DAT405 Assignment 4 – Group 53

Venkata Sai Dinesh Uddagiri - (17 hrs)
Madumitha Venkatesan - (17 hrs)

November 29, 2022

Problem 1

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') [?].

```
# Functions that write the mail content and type to data frame
def readFileContent(path, type):
    rows=[]
    for file_name in os.listdir(path):
        file = os.path.join(path, file_name)
        if os.path.isfile(file):
            with open(file, encoding='latin-1' ) as file:
                rows.append({'message': file.read(), 'type': type})
    return pd.DataFrame(rows)

# read mail and its classification in to data frame(classification Ham=0 & Spam =1)
email_easy_ham_df = readFileContent('./20021010_easy_ham/easy_ham/', 0)
email_hard_ham_df = readFileContent('./20021010_hard_ham/hard_ham/', 0)
email_spam_df = readFileContent('./20021010_spam/spam/', 1)

#Splitting data in to test and train sets
easy_hamtrain,easy_hamtest=train_test_split(email_easy_ham_df, test_size =0.3 ,
    random_state =4)
hard_hamtrain,hard_hamtest=train_test_split(email_hard_ham_df, test_size =0.3 ,
    random_state =4)
hamtrain=pd.concat([easy_hamtrain,hard_hamtrain])
hamtest= pd.concat([easy_hamtest,hard_hamtest])

spamtrain, spamtest= train_test_split(email_spam_df, test_size =0.3 , random_state =4)
```

Listing 1: read mail and its classification in to data frame and Spliting the spam and the ham datasets in a training set and a test set.

Problem 2

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](https://scikit-learn.org/stable/modules/naive$_bayes.html$)).
Test two of these classifiers that are well suited for this problem
- Multinomial Naive Bayes
- Bernoulli Naive Bayes.

```
def naive_bayes(x_train, x_test, y_train, y_test, vectorizer=None):
    #create a Count Vectorizer and fit it to the training set of data.
    if vectorizer==None:
        vectorizer = CountVectorizer()
    vectorizer.fit(x_train)
    x_train_vec = vectorizer.transform(x_train)
    x_test_vec = vectorizer.transform(x_test)

    fig, ((ax1, ax2)) = plt.subplots(1, 2,figsize=(10,10))

    #Train the Multinomial Naive Bayes model with train sets
    mnb = MultinomialNB().fit(x_train_vec, y_train)
    #Predict the values with test sets
    mnb_predict = mnb.predict(x_test_vec)
    tp_mnb, fp_mnb, fn_mnb, tn_mnb = confusion_matrix(y_test,mnb_predict).ravel()
    plot_confusion_matrix(mnb, x_test_vec, y_test, ax=ax1, colorbar=False, normalize='
    true', cmap = 'OrRd')

    #Train the Bernoulli Naive Bayes model with train sets
    bnb = BernoulliNB().fit(x_train_vec, y_train)
    #Predict the values with test sets
    bnb_predict = bnb.predict(x_test_vec)
    tp_bnb, fp_bnb, fn_bnb, tn_bnb  = confusion_matrix(y_test,bnb_predict).ravel()
    plot_confusion_matrix(bnb, x_test_vec, y_test, ax=ax2, colorbar=False, normalize='
    true',cmap = 'OrRd')


    # Declaring labels and title for each subplot
    ax1.set_xlabel('Predicted type of email\n ham=0 & spam=1')
    ax1.set_ylabel('Actual type of email\n ham=0 & spam=1')
    ax1.set_title('Multinomial Naive Bayes\n'+'Accuracy Score: '+str(metrics.
    accuracy_score(y_test, mnb_predict)*100)+'\n'+'tp: '+str((tp_mnb/(tp_mnb+fp_mnb))
    *100)+ ', fn: '+ str((fn_mnb/(tn_mnb+fn_mnb))*100), size=10)
    ax2.set_xlabel('Predicted type of email\n ham=0 & spam=1')
    ax2.set_ylabel('Actual type of email\n ham=0 & spam=1')
    ax2.set_title('Bernoulli Naive Bayes\n'+'Accuracy Score: '+str(metrics.
    accuracy_score(y_test, bnb_predict)*100)+'\n'+'tp: '+str((tp_bnb/(tp_bnb+fp_bnb))
    *100)+ ', fn: '+ str((fn_bnb/(tn_bnb+fn_bnb))*100), size=10)
    plt.subplots_adjust(wspace=0.3)
    plt.show()

    print("Multinomial naive Bayes classifier, Accuracy score:", metrics.
    accuracy_score(y_test, mnb_predict)*100)
    print("Multinomial naive Bayes True Positive rate:", str((tp_mnb/(tp_mnb+fp_mnb))
    *100))
    print("Multinomial naive Bayes False Negative rate:", str((fn_mnb/(tn_mnb+fn_mnb))
    *100))
    print("Multinomial naive Bayes tp_mnb, fp_mnb, fn_mnb, tn_mnb",tp_mnb, fp_mnb,
    fn_mnb, tn_mnb)

    print("Bernoulli naive Bayes classifier, Accuracy score:", metrics.accuracy_score(
    y_test, bnb_predict)*100)
    print("Bernoulli naive Bayes True Positive rate:", str((tp_bnb/(tp_bnb+fp_bnb))
    *100))
    print("Bernoulli naive Bayes False Negative rate:", str((fn_bnb/(tn_bnb+fn_bnb))
    *100))
    print("Bernoulli naive Bayes tp_bnb, fp_bnb, fn_bnb, tn_bnb",tp_bnb, fp_bnb,
    fn_bnb, tn_bnb)

#Concatinating hamtrain & spamtrain
train_df=pd.concat([hamtrain,spamtrain])
#Concatinating hamtest & spamtest
test_df=pd.concat([hamtest,spamtest])
#Assigning data to the variables that will be used to develop the model
x_train = train_df['message']
y_train = train_df['type']
x_test = test_df['message']
y_test = test_df['type']
naive_bayes(x_train, x_test, y_train, y_test)
```

Listing 2: Traning and Testing Multinomial Naive Bayes model and Bernoulli Naive Bayes.

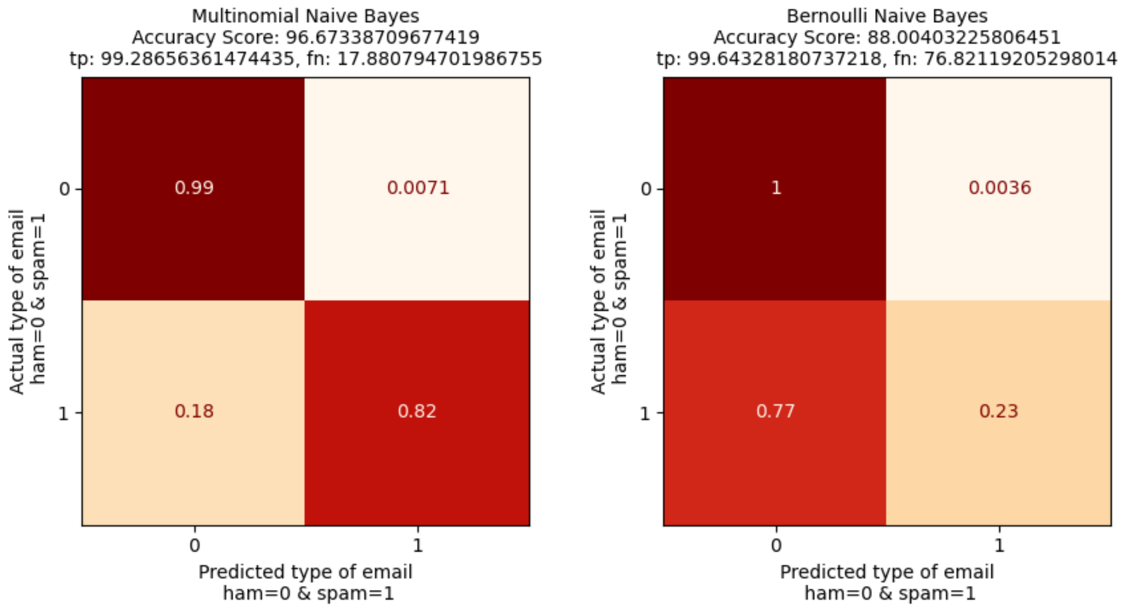

Figure 1: Test reults of Multinomial Naive Bayes model and Bernoulli Naive Bayes uisng confusion matrix

The multinomial classifier takes into account the frequency with which a feature (in our data, the word) occurs, whereas the Bernoulli classifier just takes into account whether the feature (word) occurs or not. In Bernoulli classifier, when more than one element is sent into the classifier during fitting, the value is immediately transformed to 1, indicating that the term appears in the text with out taking frequency in to account. Bernoulli classifier is used when features are binary.

From the observation of above confusion matrices, True positive rate, False negative rate and accuracy scores of Multinomial Naive Bayes and Bernoulli Naive Bayes models. We can say that Multinomial Naive Bayes performed well when compared to Bernoulli Naive Bayes for the given emails data. In the two models Ham data has be classified with more accuracy with only little variation. But the Spam mails are classified well in the Multinomial Naive Bayes, where as in Bernoulli Naive Bayes 77 percent of the spam mails are classified as Ham mails which will impact the user very badly. When we consider total accuracy of the model, the multinomial model classifies data with more accuracy with 3.5 percent of mails, predicted wrong(false negative and false positive). False negative in our problem means mail is spam but predicted as Ham(If user don't recognise mail it spam, then user will be at risk), False positive means Ham but predicted as spam(If mail has some imporatant data but recognised as spam, then user might miss some important data).

From the above diffrences and observation, we can say that classification of the email based on frequency as feature has determined type of mail more acurately. Which means for email classsifaction, Multinomial Naive Bayes model is best suited.

Problem 3

Run your program on
-Spam versus easy-ham
-Spam versus hard-ham.

By reading the question it looks like we need to run Spam versus easy-ham and Spam versus hard-ham data on program created in question2(trained with easy ham, hard ham and spam data) but it was not clearly stated. So considering this we have written the code for Spam versus easy-ham and Spam versus hard-ham using total trained data as well as respective trained data also. This the reason we have splitted the hard ham and easy ham data to train and test sets individually in question1.

```
# Spam versus easy-ham trained with full data and tested with easy ham and spam data
def easyHam_spam_total_train(vectorizer=None):
    test_easyHam_spam_df=pd.concat([easy_hamtest,spamtest])
    x_train = train_df['message']
    y_train = train_df['type']
    x_easyHam_spam_test = test_easyHam_spam_df['message']
    y_easyHam_spam_test = test_easyHam_spam_df['type']
    naive_bayes(x_train, x_easyHam_spam_test, y_train, y_easyHam_spam_test,vectorizer=
    vectorizer)
easyHam_spam_total_train()
```

Listing 3: Spam versus easy-ham trained with full data and test only with easy ham and spam data

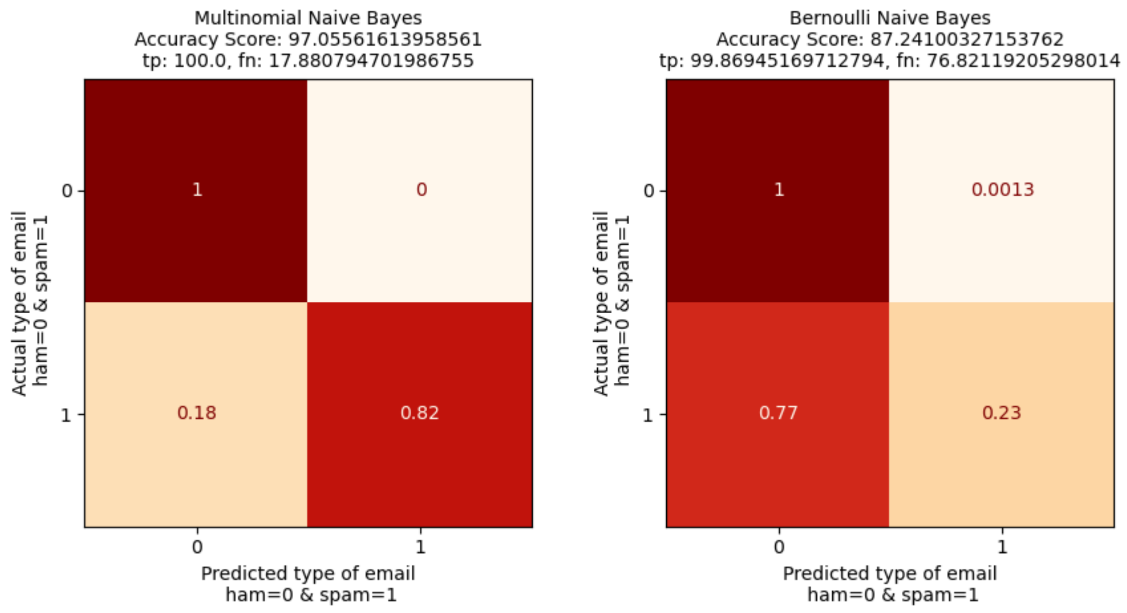

Figure 2: Test result - Spam versus easy-ham trained with full data and test only with easy ham and spam data

```
# Spam versus hard-ham trained with full data and tested only with Hard ham and spam
    data
def hardHam_spam_total_train(vectorizer=None):
    test_hardHam_spam_df=pd.concat([hard_hamtest,spamtest])
    x_train = train_df['message']
    y_train = train_df['type']
    x_hardHam_spam_test = test_hardHam_spam_df['message']
    y_hardHam_spam_test = test_hardHam_spam_df['type']
    naive_bayes(x_train, x_hardHam_spam_test, y_train, y_hardHam_spam_test,vectorizer=
    vectorizer)
hardHam_spam_total_train()
```

Listing 4: Spam versus hard-ham trained with full data and tested with Hard ham and spam data

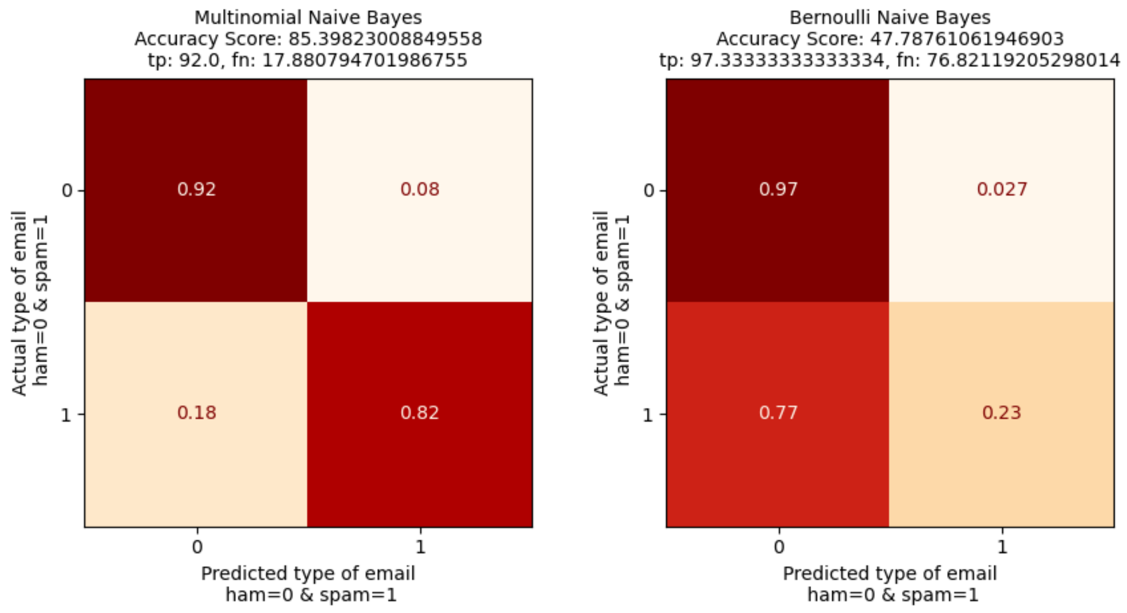

Figure 3: Test result - Spam versus hard-ham trained with full data and test only with hard ham and spam data

For Spam versus easy-ham and Spam versus hard-ham using total trained data:

By observing above confusion matrices, accuracy score of Spam versus easy-ham, is more when compared to the, Spam versus hard-ham, in both the classifications. As it is difficult to classify type(Ham or Spam) between Spam versus hard-ham. But mutinomial performed better than bernoulli with both the Spam versus easy-ham and Spam versus hard-ham data. One more observation we made is that spam mails are classified with same accuracy in both the models when tested with Spam versus easy-ham and Spam versus hard-ham respectively. Even though test data different, this occur because as both the classifications are done, with the model trained on same data and number of spam mails used are same.

```python
# Spam versus easy-ham trained and tested with easy ham and spam data
def easyHam_spam_train(vectorizer=None):
    train_easyHam_spam_df=pd.concat([easy_hamtrain,spamtrain])
    test_easyHam_spam_df=pd.concat([easy_hamtest,spamtest])
    x_easyHam_spam_train = train_easyHam_spam_df['message']
    y_easyHam_spam_train = train_easyHam_spam_df['type']
    x_easyHam_spam_test = test_easyHam_spam_df['message']
    y_easyHam_spam_test = test_easyHam_spam_df['type']
    naive_bayes(x_easyHam_spam_train, x_easyHam_spam_test, y_easyHam_spam_train,
    y_easyHam_spam_test,vectorizer=vectorizer)
easyHam_spam_train()
```

Listing 5: Spam versus easy-ham trained and tested with easy ham and spam data

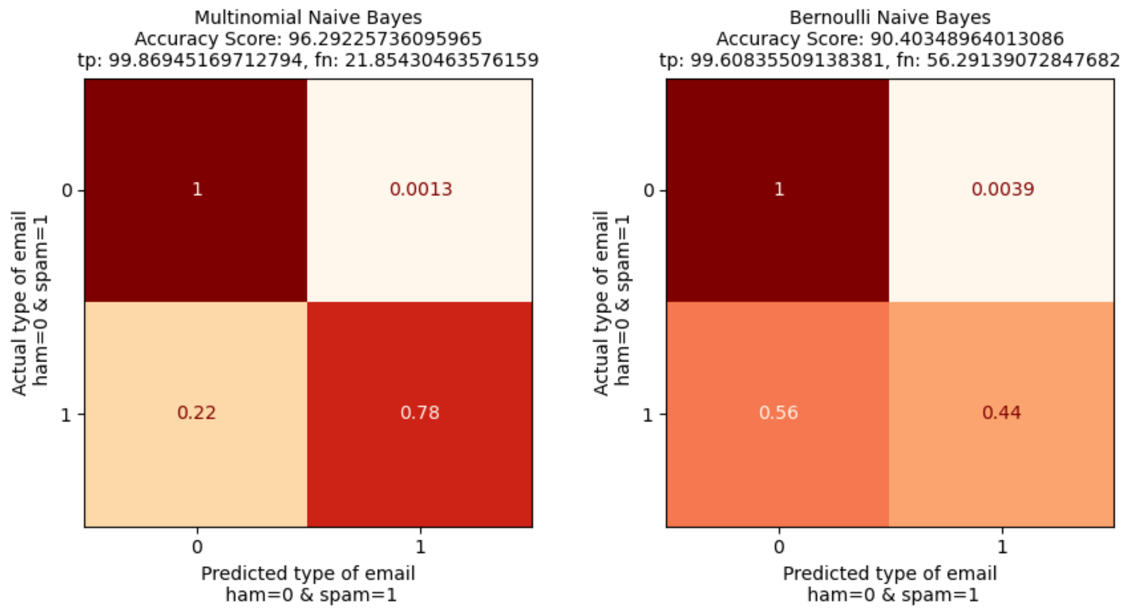

Figure 4: Test result - Spam versus easy-ham trained and tested with easy ham and spam data

```
# Spam versus hard-ham trained and tested with Hard ham and spam data
def hardHam_spam_train(vectorizer=None):
    train_hardHam_spam_df=pd.concat([hard_hamtrain,spamtrain])
    test_hardHam_spam_df=pd.concat([hard_hamtest,spamtest])
    x_hardHam_spam_train = train_hardHam_spam_df['message']
    y_hardHam_spam_train = train_hardHam_spam_df['type']
    x_hardHam_spam_test = test_hardHam_spam_df['message']
    y_hardHam_spam_test = test_hardHam_spam_df['type']
    print(x_hardHam_spam_train.shape,y_hardHam_spam_train.shape,x_hardHam_spam_test.
    shape,y_hardHam_spam_test.shape)
    naive_bayes(x_hardHam_spam_train, x_hardHam_spam_test, y_hardHam_spam_train,
    y_hardHam_spam_test,vectorizer=vectorizer)
hardHam_spam_train()
```

Listing 6: Spam versus hard-ham trained and tested with Hard ham and spam data

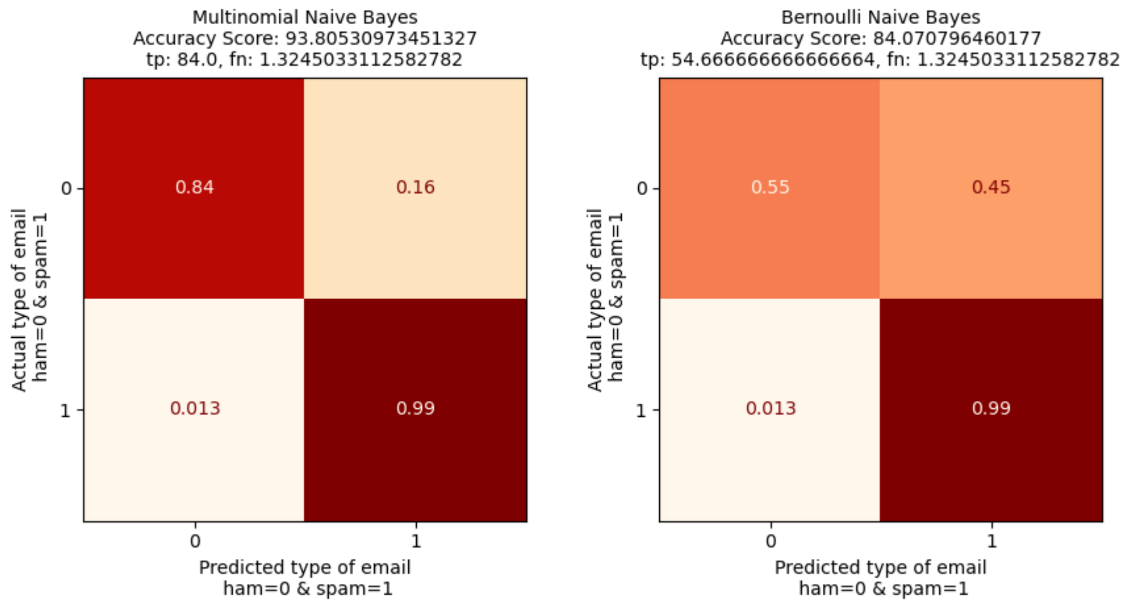

Figure 5: Test result - Spam versus hard-ham trained and tested with Hard ham and spam data

For Spam versus easy-ham and Spam versus hard-ham using respective trained data:

By observing above confusion matrices, accuracy score of Spam versus easy-ham, is more when compared to the, Spam versus hard-ham, in both the classifications. As it is difficult to classify type(Ham or Spam) between Spam versus hard-ham. But mutlinomial perfromed better than bernoulli in both the Spam versus easy-ham and Spam versus hard-ham data. Which is same as when model trained with, complete trained data. But when we compared these accuracy with the accuracy obtained, when model trained with complete trained data. The accuracy obtained when model trained with only resective data is higher in all the cases. Which means the model trained on the extra data leads to decrease in performance classification.

Problem 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.

```
#code to Find common/uncommon words in data set
ham_df = pd.concat([email_easy_ham_df, email_hard_ham_df,email_spam_df])
countHam = Counter(" ".join(ham_df["message"]).split()).most_common()
dataHam = pd.DataFrame.from_dict(countHam)
dataHam_words=dataHam[0]
print(dataHam_words.head(30).to_numpy())
print(dataHam_words.tail(30).to_numpy())
```

Listing 7: Find common/uncommon words in data set

Output:

Most Common Words ['the' '**2002**' 'to' '¿' 'for' 'with' 'from' 'by' 'of' 'and' 'Received:' 'a' 'id' 'Sep' 'in' 'is' 'ESMTP' '+0100' 'that' 'I' '¡td' 'you' 'Aug' 'localhost' 'on' 'Oct' 'be' 'it' '=' '[**127.0.0.1**])']

Least Common Words ['65.00).''*XBR*''*NORTH*''**180.00' 'SOUTH'** '190.00''**280.'** '70)''**265.'** '95)''*E*)''*WET*''**145.00'** '205.''**60)' 'F)' 'G)'** '110.00''**25)' 'H)' 'VISIONARY'** '310.00''**615.'** '305)''1− 623 − 972 − 5999''*Hours* :' '*Mon.*''*Mail.*''*convenience.*''*mailto* : *bm7@btamail.net.cn?subject* = *Remove*']

The finding the common/uncommon words in the dataset helps to train the model with data of words which are good indicators, to classify between the spam or ham. The common words such as com, from etc.. can be found most commonly in both the spam and ham emails, which are uninformative to classify.

```
#Creating word count vectorizer with words that appear more than 90% of the time,
    words that appear in just one email, and common English words
vectorizer = CountVectorizer(max_df = 0.90, min_df = 2, stop_words="english")
easyHam_spam_total_train(vectorizer=vectorizer)
easyHam_spam_train(vectorizer=vectorizer)
```

Listing 8: Spam versus Easy-ham with count vectorizer Filter

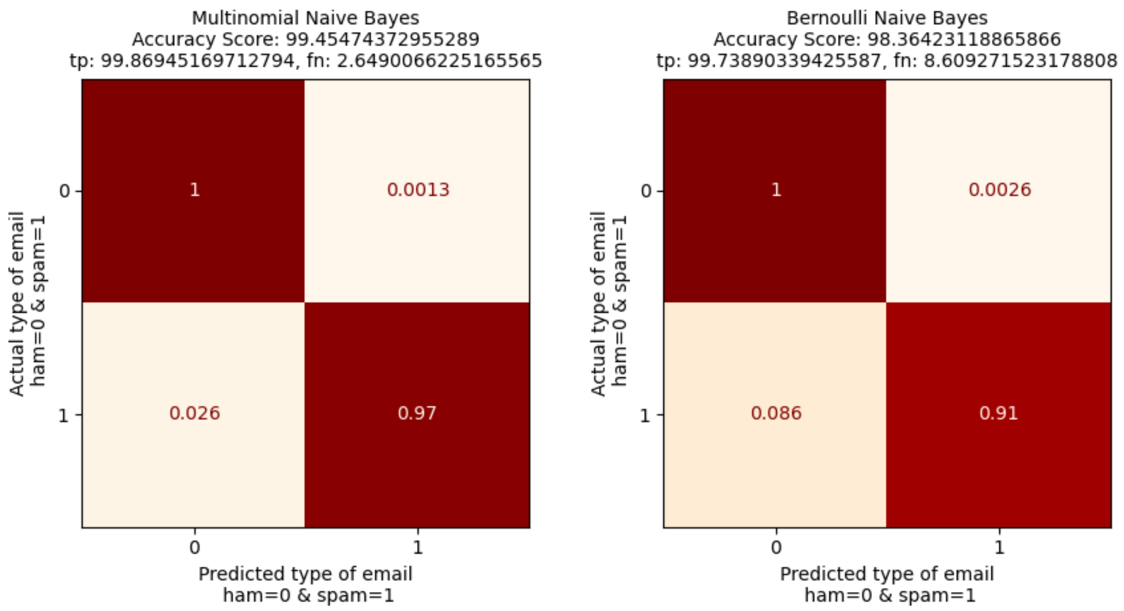

Figure 6: Test result - Spam versus Easy-ham with count vectorizer Filter using full train data

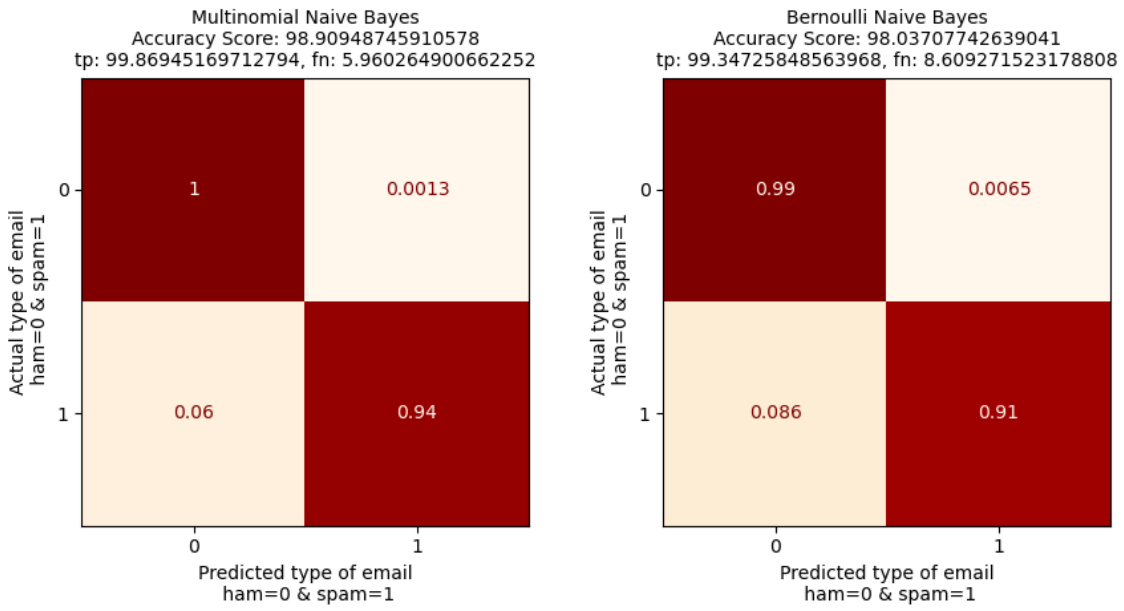

Figure 7: Test result - Spam versus Easy-ham with count vectorizer Filter using respective train data

```
#Creating word count vectorizer with words that appear more than 90% of the time,
    words that appear in just one email, and common English words
vectorizer = CountVectorizer(max_df = 0.90, min_df = 2, stop_words="english")
hardHam_spam_total_train(vectorizer=vectorizer)
hardHam_spam_train(vectorizer=vectorizer)
```

Listing 9: Spam versus Hard-ham with count vectorizer Filter

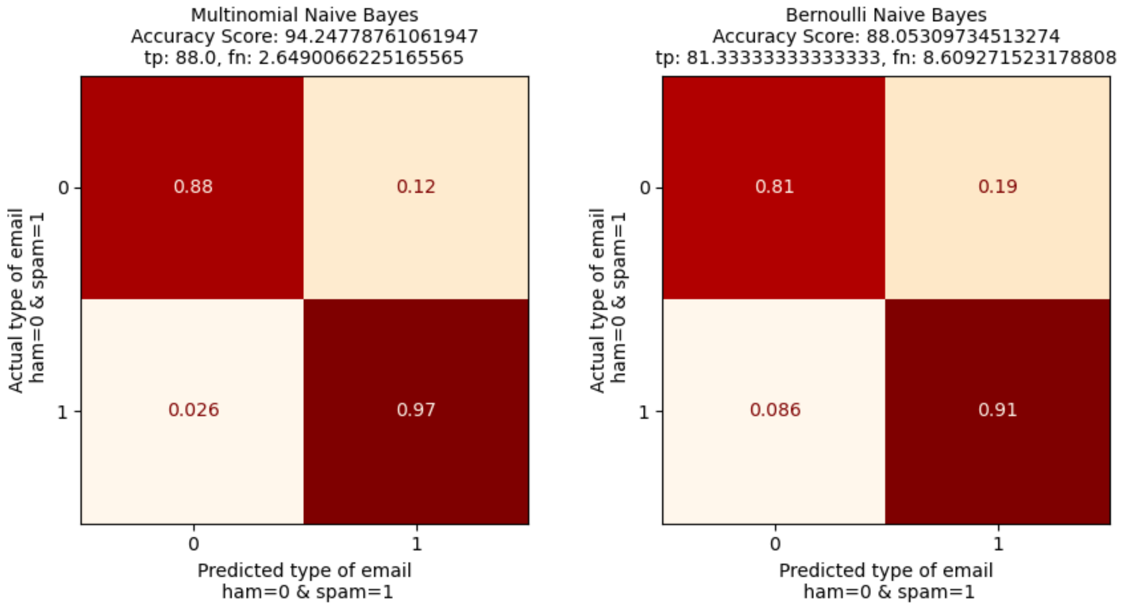

Figure 8: Test result - Spam versus Hard-ham with count vectorizer Filter using full train data

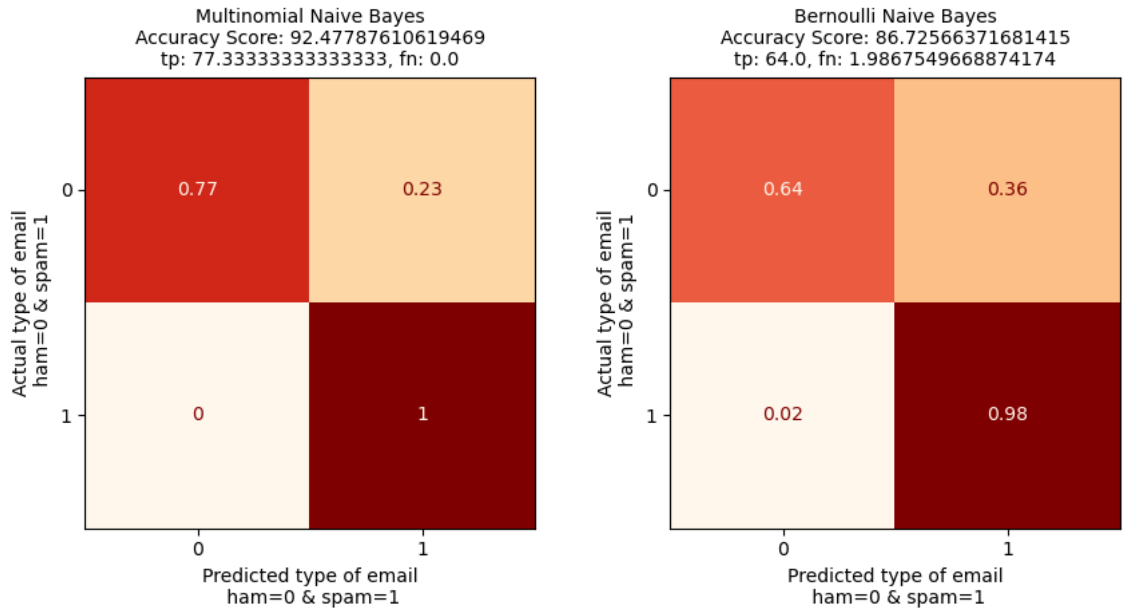

Figure 9: Test result - Spam versus Hard-ham with count vectorizer Filter using respective train data

As mentioned in the question, there are two ways to filter out these words. One approach is to use the extracted lists of most common/uncommon words of ham/spam emails. Another approach is to filter out common words using the built-in stop word list for English.

However, by looking at the data, we can see that the spam and ham data sets are imbalanced in size. If we build our model to ignore the words that are the most common or uncommon, it's possible that we will also delete words that make it clear whether an email is spam or ham. Choosing how many words should be considered common or uncommon and stopping them is another challenging decision.

Therefore, we have decided to employ the CountVectorizer method while utilizing stop words and built-in arguments. In this method, we've given the CountVectorizer three arguments: the first is "stop words = 'english'" which will automatically filter out many of the most popular English words; the second is "max df=0.90" which will force the model to avoid using the top 10 percent of most popular words for classification; and the third argument is "min df=2" which will cause all words that appear in fewer than two emails—that is, if the word appears in