITH FILTER 0.3
Multinomial Naive Bayes
Ham Accuracy: 0.5733333333333334
Spam Accuracy: 0.9733333333333334

Bernoulli Naive Bayes
Ham Accuracy: 0.7733333333333333
Spam Accuracy: 0.9333333333333333
```

1.5.2 4b

When applying a filter to sort out the most common and uncommon words the spam accuracy increases a lot for the easy ham vs spam case both when using multinomial and bernoulli naive bayes. However, the ham accuracy goes down somewhat. We also notice, when comparing filter values 0.05 vs 0.3, that in this case a lower filter gives better accuracy than the higher value on the filter. This is because when we remove the unnecessary words the model won't fit itself to the unnecessary words that don't contribute to a better determination of spam or ham.

In the hard ham vs spam case, the accuracy goes down when applying a filter and decreases even more when we increase the filter value. This behavior may be due to the fact that the model needs the filtered out word in order to determine the character of the hard emails.

[203]:
```
df_easy_ham = gen_df('data/easy_ham/', "ham", True)
df_hard_ham = gen_df('data/hard_ham/', "ham", True)
df_spam = gen_df('data/spam/', "spam", True)
```

[204]:
```
easy_mail = np.concatenate((df_easy_ham['text'], df_spam['text']))
hard_mail = np.concatenate((df_hard_ham['text'], df_spam['text']))
```

[205]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(easy_mail,␣
↪df_easy_ham, df_spam, 0)
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Multinomial Naive Bayes
Ham Accuracy: 0.9973890339425587
Spam Accuracy: 0.8

Bernoulli Naive Bayes
Ham Accuracy: 0.9817232375979112
Spam Accuracy: 0.48
```

[206]:
```
X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest = make_shity_work(hard_mail,␣
↪df_hard_ham, df_spam, 0)
multinominal_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
print("")
bernoulli_naive_bayes(X, y, y_hamtest, y_spamtest, X_hamtest, X_spamtest)
```

```
Multinomial Naive Bayes
Ham Accuracy: 0.5466666666666666
Spam Accuracy: 0.96

Bernoulli Naive Bayes
Ham Accuracy: 0.24
Spam Accuracy: 0.9933333333333333
```

1.5.3 5a

We expected this to improve the results but in our case it did not. This we think is because we might have filtered out to much when deleting the headers and footers. It may also be because some information in the header such as the email adress and subject is important for determening the class of the mail. It could also be because the format of the mails are very different and when running our data cleaning we might be left with some mails that have much headers left and some that have been properly cleaned. this makes mails in the same class look very different which can lead to skewed results. However we think that erasing headers and footers should increase accuracy since only relevan information is left.

1.5.4 5b

The fact that the train-test-split can lead to skewed results is because the spam mails are very few in comparison to the ham mails and as the test-train-split divides differently every time it can happen that the test set consists mostly of spam while the train set consists mostly of ham messages which is not a good division.

1.5.5 5c

If the test set were mostly ham mails and the training sets mostly spam we would expect the model to be quite bad at classifying ham messages since it will have learned mostly on spam and will therefore not recognize ham messeges as good.