Assignment 4: Spam classification using Naïve Bayes

Group-67-Yaochen Rao-Wanqiu Wang-20hrs each

September 27, 2022

```
# 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
```

```
HTTP request sent, awaiting response... 200 OK
Length: 1677144 (1.6M) [application/x-bzip2]
Saving to: ‘20021010_easy_ham.tar.bz2.6’

20021010_easy_ham.t 100%[===================>]   1.60M  --.-KB/s    in 0.006s

2022-09-27 20:06:40 (278 MB/s) - ‘20021010_easy_ham.tar.bz2.6’ saved [1677144/1677144]

--2022-09-27 20:06:40--  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.bz2.6’

20021010_hard_ham.t 100%[===================>] 997.19K  --.-KB/s    in 0.005s

2022-09-27 20:06:40 (205 MB/s) - ‘20021010_hard_ham.tar.bz2.6’ saved [1021126/1021126]

--2022-09-27 20:06:40--  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.bz2.6’

20021010_spam.tar.b 100%[===================>]   1.14M  --.-KB/s    in 0.006s

2022-09-27 20:06:40 (204 MB/s) - ‘20021010_spam.tar.bz2.6’ saved [1192582/1192582]
```

```
!ls -lah
```

```
total 27M
drwxrwxrwx 5 root root   26 Sep 27 20:06 .
drwxr-xr-x 1 root root  211 Sep 27 19:12 ..
-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 1.6M Jun 29  2004 20021010_easy_ham.tar.bz2.6
-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 998K Dec 16  2004 20021010_hard_ham.tar.bz2.6
-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
-rw-r--r-- 1 root root 1.2M Jun 29  2004 20021010_spam.tar.bz2.6
drwx--x--x 2  500  500 2.5K Oct 10  2002 easy_ham
drwx--x--x 2 1000 1000  252 Dec 16  2004 hard_ham
drwxr-xr-x 2  500  500  503 Oct 10  2002 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`)

```python
# Pre-processing code here
#Import package
import os
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.core.frame import DataFrame
plt.style.use('fivethirtyeight')
sns.set()
```

```
def get_file(dir):
  # Get all the file and directory names in directory and return an array
  all_files = os.listdir(os.path.abspath(dir))
  files = []
  try:
    for file_name in all_files:
      files.append(open(dir + file_name, "r", errors='ignore').read())
  except:
    print("ERROR:Invalide file from {}".format(dir))
  print('{} files found in {}'.format(len(files), dir))
  return files

# Read the file and put in list of emails
easy_ham = get_file("easy_ham/")
hard_ham = get_file("hard_ham/")
spam = get_file("spam/")
```

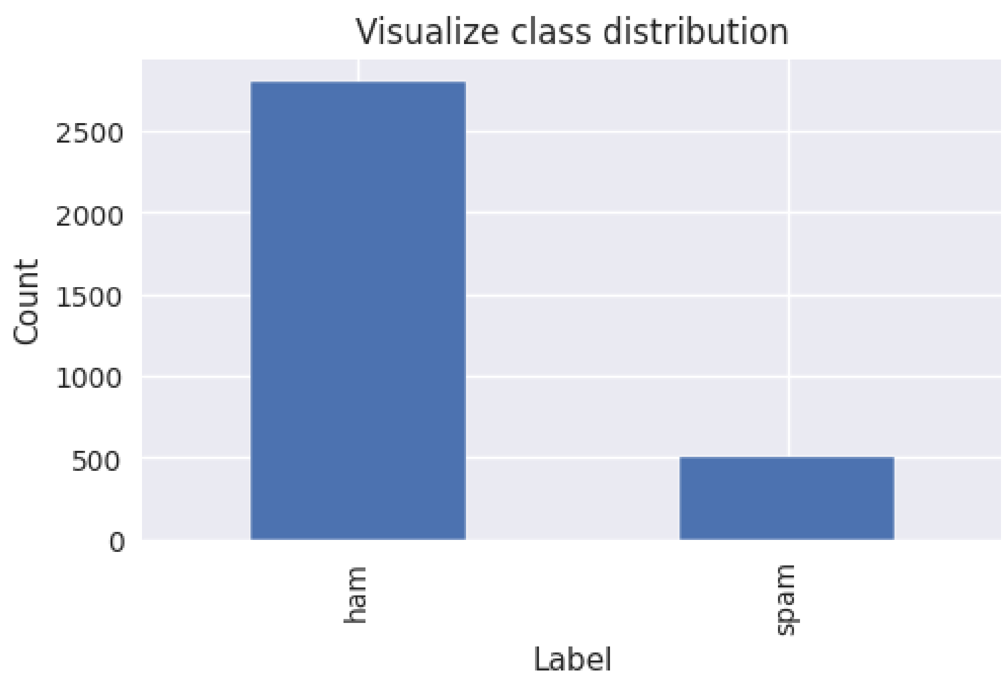
```
2551 files found in easy_ham/
250 files found in hard_ham/
501 files found in spam/
```

```
# Create dataframe for ham emails(easy ham and hard ham)
df_easy = pd.DataFrame(easy_ham)
df_hard = pd.DataFrame(hard_ham)
# Create dataframe for spam emails
df_spam = pd.DataFrame(spam)
```

```
# Create label to each email
df_easy['label'] = 'easy'
df_hard['label'] = 'hard'
df_spam['label'] = 'spam'
df_easy.columns = ['email_message','label']
df_hard.columns = ['email_message','label']
df_spam.columns = ['email_message','label']
# Create dataFrames for
# all ham(easy and hard), easy ham and spam hard ham and spam, all email
df_ham = df_easy.append(df_hard,ignore_index=True)
df_ham['label'] = 'ham'
df_easy_spam = df_easy.append(df_spam,ignore_index=True)
df_hard_spam =df_hard.append(df_spam,ignore_index=True)
df_all = df_ham.append(df_spam,ignore_index=True)
```

```
# Visualize distribution of ham email and spam email
plt.style.use('seaborn')
plt.figure(figsize=(5,3), dpi=100)
plt.title('Visualize class distribution')
df_all['label'].value_counts().plot(kind='bar')
plt.xlabel('Label')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



```python
# Split datasets in a training set and a test set.
hamtrain, hamtest, spamtrain, spamtest = train_test_split(
df_all['email_message'],df_all['label'], test_size=0.25, random_state=15)
easytrain, easytest, easy_spamtrain, easy_spamtest = train_test_split(
    df_easy_spam['email_message'], df_easy_spam['label'], test_size=0.25,
    random_state=15)
hardtrain, hardtest, hard_spamtrain, hard_spamtest = train_test_split(
    df_hard_spam['email_message'], df_hard_spam['label'], test_size=0.25,
    random_state=15)
```

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.

Answer:
When predicting, Multinomial Naive Bayes only considers the words that appear in the predicted text, ignoring the features that do not appear. And Multinomial Naive Bayes takes into account the frequency

of words. But Bernoulli Naive Bayes also considers words that don't appear. And Bernoulli Naive Bayes considers binary, just to see if the word exists. It can be used in cases where word frequency is not so important, and it is also suitable for problems with large data sets.

We prefer to only consider the words that appear, because the number of core words in a category will not be only a few, and a text, especially a short text, cannot appear all the core words of the category, and these core words that do not appear are bound to appear. It will reduce the predicted probability value, and it is easy to make the predicted probability values for various purposes have a small difference.

As in this question, not only the existence of the word but also its frequency should be considered. Some words that appear a few times may be ham emails, but more than a few times are spam emails.

Comparing the True Positive Rate and False Negative Rate of the two models at the same time, it can also be found that the True Positive Rate of the Multinomial Naive Bayes model is higher and the False Negative Rate is lower i.e. important emails are not classified as spam. But Bernoulli Naive Bayes is the opposite, so the prediction accuracy is much lower than that of Multinomial Naive Bayes (in the ham and spam datasets).

So in contrast, in this dataset, Multinomial Naive Bayes can handle text classification better because he has few misclassifications. Bernoulli Naive Bayes may perform better on some datasets, especially those that don't need to care about word frequency.

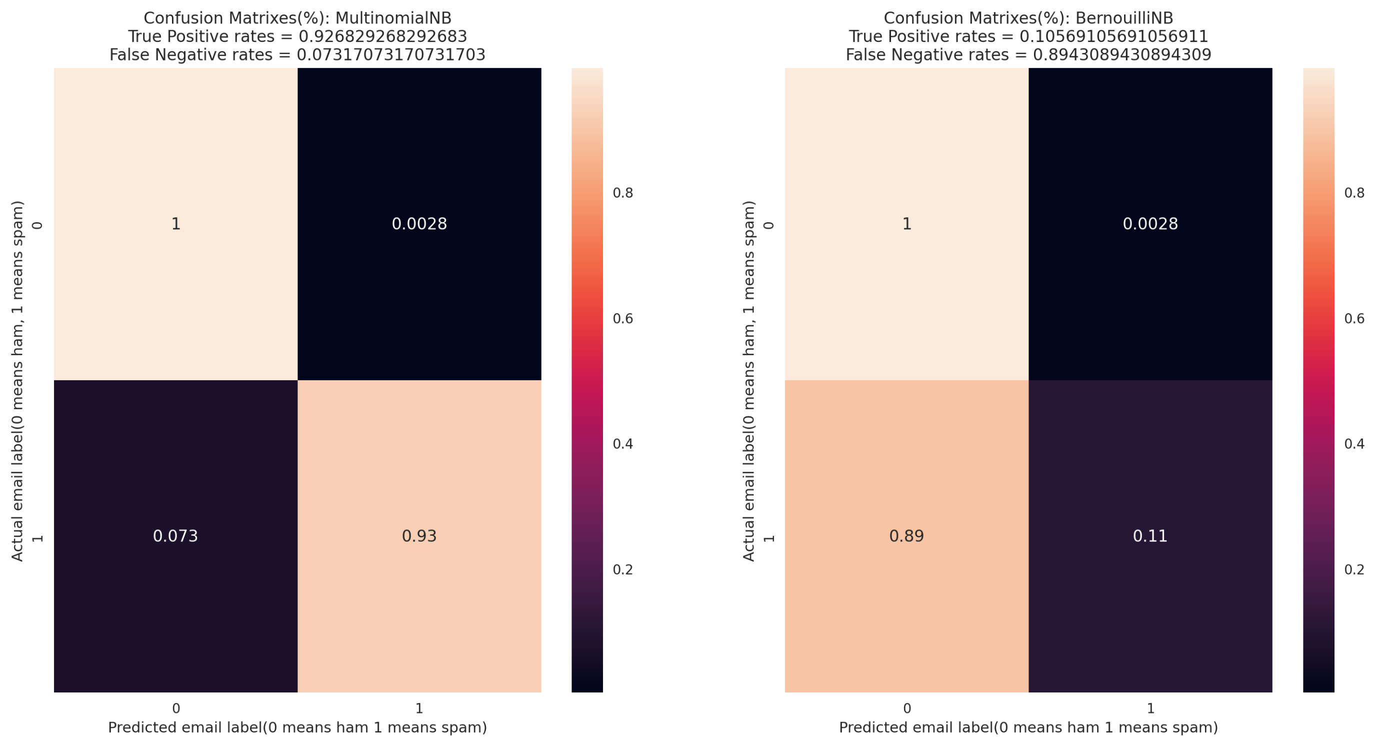
From the heatmap comparison we can see that the Bernoulli model has a probability of 0.88 to classify spam as ham, which is very confusing in real life and means that the user's inbox will often receive spam email. Moreover, user may get a poor experience and may miss important emails due to too much spam.

```python
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn import metrics
from sklearn.metrics import confusion_matrix,accuracy_score
```

```python
# Create a function to compare Multinomial Naive Bayes and Bernoulli Naive Bayes
def Naive_Bayes_MNB_BNB(x_train,x_test,y_train,y_test,vectorizer=None,
    fit_prior = None):
  # Transform data into word feature vector
  if vectorizer is None:
    vectorizer = CountVectorizer()
  x_train_fit = vectorizer.fit_transform(x_train)
  x_test_fit = vectorizer.transform(x_test)

  if fit_prior is None:
    # Multinomial Naive Bayes and calcuate accuracy scores
    mnb = MultinomialNB().fit(x_train_fit,y_train)
    y_pred_mnb = mnb.predict(x_test_fit)
    print("Multinomial Naive Bayes: The total points: %d, The number of mislabeled points:
                    (x_test_fit.shape[0], (y_test != y_pred_mnb).sum()))
    print("Multinomial Naive Bayes: The accuracy score is %f"% accuracy_score(y_test, y_pre
  else:
    # Multinomial Naive Bayes and calcuate accuracy scores
    mnb = MultinomialNB(fit_prior=fit_prior).fit(x_train_fit,y_train)
    y_pred_mnb = mnb.predict(x_test_fit)
    print("Multinomial Naive Bayes(fit_prior): The total points: %d, The number of mislabel
                    (x_test_fit.shape[0], (y_test != y_pred_mnb).sum()))
    print("Multinomial Naive Bayes(fit_prior): The accuracy score is %f"% accuracy_score(y_

  if fit_prior is None:
    # Bernoulli Naive Bayes and calcuate accuracy scores
    bnb = BernoulliNB(binarize = True).fit(x_train_fit,y_train)
    y_pred_bnb = bnb.predict(x_test_fit)
    print("Bernoulli Naive Bayes: The total points: %d, The number of mislabeled points: %d
                    (x_test_fit.shape[0], (y_test != y_pred_bnb).sum()))
    print("Bernoulli Naive Bayes: The accuracy score is %f"% accuracy_score(y_test, y_pred_
  else:
    # Bernoulli Naive Bayes and calcuate accuracy scores
    bnb = BernoulliNB(binarize = True, fit_prior=fit_prior).fit(x_train_fit,y_train)
    y_pred_bnb = bnb.predict(x_test_fit)
    print("Bernoulli Naive Bayes(fit_prior): The total points: %d, The number of mislabeled
                    (x_test_fit.shape[0], (y_test != y_pred_bnb).sum()))
    print("Bernoulli Naive Bayes(fit_prior): The accuracy score is %f"% accuracy_score(y_te
  # Create confusion matrixes
  cm_mnb = confusion_matrix(y_test,y_pred_mnb)
  cm_bnb = confusion_matrix(y_test,y_pred_bnb)
  # Normalized
  norm_mnb = cm_mnb.astype('float') / cm_mnb.sum(axis=1)[:, np.newaxis]
  norm_bnb = cm_bnb.astype('float') / cm_bnb.sum(axis=1)[:, np.newaxis]

  # Calcuate true positive(tp) and false negative(fn)
  tn_mnb, fp_mnb, fn_mnb, tp_mnb =cm_mnb.ravel()
  tn_bnb, fp_bnb, fn_bnb, tp_bnb =cm_bnb.ravel()
  # TPR = TP / (TP + FN) and FNR = FN /（TP + FN） (it means FPR+FNR=1)
  tpr_mnb = tp_mnb / (tp_mnb+fn_mnb)
  tpr_bnb = tp_bnb / (tp_bnb+fn_bnb)
  fnr_mnb = 1-tpr_mnb
  fnr_bnb = 1-tpr_bnb

  # Plot confusion matrixes & True Positive and False Negative rates
  fig, [ax1, ax2] = plt.subplots(1, 2, figsize=(16,8), dpi=200)
  ax1 = sns.heatmap(norm_mnb, annot=True, ax=ax1)
  ax1.title.set_text('Confusion Matrixes(%): MultinomialNB\nTrue Positive rates = {0}\nFals
  ax1.set_xlabel("Predicted email label(0 means ham 1 means spam)")
  ax1.set_ylabel("Actual email label(0 means ham, 1 means spam)")
  ax2 = sns.heatmap(norm_bnb, annot=True, ax=ax2)
  ax2.title.set_text('Confusion Matrixes(%): BernouilliNB\nTrue Positive rates = {0}\nFalse
  ax2.set_xlabel("Predicted email label(0 means ham 1 means spam)")
  ax2.set_ylabel("Actual email label(0 means ham, 1 means spam)")
```

```
# Uses four datasets (hamtrain, spamtrain, hamtest, and spamtest)
print("Ham vs Spam: \n")
Naive_Bayes_MNB_BNB(hamtrain,hamtest,spamtrain,spamtest,vectorizer=None)
```

```
Ham vs Spam:


Multinomial Naive Bayes: The total points: 826, The number of mislabeled points: 11
Multinomial Naive Bayes: The accuracy score is 0.986683


Bernoulli Naive Bayes: The total points: 826, The number of mislabeled points: 113
Bernoulli Naive Bayes: The total points: 826, The number of mislabeled points: 112
Bernoulli Naive Bayes: The accuracy score is 0.864407
```

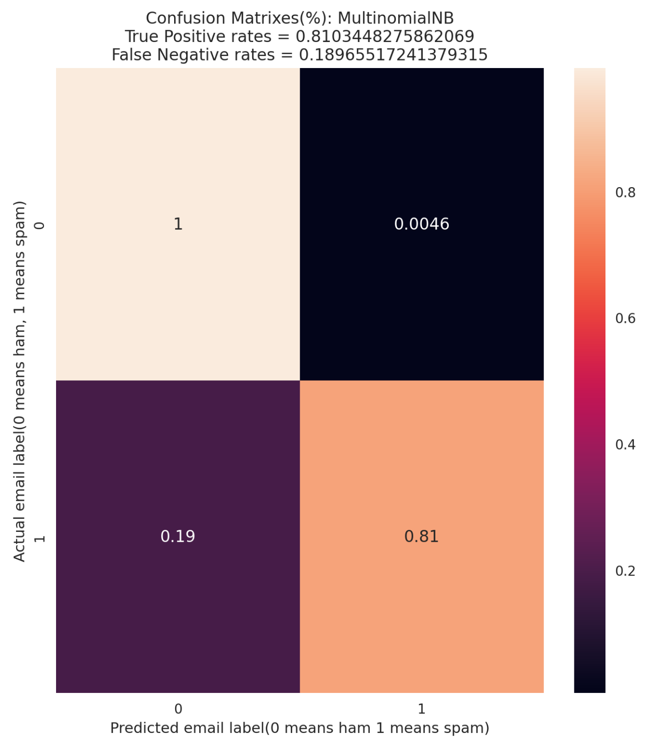


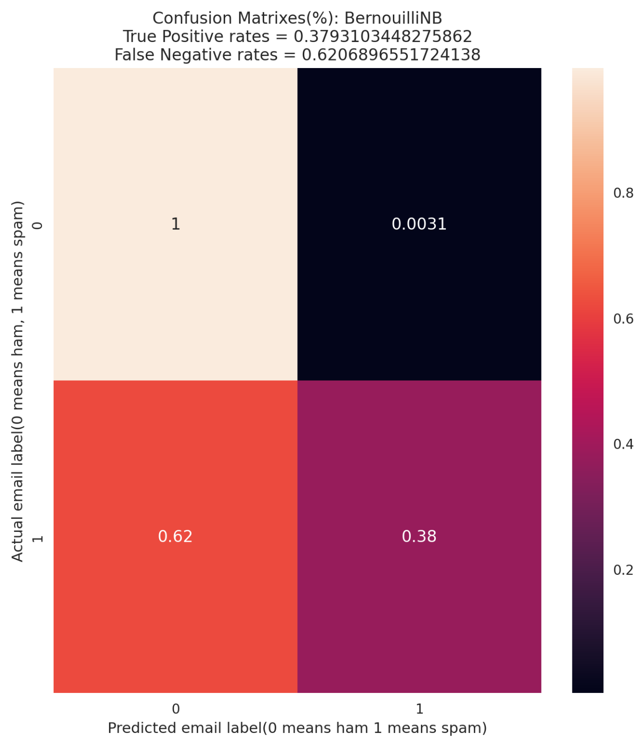
3.Run your program on

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

```python
# Spam versus easy-ham
# Uses four datasets (easytrain, easy_spamtrain, easytest,
# and easy_spamtest)
print("Easy ham vs Spam: \n")
Naive_Bayes_MNB_BNB(easytrain,easytest,easy_spamtrain,easy_spamtest,
                    vectorizer=None)
```

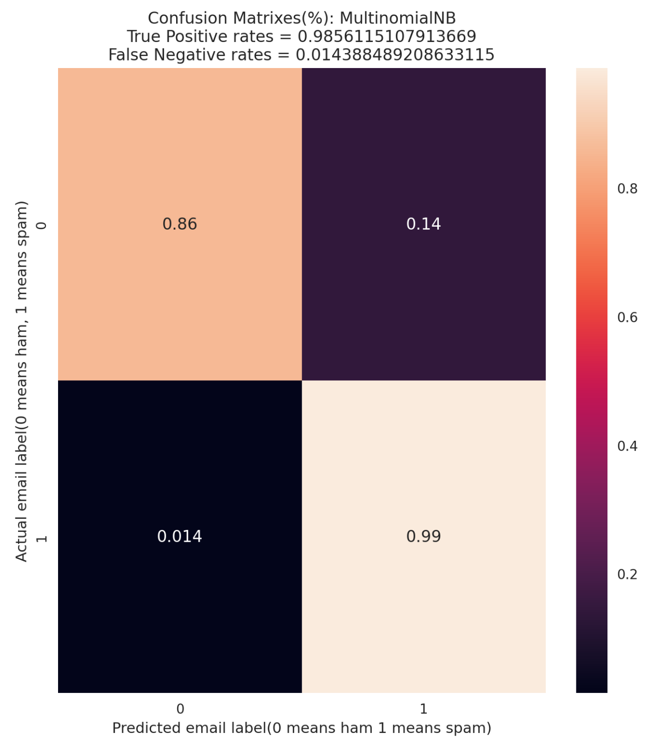
```
Easy ham vs Spam:


Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 25
Multinomial Naive Bayes: The accuracy score is 0.967235
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 74
Bernoulli Naive Bayes: The accuracy score is 0.903014
```

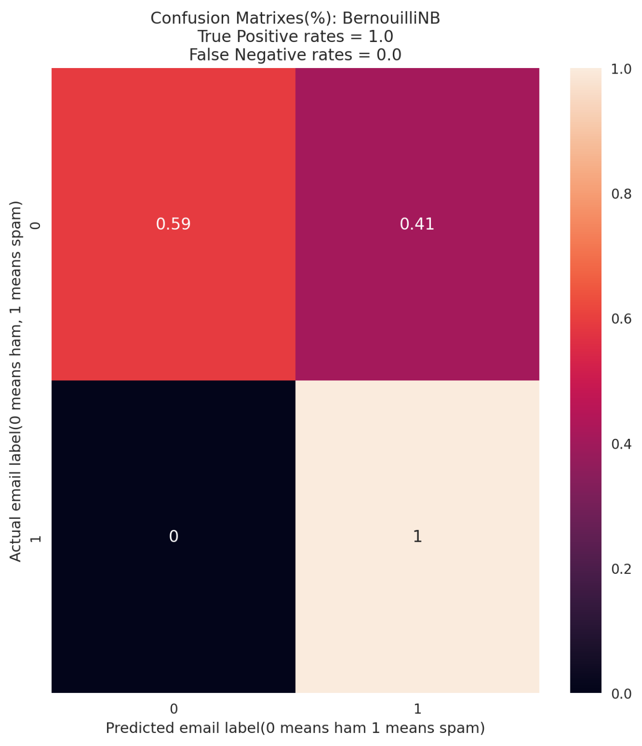




```python
# Spam versus hard-ham.
# Uses four datasets (hardtrain, hard_spamtrain, hardtest,
# and hard_spamtest)
print("Hard ham vs Spam: \n")
Naive_Bayes_MNB_BNB(hardtrain,hardtest,hard_spamtrain,hard_spamtest,
                    vectorizer=None)
```

```
Hard ham vs Spam:

Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 9
Multinomial Naive Bayes: The accuracy score is 0.952128
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 20
Bernoulli Naive Bayes: The accuracy score is 0.893617
```





It can be found that both Multinomial Naive Bayes and Bernoulli Naive Bayes have higher true positives when only hard-ham is considered. But when considering only easy-ham, a distinction emerges between Multinomial Naive Bayes and Bernoulli Naive Bayes, with Multinomial Naive Bayes having higher true positive rates than Bernoulli Naive Bayes, and Bernoulli Naive Bayes having higher false negative rates than its own true positives.

Looking at the heatmap comparison of easy_ham first, we can see that FN and TP are very different in the two models. The FN of Multinomial is smaller than Bernoulli, which means that Multinomial has smaller probability to judge the email as normal email when it is spam. Meanwhile, the TP of the Multinomial is larger than Bernoulli, which means that the Multinomial has a higher probability of judging an email as spam when it is indeed a spam. Since we do not want spam to fill our inbox, which not only brings us bad user experience but also may make us miss important information, we conclude that Multinomial performs better than Bernoulli on easy_ham.

For the comparison of the performance of multinomial and Bernoulli on hard_ham, we can see that the difference between the FP of multinomial and Bernoulli is large, and the FP of Bernoulli is larger than that of multinomial, which means that the Bernoulli model has a greater probability of classifying normal messages into spam, which is a very bad result for us because we do not want to miss any normal message, so we conclude that the performance of multinomial on hard_ham is also better than that of Bernoulli.

And when we compare the performance of the model itself on easy_ham and hard_ham, we can see that the performance of both multinomial and Bernoulli decreases, where Bernoulli's confusion matrix shows a large change, and this change is especially reflected in the FN and TP. Bernoulli's FN decreases significantly while TP increases significantly, indicating that Bernoulli's classification of spam on the hard_ham dataset has a large error. The reason for this error is likely because when the model is trained on easy_ham and spam, the amount of data in easy_ham is much larger than that of spam, so the model can identify normal emails better. And when the model is trained on hard_ham and spam, the amount of data of spam is larger than that of normal emails, so the model can identify spam better.

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.

Answer a:
Look at Multinomial Naive Bayes first. If the stop word removal operation is performed, the probability of the core word will be increased. But for Bernoulli Naive Bayes, the removal of stop words basically does not affect the probability of core words.

To sum up, if you remove stop words, then for Multinomial Naive Bayes, it will improve the probability estimates of core words, which means that the model will be more accurate. Only if the stop words are classified to Said it didn't work. Or even if your stop words are inaccurate, as long as there are words in the sentence that are useful for classification, the prediction results will not have much impact.
But for Bernoulli Naive Bayes, if the stop word is removed, the word will be regarded as not appearing. If the word is a word that has a role in classification, it will either increase the predicted probability value, or will Decrease the predicted probability value.

In short, some very repetitive words, such as 'the', 'to', etc., are not useful for classification. They don't help us distinguish spam email from ham email. Therefore, we can view these words as noise, as they will not contribute to our analysis.
In addition to that, some words that appear only once will not contribute to the classification. But removing this probably won't affect the classification accuracy either. Because words that appear only once should have little effect on the classifier before. Removing words that occur only once will help determine what words are really helpful for classification.

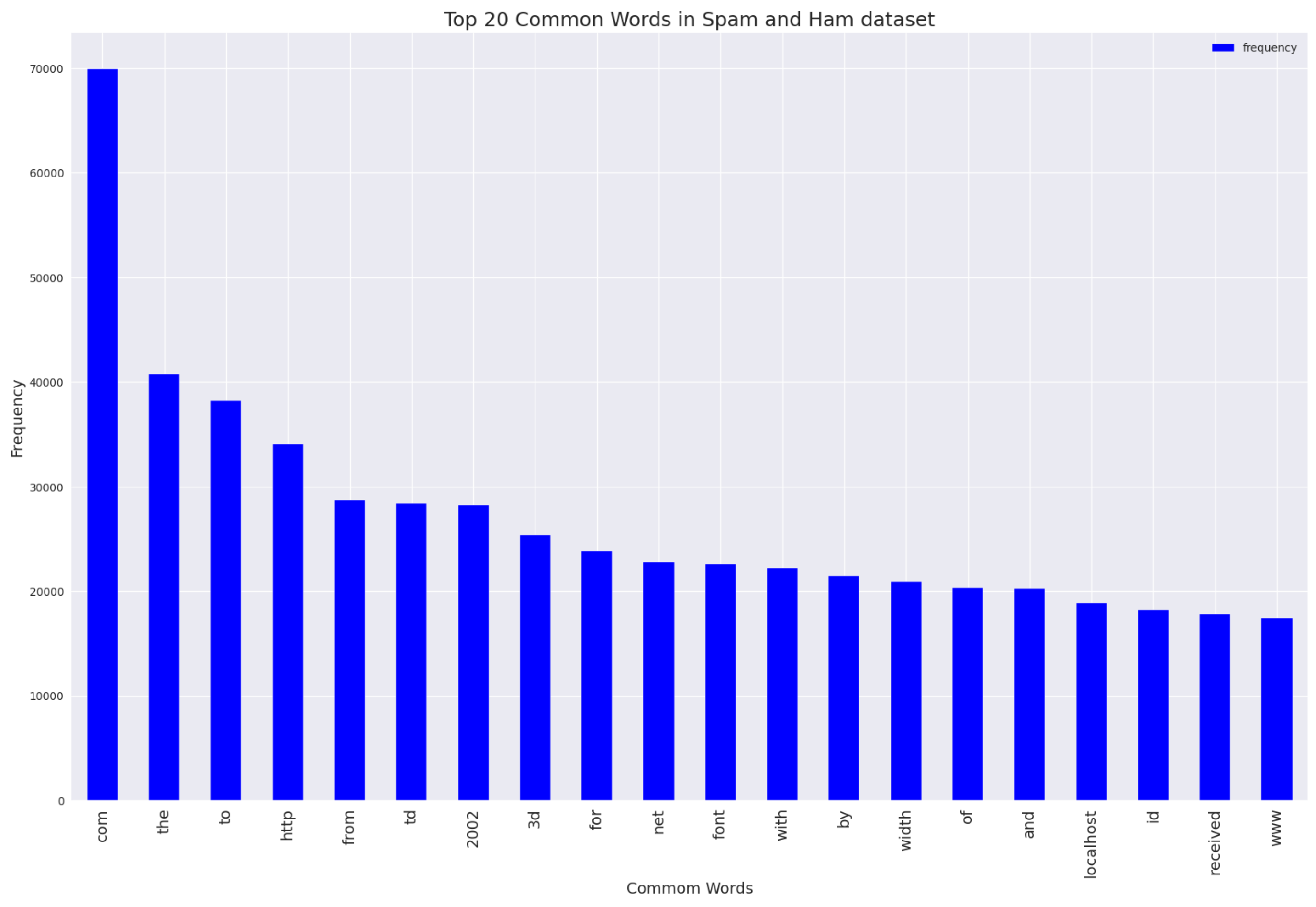
```
# Create function to find too common words
def get_common_words(df, n=None):

  # Transform data into feature vector
  cv = CountVectorizer().fit(df)
  words_vector = cv.transform(df)
  # Count occurances
  words_sums = words_vector.sum(axis=0)
  frequency = [(word, words_sums[0, idx]) for word, idx
               in cv.vocabulary_.items()]
  frequency = sorted(frequency, key = lambda word: word[1], reverse=True)
  return frequency[:n]

# Create function to find too uncommon words
```

```python
def get_uncommon_words(df, n=None):
  i=0
  # Transform data into feature vector
  cv = CountVectorizer().fit(df)
  words_vector = cv.transform(df)
  # Count occurances
  words_sums = words_vector.sum(axis=0)
  frequency = [(word, words_sums[0, idx]) for word, idx
               in cv.vocabulary_.items()]
  frequency = sorted(frequency, key = lambda word: word[1], reverse=False)
  for word in frequency:
    if word[1] <= 1:
      i+=1
  print("The number of words that appear only once: ",i)
  return frequency[:n]

Too_common_words = get_common_words(df_all['email_message'],20)
Too_uncommon_words = get_uncommon_words(df_all['email_message'],65281)
# Show data
print("The top 20 common words is: ",Too_common_words)
print("The uncommon words is: ",Too_uncommon_words)
```

```
mb250pjwvzgl2pg0kicagicagicagicagpc90zd4n', 1), ('icagidx0cj4gdqogicagicagicagica8dgqgd2lkdgg9ijiynyigdmf
```

```python
Top_common_words = DataFrame(Too_common_words)
Top_common_words.rename(columns={0:'top_words',1:'frequency'},inplace=True)
Top_uncommon_words = DataFrame(Too_uncommon_words)
Top_uncommon_words.rename(columns={0:'top_words',1:'frequency'},inplace=True)

#Visualize top common words and top uncommon words
Top_common_words.plot.bar(color = 'blue', figsize = (18,12))
y_axis = np.arange(len(Top_common_words['top_words']))
plt.xticks(y_axis, Top_common_words['top_words'], fontsize=14)
plt.xlabel('Commom Words', fontsize=14)
plt.ylabel('Frequency' ,fontsize=14)
plt.title('Top 20 Common Words in Spam and Ham dataset', fontsize=18)
```

```
Text(0.5, 1.0, 'Top 20 Common Words in Spam and Ham dataset')
```



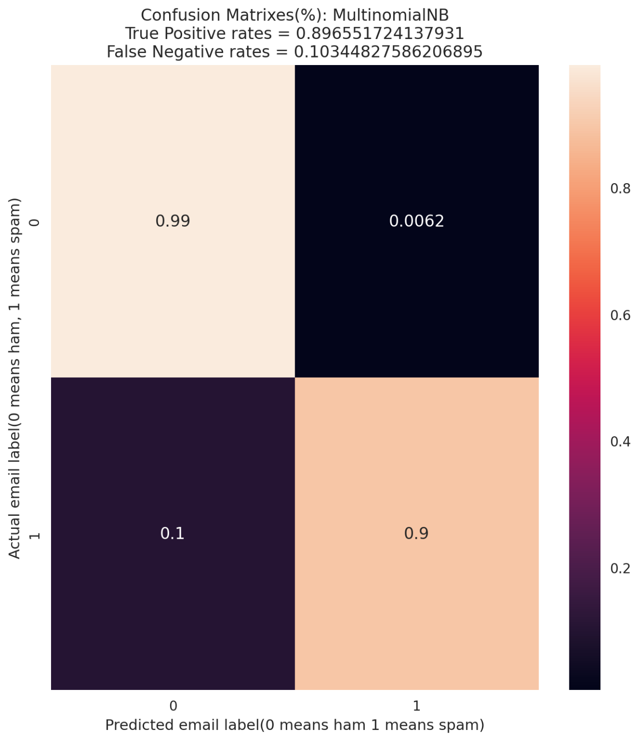
Answer b:
Try using the first method, filtering out the first 20 common words and only the first occurrence of the word. Then use the second method (stop_words='english'). Lastly, we use third method of defining parameter of max_df and min_df. Act on easyham&spam and hardham&spam respectively, and compare.

```
filter_out_words = [word[0] for word in Too_common_words]
filter_out_words2 = [word[0] for word in Too_uncommon_words]
print(f'filtered out words:(common) {filter_out_words}')
print(f'filtered out words:(uncommon) {filter_out_words2}')

# Filter out by using the words found in part (a)
cv_filter1 = CountVectorizer(stop_words=filter_out_words)
cv_filter2 = CountVectorizer(stop_words=filter_out_words2)

# Filter out by letting Sklearn do it for you
cv_filter3 = CountVectorizer(stop_words='english')

# Filter out too frequently and too unfrequently terms
cv_filter4 = CountVectorizer(max_df = 0.8, min_df = 0.004)
```

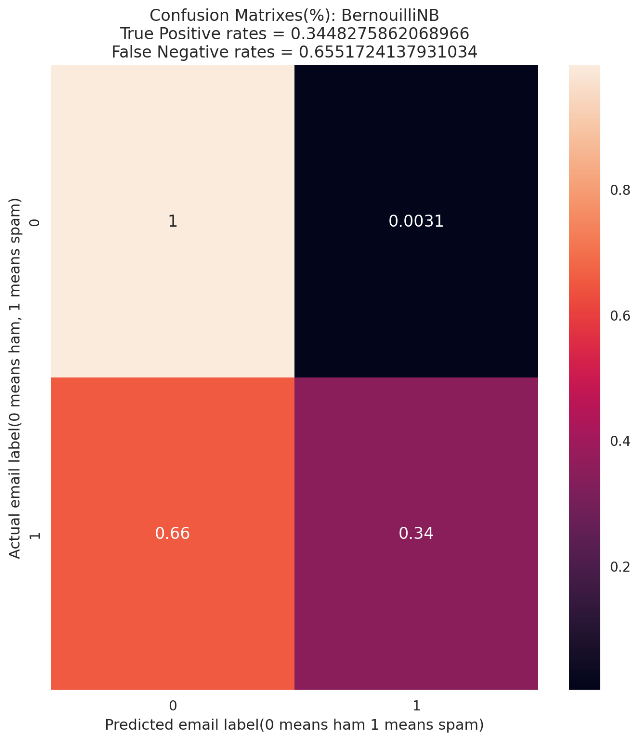
```
ejagx', 'dibi24h8oemeajldafx4agbf0byvafvsaeyzafytafxfc', 'eucacxebejvaefsy7vzo', 'qceasmwd9hzlcv', 'lswza
```

```python
# Filter out common words(easy ham vs spam)
print("Easy ham vs Spam (filter out common or/and uncommon words)\n")
print("Filter out common words by using part a: ")
Naive_Bayes_MNB_BNB(easytrain,easytest,easy_spamtrain,easy_spamtest,
                    vectorizer=cv_filter1)
print(" ")
# Filter out uncommon words(only once)
print("Filter out uncommon words by using part a:  ")
Naive_Bayes_MNB_BNB(easytrain,easytest,easy_spamtrain,easy_spamtest,
                    vectorizer=cv_filter2)
print(" ")
# Filter out by Sklearn
print("Filter out words by using Sklearn: ")
Naive_Bayes_MNB_BNB(easytrain,easytest,easy_spamtrain,easy_spamtest,
                    vectorizer=cv_filter3)
print(" ")
# Filter out too frequently and too unfrequently terms
print("Filter out too frequently and too unfrequently terms")
Naive_Bayes_MNB_BNB(easytrain,easytest,easy_spamtrain,easy_spamtest,
                    vectorizer=cv_filter4)
```

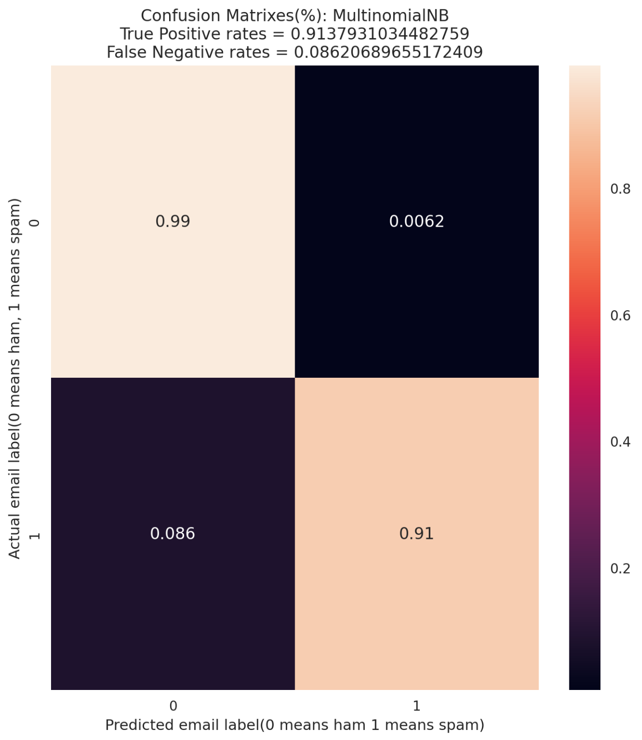
```
Easy ham vs Spam (filter out common or/and uncommon words)

Filter out common words by using part a:
Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 16
Multinomial Naive Bayes: The accuracy score is 0.979030
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 78
Bernoulli Naive Bayes: The accuracy score is 0.897772

Filter out uncommon words by using part a:
Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 14
Multinomial Naive Bayes: The accuracy score is 0.981651
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 61
Bernoulli Naive Bayes: The accuracy score is 0.920052

Filter out words by using Sklearn:
Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 25
Multinomial Naive Bayes: The accuracy score is 0.967235
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 76
Bernoulli Naive Bayes: The accuracy score is 0.900393

Filter out too frequently and too unfrequently terms
Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 11
Multinomial Naive Bayes: The accuracy score is 0.985583
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 39
Bernoulli Naive Bayes: The accuracy score is 0.948886
```



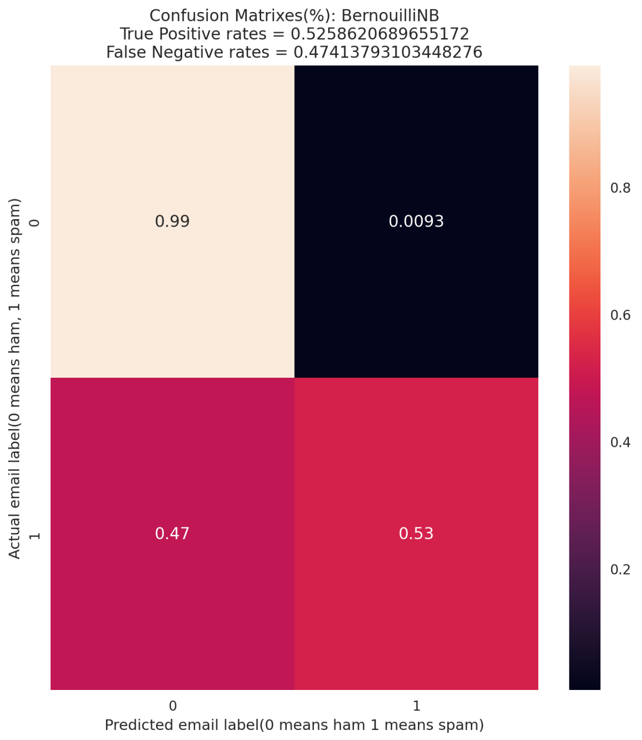
Confusion Matrixes(%): MultinomialNB
True Positive rates = 0.896551724137931
False Negative rates = 0.10344827586206895
0.99
0.0062
0.1
0.9
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0
1
0.2
0.4
0.6
0.8



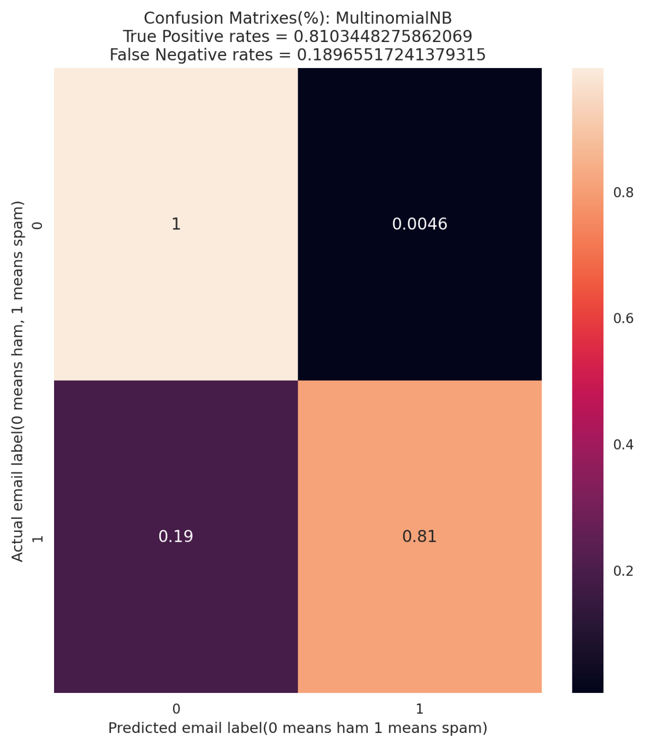
Confusion Matrixes(%): BernouilliNB
True Positive rates = 0.3448275862068966
False Negative rates = 0.6551724137931034
1
0.0031
0.66
0.34
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0
1
0.2
0.4
0.6
0.8

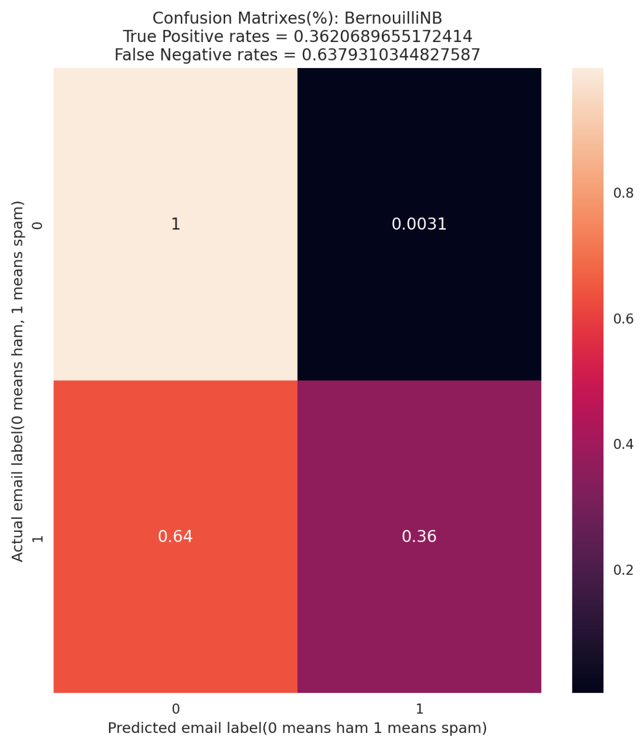


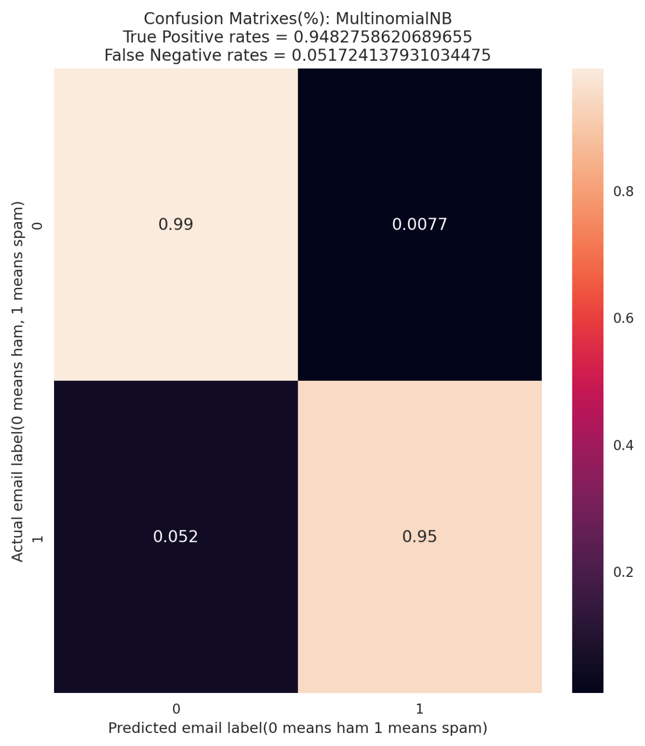
Confusion Matrixes(%): MultinomialNB
True Positive rates = 0.9137931034482759
False Negative rates = 0.08620689655172409
0.99
0.0062
0.086
0.91
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0
1
0.2
0.4
0.6
0.8

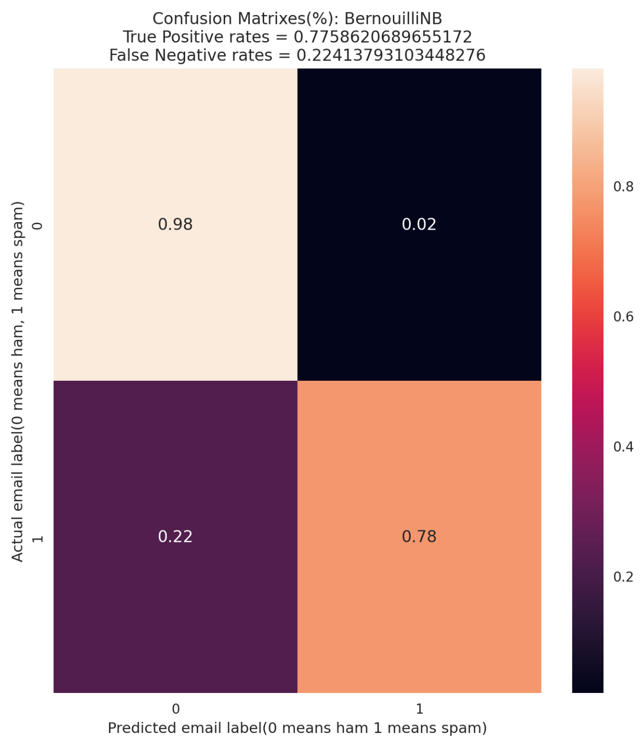


Confusion Matrixes(%): BernouilliNB
True Positive rates = 0.5258620689655172
False Negative rates = 0.47413793103448276
0.99
0.0093
0.47
0.53
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0
1
0.2
0.4
0.6
0.8





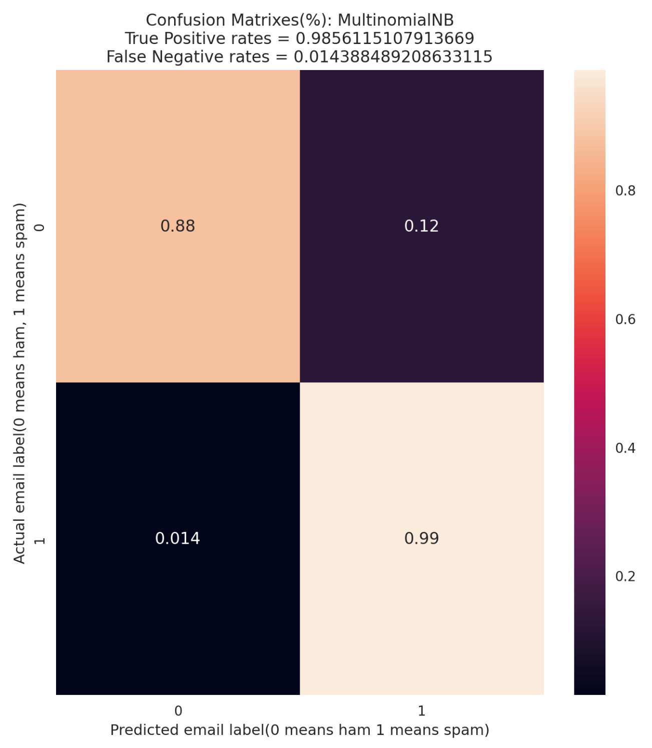


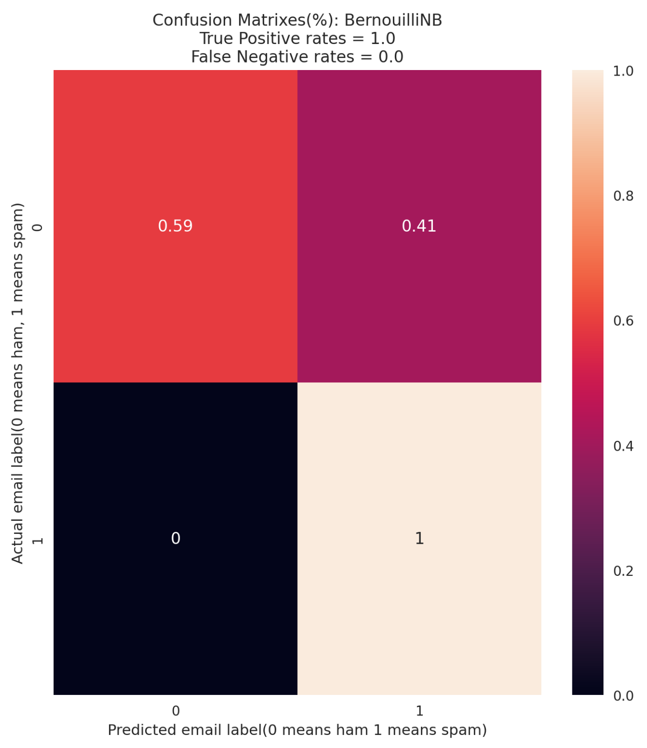


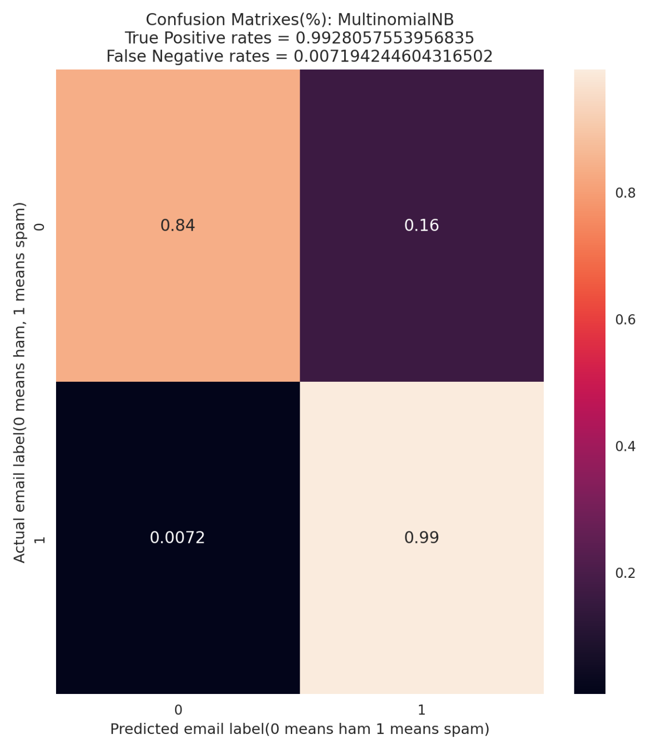
```
# Filter out common words(hard ham vs spam)
print("Hard ham vs Spam (filter out common or/and uncommon words)\n")
print("Filter out common words by using part a:  ")
Naive_Bayes_MNB_BNB(hardtrain,hardtest,hard_spamtrain,hard_spamtest,
                    vectorizer=cv_filter1)
print(" ")
# Filter out uncommon words(only once)
```

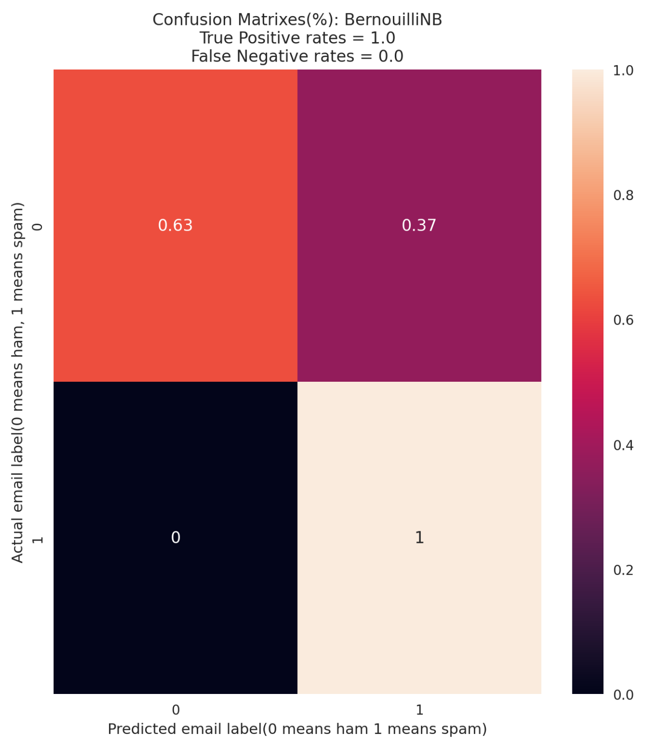
```python
print("Filter out uncommon words by using part a:  ")
Naive_Bayes_MNB_BNB(hardtrain,hardtest,hard_spamtrain,hard_spamtest,
                    vectorizer=cv_filter2)
print(" ")
# Filter out by Sklearn
print("Filter out words by using Sklearn:  ")
Naive_Bayes_MNB_BNB(hardtrain,hardtest,hard_spamtrain,hard_spamtest,
                    vectorizer=cv_filter3)
print(" ")
# Filter out too frequently and too unfrequently terms
print("Filter out too frequently and too unfrequently terms: ")
Naive_Bayes_MNB_BNB(hardtrain,hardtest,hard_spamtrain,hard_spamtest,
                    vectorizer=cv_filter4)
```

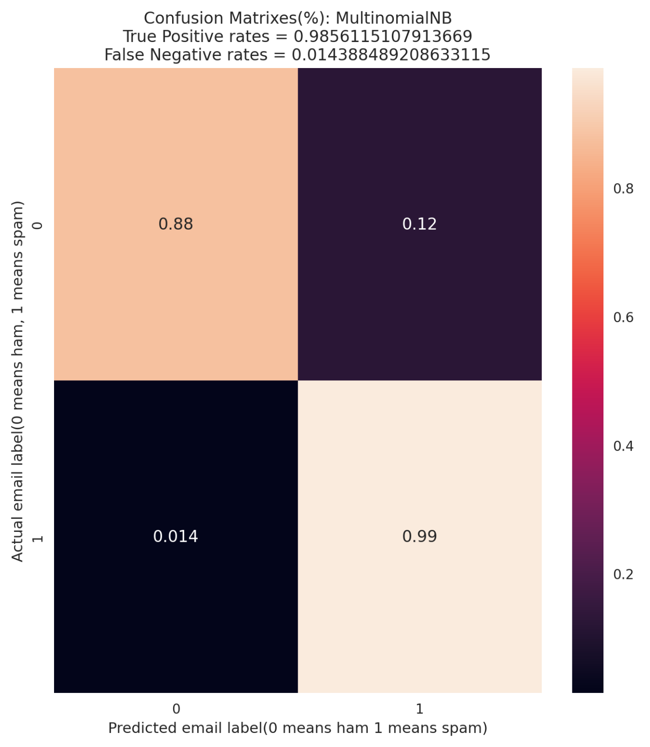
```
Hard ham vs Spam (filter out common or/and uncommon words)

Filter out common words by using part a:
Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 8
Multinomial Naive Bayes: The accuracy score is 0.957447
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 20
Bernoulli Naive Bayes: The accuracy score is 0.893617

Filter out uncommon words by using part a:
Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 9
Multinomial Naive Bayes: The accuracy score is 0.952128
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 18
Bernoulli Naive Bayes: The accuracy score is 0.904255

Filter out words by using Sklearn:
Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 8
Multinomial Naive Bayes: The accuracy score is 0.957447
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 19
Bernoulli Naive Bayes: The accuracy score is 0.898936

Filter out too frequently and too unfrequently terms:
Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 11
Multinomial Naive Bayes: The accuracy score is 0.941489
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 18
Bernoulli Naive Bayes: The accuracy score is 0.904255
```

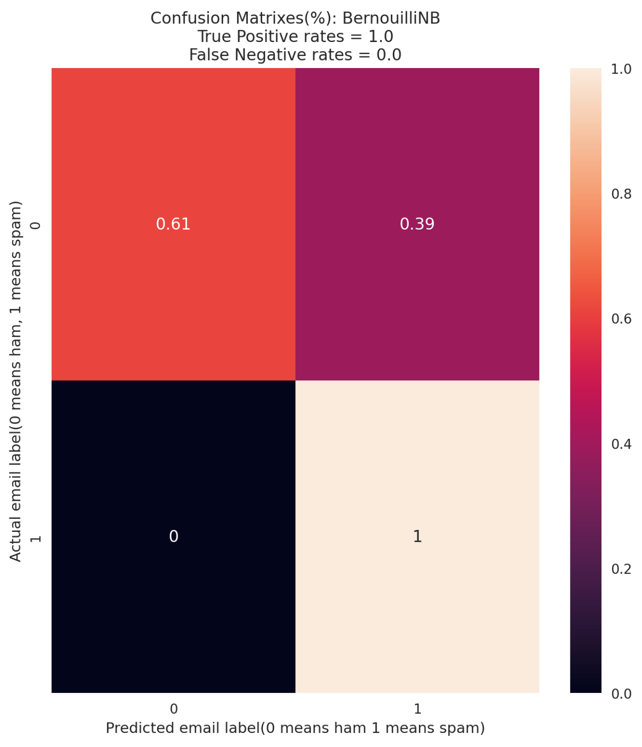

Confusion Matrixes(%): MultinomialNB
True Positive rates = 0.9856115107913669
False Negative rates = 0.014388489208633115
0.88
0.12
0.014
0.99
0
1
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0.8
0.6
0.4
0.2

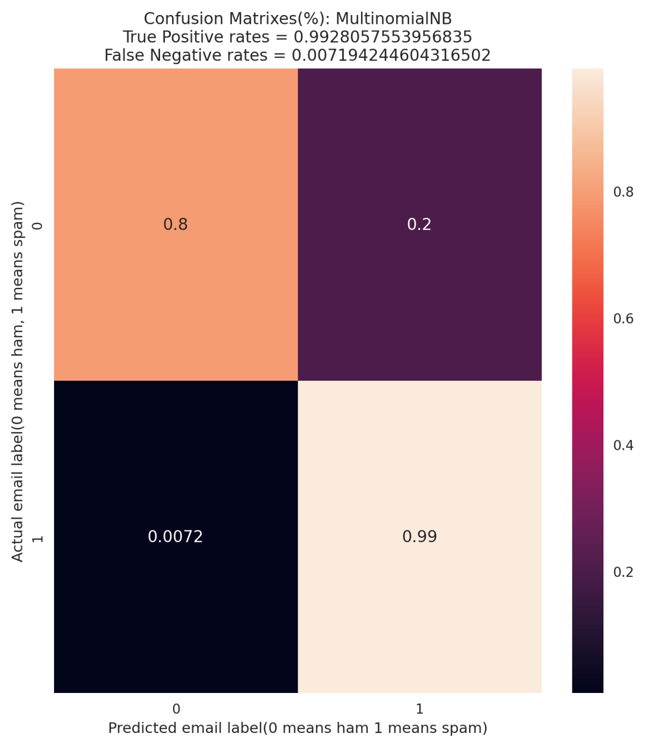

Confusion Matrixes(%): BernouilliNB
True Positive rates = 1.0
False Negative rates = 0.0
0.59
0.41
0
1
0
1
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
1.0
0.8
0.6
0.4
0.2
0.0

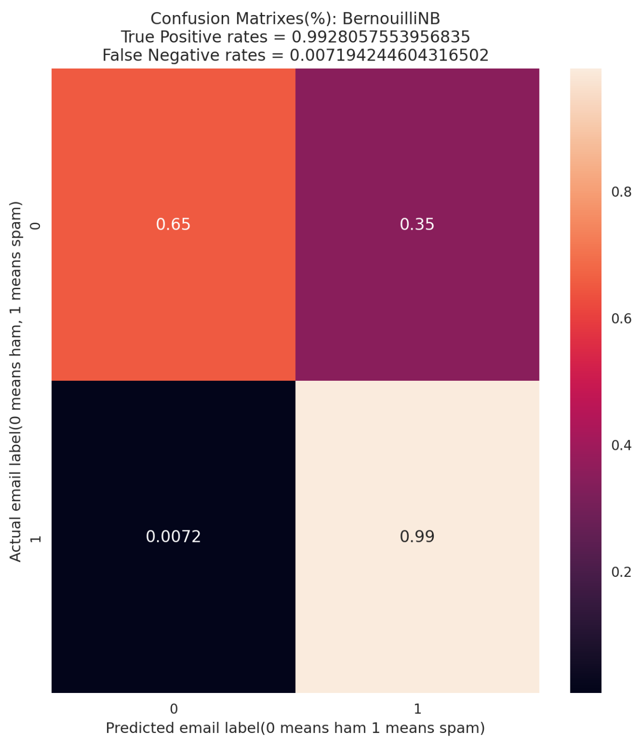

Confusion Matrixes(%): MultinomialNB
True Positive rates = 0.9928057553956835
False Negative rates = 0.007194244604316502
0.84
0.16
0.0072
0.99
0
1
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
0.8
0.6
0.4
0.2


Confusion Matrixes(%): BernouilliNB
True Positive rates = 1.0
False Negative rates = 0.0
0.63
0.37
0
1
0
1
Actual email label(0 means ham, 1 means spam)
Predicted email label(0 means ham 1 means spam)
1.0
0.8
0.6
0.4
0.2
0.0









Tried both methods of removal and found that when only the first 20 common words were filtered: the results obtained by the two methods differed very little.

In the case of filtering infrequent words (words that occur once): Bernoulli Naive Bayes and Multinomial Naive Bayes has improved accuracy. But Bernoulli Naive Bayes vary greatly in both cases, while

Multinomial Naive Bayes vary greatly only in hard ham & spam. It shows that after reducing the words that appear once, the noise is reduced, and Bernoulli Naive Bayes can classify better.

The purpose of the first method of filtering out common and uncommon words is to allow us to focus more on the important information. We can see that there is an improvement of roughly 0.01 in the accuracy of both models after the first method of filtering.

The second method aims to automatically delete some common English words. But this deletion is a more generalized operation and does not adjust to the data set. By deleting these words, we may also delete some meaningful information, so the accuracy of our model decreases compared to the first one.

Next we try the third method, the third method is to adjust the max_df and min_df parameters to remove some of the words that appear too often and ignore some of the words that appear too infrequently. We adjust max_df equal to 0.8 to remove the words that appear more than 80%, by adjusting min_df equal to 0.005 to ignore the words that appear less than 0.4%. By adjusting the parameters in this way, we identified the common and uncommon words in the dataset and filtered them out, so that we could focus more on the useful information.

From the scores and graph above we can see that the third filtering method is the best for the classification of the model. It filters the words by determining the filter boundaries of the dataset, and it filters a wider range than the first method and is more specific to the target dataset than the second method.

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.

```
# Create a function to get email body(Filter out the headers and footers of the emails)
import email
def get_email_body(msg):
  output = ""
  if msg.is_multipart():
    return get_email_body(msg.get_payload(0))
  else:
    return msg.get_payload()
```

```python
easy_ham_body = list(map(get_email_body, map(email.message_from_string, easy_ham)))
hard_ham_body = list(map(get_email_body, map(email.message_from_string, hard_ham)))
spam_body = list(map(get_email_body, map(email.message_from_string, spam)))
```

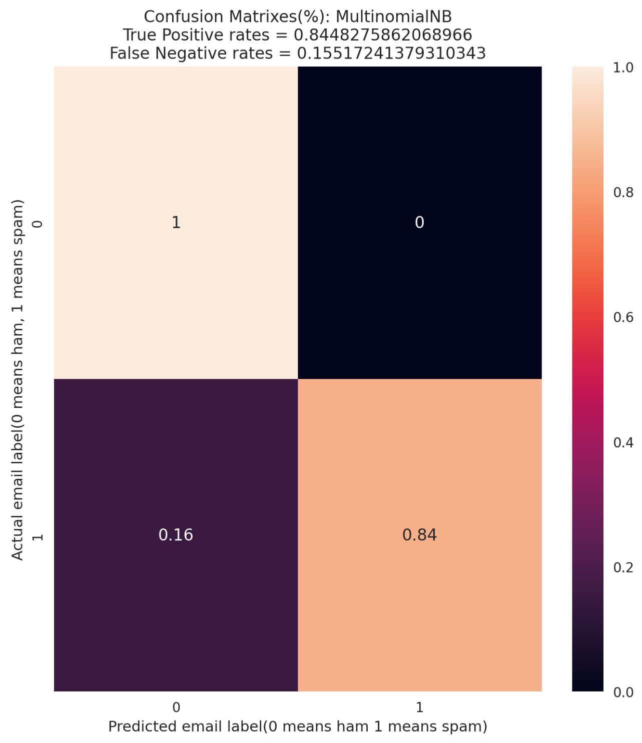
```python
# Create dataframe
df_easyham_body = pd.DataFrame(easy_ham_body)
df_hardham_body = pd.DataFrame(hard_ham_body)
df_spam_body = pd.DataFrame(spam_body)
# Create label to each email
df_easyham_body['label'] = 'easy'
df_hardham_body['label'] = 'hard'
df_spam_body['label'] = 'spam'
df_easyham_body.columns = ['bodymessage','label']
df_hardham_body.columns = ['bodymessage','label']
df_spam_body.columns = ['bodymessage','label']
# Create dataFrames for all ham(easy and hard), easy ham and spam,
# hard ham and spam, all email
df_ham_body = df_easyham_body.append(df_hardham_body,ignore_index=True)
df_ham_body['label'] = 'ham'
df_easy_spam_body = df_easyham_body.append(df_spam_body,ignore_index=True)
df_hard_spam_body =df_hardham_body.append(df_spam_body,ignore_index=True)
df_all_body = df_ham_body.append(df_spam_body,ignore_index=True)
```

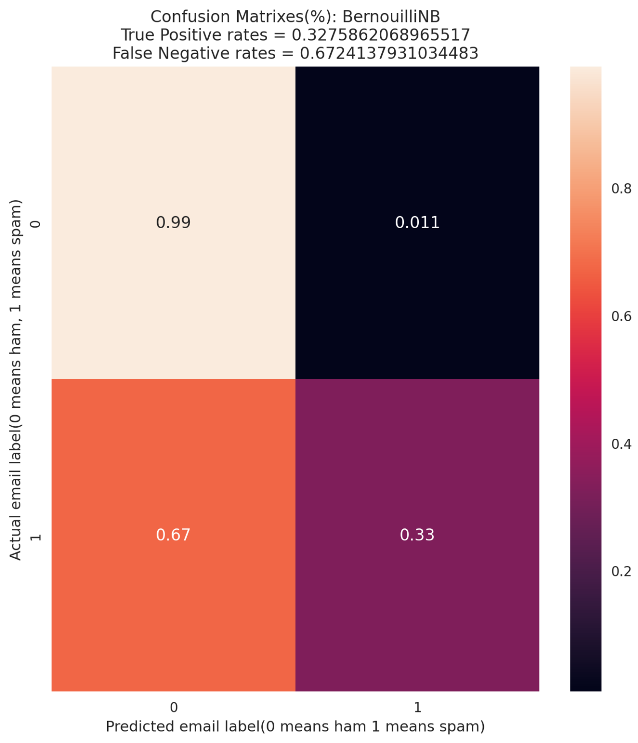
```python
# Split datasets in a training set and a test set.
hamtrain2, hamtest2, spamtrain2, spamtest2 = train_test_split(
    df_all_body['bodymessage'], df_all_body['label'],
    test_size=0.25, random_state=15)
easytrain2, easytest2, easy_spamtrain2, easy_spamtest2 = train_test_split(
    df_easy_spam_body['bodymessage'], df_easy_spam_body['label'],
    test_size=0.25, random_state=15)
hardtrain2, hardtest2, hard_spamtrain2, hard_spamtest2 = train_test_split(
    df_hard_spam_body['bodymessage'], df_hard_spam_body['label'],
    test_size=0.25, random_state=15)
```

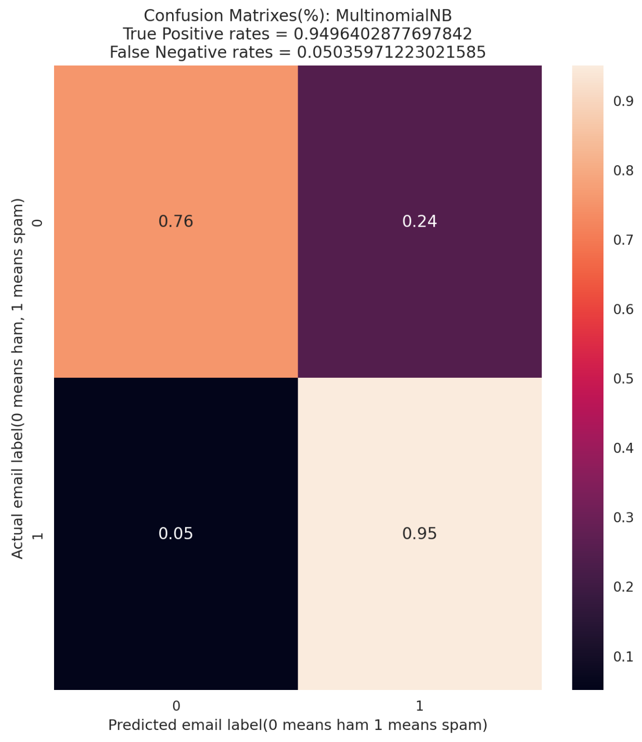
```python
print("Easy_ham & Spam (body):")
Naive_Bayes_MNB_BNB(easytrain2,easytest2,easy_spamtrain2,easy_spamtest2)
print(" ")
print("Hard_ham & Spam (body):")
Naive_Bayes_MNB_BNB(hardtrain2,hardtest2,hard_spamtrain2,hard_spamtest2)
print(" ")
```

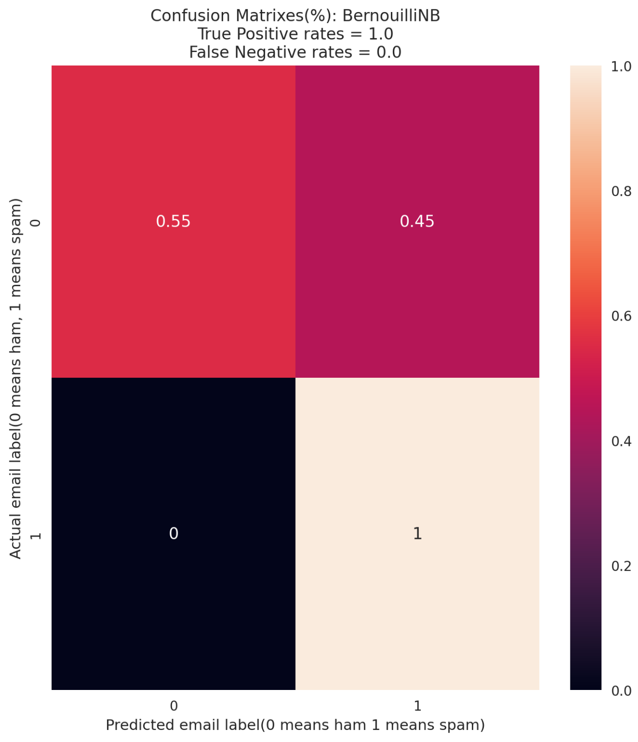
```
Easy_ham & Spam (body):
Multinomial Naive Bayes: The total points: 763, The number of mislabeled points: 18
Multinomial Naive Bayes: The accuracy score is 0.976409
Bernoulli Naive Bayes: The total points: 763, The number of mislabeled points: 85
Bernoulli Naive Bayes: The accuracy score is 0.888598

Hard_ham & Spam (body):
Multinomial Naive Bayes: The total points: 188, The number of mislabeled points: 19
Multinomial Naive Bayes: The accuracy score is 0.898936
```

```
Bernoulli Naive Bayes: The total points: 188, The number of mislabeled points: 22
Bernoulli Naive Bayes: The accuracy score is 0.882979
```









After removing the headers and footers of the emails by the above method, the result is that the classification accuracy decreases. This made us wonder if this result was caused by our incomplete removal, so we tried the second method to remove the headers and footers of the emails.

```python
from os import walk
from os.path import join
# Define file path
spam_path_1 = '/work/spam'
ham_path_1 = '/work/easy_ham'
ham_path_2 = '/work/hard_ham'

# Get the email body
def get_email_body(path):
    for root, dirnames, filenames in walk(path):
        for filename in filenames:
            file_path = join(root, filename)
            file = open(file_path, encoding='latin-1')
            is_body = False
            lines = []
            for line in file:
                if is_body:
                    lines.append(line)
                elif line=='\n':
                    is_body = True
            email_body = '\n'.join(lines)
            yield email_body, filename

def make_df(path, label):
    file_name = []
    file_data = []
    for email_body, filename in get_email_body(path):
        file_data.append({'Body': email_body, 'Label': label})
        file_name.append(filename)
    data_df = pd.DataFrame(file_data, index=file_name )
    return data_df
```

```python
# Make dataframe for the email data (spam and ham)
spam_df = make_df(spam_path_1, 1)
ham_df = make_df(ham_path_1, 0)
ham_df = ham_df.append(make_df(ham_path_2, 0))
# Generate all datasets
email_data = pd.concat([spam_df, ham_df])
# Generate id for dataset
email_id = list(range(0,len(email_data.index)))
# Add ID to the dataset
email_data['ID'] = email_id
# Set ID as the index of the dataset
email_data = email_data.set_index('ID')
# Tranfer UPPER letter to lower letter
email_data['Body'] = email_data['Body'].str.lower()
```

```python
# Import and download
from nltk import download
from nltk.corpus import stopwords
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize, RegexpTokenizer
```

```
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

```
True
```

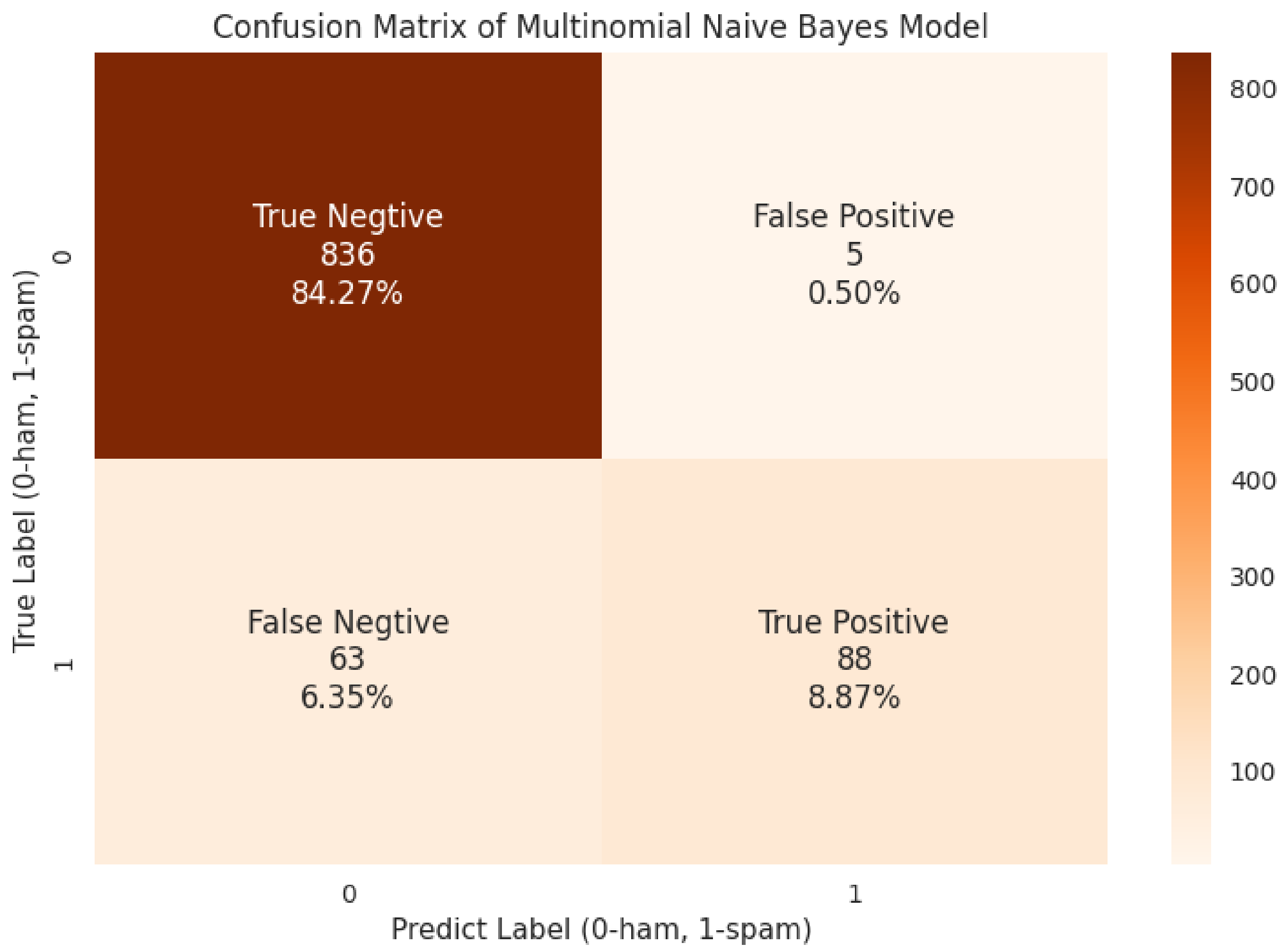
```
# Load stop words
stop_words = set(stopwords.words('english'))
stop_words.add('subject')
# Define function to process our text
def pro_text(text):
  tokenizer = RegexpTokenizer('[a-z]+')
  token = tokenizer.tokenize(text)
  lemmatizer = WordNetLemmatizer()
  token = [lemmatizer.lemmatize(w) for w in token if lemmatizer.lemmatize(w) not in stop_wo
  return token
# Apply function to process data
email_data['Body'] = email_data['Body'].apply(pro_text)
# Extract ham and spam datasets
ham = email_data.loc[email_data['Label'] == 0]
spam = email_data.loc[email_data['Label'] == 1]
# Split into hamtrain, hamtest, spamtrain, spamtest datasets
hamtrain, hamtest = train_test_split(ham, test_size=0.3)
spamtrain, spamtest = train_test_split(spam, test_size=0.3)
# Merge into train and test datasets
X_train = pd.concat([hamtrain['Body'], spamtrain['Body']])
X_test = pd.concat([hamtest['Body'], spamtest['Body']])
y_train = pd.concat([hamtrain['Label'], spamtrain['Label']])
y_test = pd.concat([hamtest['Label'], spamtest['Label']])
# Organize words in emails into sentences
x_train_texts = [' '.join(text) for text in X_train]
x_test_texts = [' '.join(text) for text in X_test]
```

```
# Transfer data to vector
cv = CountVectorizer()
X_train_count = cv.fit_transform(x_train_texts)
X_test_count = cv.transform(x_test_texts)
# Train Multinomial Naive Bayes
model_multi = MultinomialNB()
model_multi.fit(X_train_count, y_train)
# Calculate model evaluation score
multi_score = model_multi.score(X_test_count, y_test)
print('The evaluation score of Multinomial Naive Bayes is: ', multi_score)
# Make prediction
y_pred_multi = model_multi.predict(X_test_count)
# Calculate confusion matrix
```

```python
cf = confusion_matrix(y_test, y_pred_multi)
# Visualize confusion matrix
gp_names = ['True Negtive', 'False Positive',
            'False Negtive', 'True Positive']
gp_counts = ['{0:0.0f}'.format(value) for value in cf.flatten()]
gp_percentages = ['{0:.2%}'.format(value) for value in cf.flatten()/np.sum(cf)]
cf_labels = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(gp_names,gp_counts,
                                                         gp_percentages)]
cf_labels = np.asarray(cf_labels).reshape(2,2)
sns.heatmap(cf, annot=cf_labels, fmt='', cmap='Oranges')
plt.xlabel('Predict Label (0-ham, 1-spam)')
plt.ylabel('True Label (0-ham, 1-spam)')
plt.title('Confusion Matrix of Multinomial Naive Bayes Model')
```

```
The evaluation score of Multinomial Naive Bayes is:  0.9314516129032258
```

```
Text(0.5, 1.0, 'Confusion Matrix of Multinomial Naive Bayes Model')
```

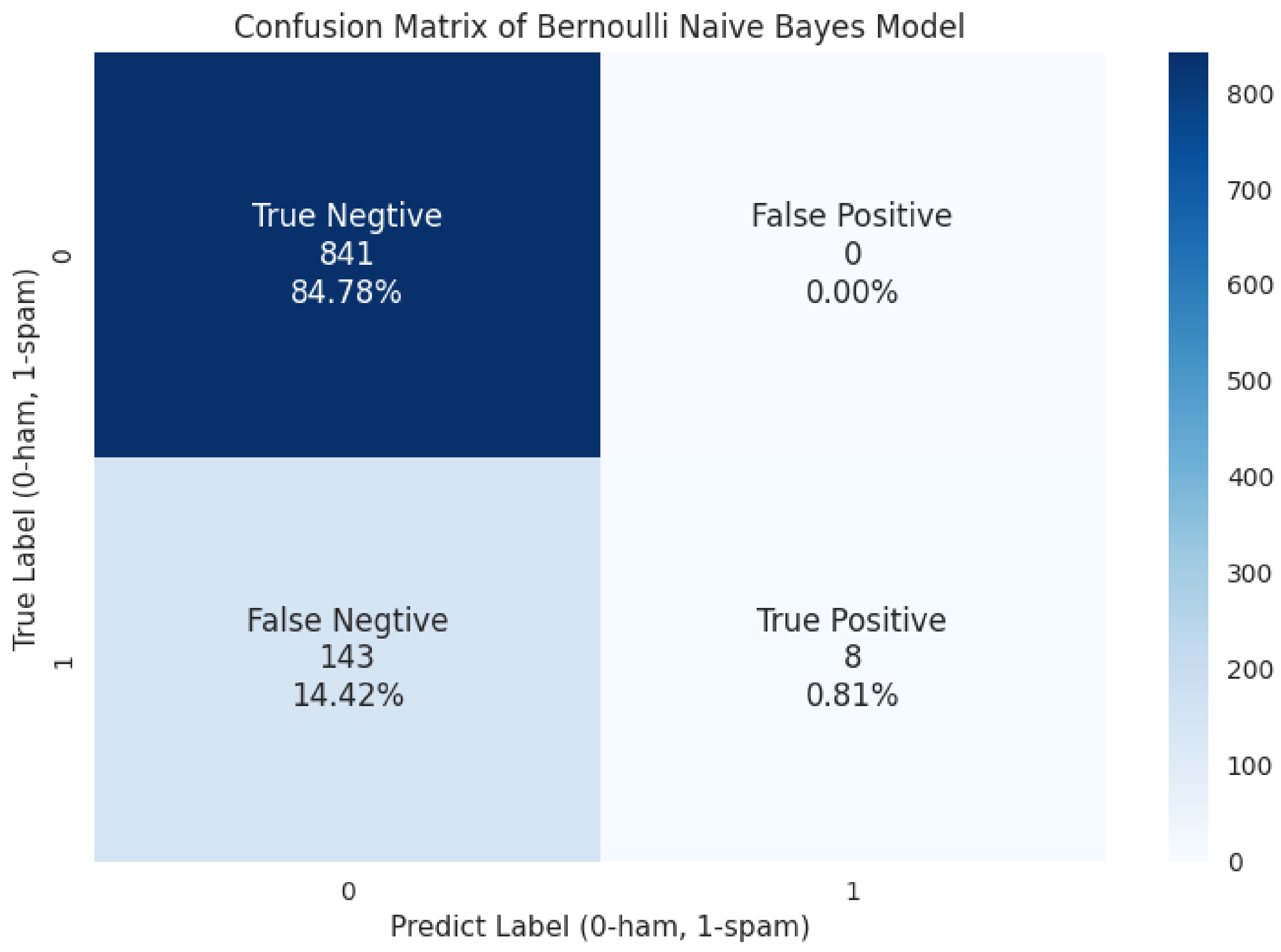


```python
# Transfer data to vector (set binary to true)
cv_bnl = CountVectorizer(binary=True)
X_train_bnl = cv_bnl.fit_transform(x_train_texts)
X_test_bnl = cv_bnl.transform(x_test_texts)
# Train Bernoulli Naive Bayes
model_bnl = BernoulliNB()
```

```
model_bnl.fit(X_train_bnl, y_train)
# Calculate model evaluation score
bnl_score = model_bnl.score(X_test_bnl, y_test)
print('The evaluation score of Bernoulli Naive Bayes is: ', bnl_score)
# Make prediction
y_pred_bnl = model_bnl.predict(X_test_bnl)
# Calculate confusion matrix
cf_bnl = confusion_matrix(y_test, y_pred_bnl)
# Visualize confusion matrix
gp_counts_bnl = ['{0:0.0f}'.format(value) for value in cf_bnl.flatten()]
gp_percentages_bnl = ['{0:.2%}'.format(value) for value in cf_bnl.flatten()/np.sum(cf_bnl)]
cf_labels_bnl = [f'{v1}\n{v2}\n{v3}' for v1, v2, v3 in zip(gp_names,
                                        gp_counts_bnl,gp_percentages_bnl)]
cf_labels_bnl = np.asarray(cf_labels_bnl).reshape(2,2)
sns.heatmap(cf_bnl, annot=cf_labels_bnl, fmt='', cmap='Blues')
plt.xlabel('Predict Label (0-ham, 1-spam)')
plt.ylabel('True Label (0-ham, 1-spam)')
plt.title('Confusion Matrix of Bernoulli Naive Bayes Model')
```

```
The evaluation score of Bernoulli Naive Bayes is:  0.8558467741935484
```

```
Text(0.5, 1.0, 'Confusion Matrix of Bernoulli Naive Bayes Model')
```



Question: Does the result improve from 3 and 4?

Answer:

Comparing with the results obtained by question3 and question4, it will be found that the classification accuracy becomes worse after removing headers and footers.
The changes in the Multinomial Naive Bayes are more obvious, and the changes in the Bernoulli Naive Bayes are small. This shows that some data that is helpful for classification in the headers and footers of the email are deleted by the data that is not conducive to classification. This also affects the classification accuracy.

Question:
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?

Answer:

It will be found that when training with all ham emails & spam and training with only easy ham & spam and training with only hard ham & spam, the results are different, although the random points are 0.75 training and 0.25 testing.

We suspect that one of the reasons for the bias is because of the sample distribution, because of randomness. It is likely that most of the data in your training samples are spam emails, which will result in a higher recognition ability for spam emails than for ham emails.

In particular, some common words appear repeatedly in both ham email and spam email, such as 'and', 'to', etc., which will affect the classification results. Because if the training set is mostly spam emails, emails containing these common words with high frequency may also be considered spam emails, but they may actually be ham emails. Also, some words appear too unfrequently and will not contribute to our training model.

Improvement method: Train after filter too frequent terms and too unfrequent terms, which is our third method of filtering.

Another reason is that the ratio between the number of ham emails and the number of spam emails in the dataset is not the same. Therefore, in the separated datasets, the ratios of ham and spam are also different, and the final recognition ability will also be different.

Improvement method: The ham and spam emails can be distributed proportionally to the training set and test set or weighted. Prevent the final classification from being inaccurate due to the large difference in the proportion of data.

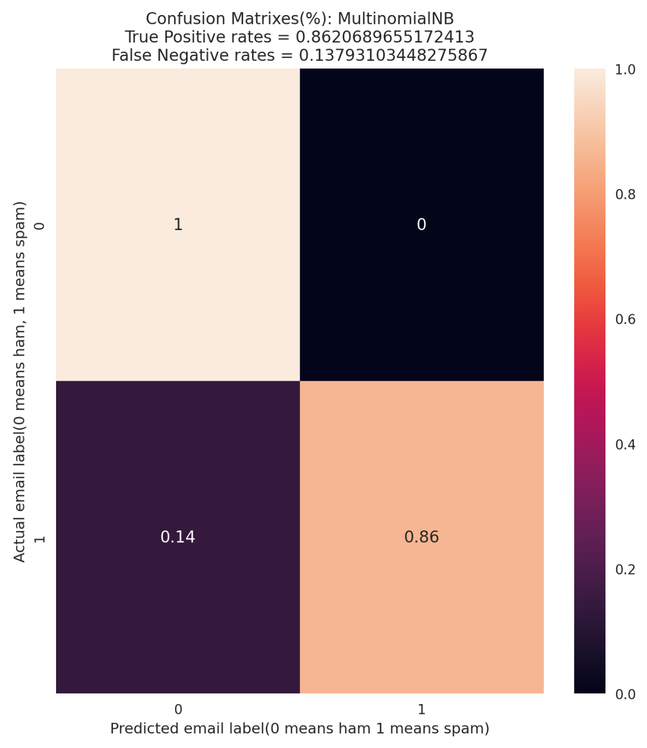
Question:
What do you expect would happen if your training set were mostly spam messages while your test set were mostly ham messages?

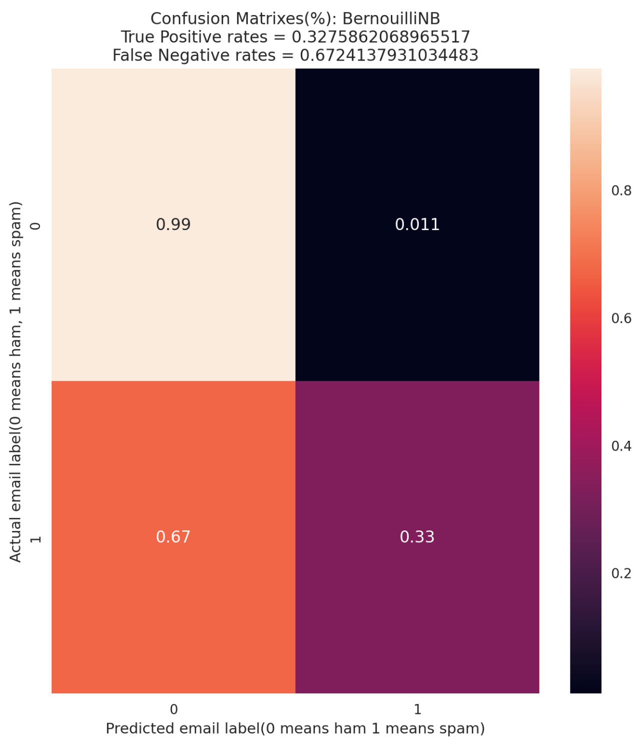
Answer:
If most of the training sets are spam emails and most of the test sets are ham emails,the false positive rate and false negative rate will increase, and the true rate will decrease. This is because the model will identify more spam indicator and treat some ham email as spam, which will affect the accuracy of the

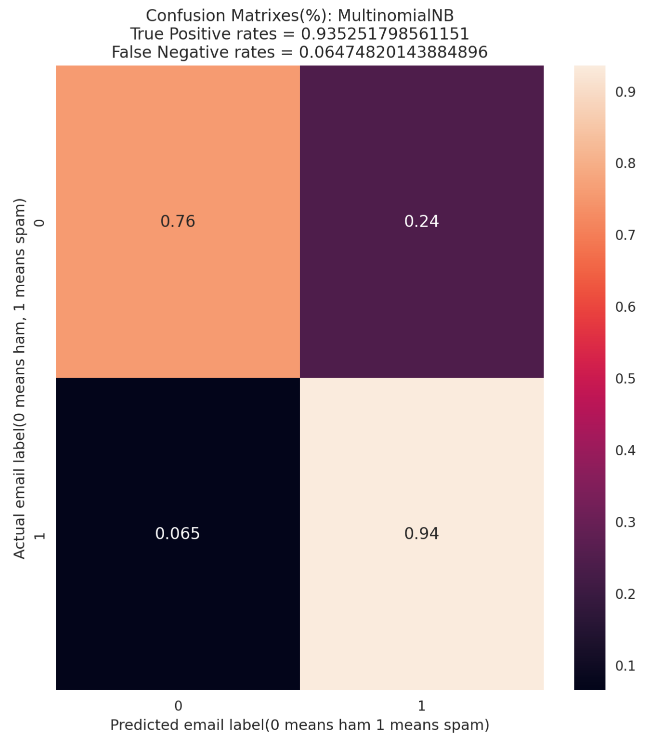
model. The end result may be: spam email and ham email cannot be distinguished correctly. Many emails that are actually ham are mistakenly identified as spam emails, and many emails that are truly spam are mistakenly identified as ham emails.

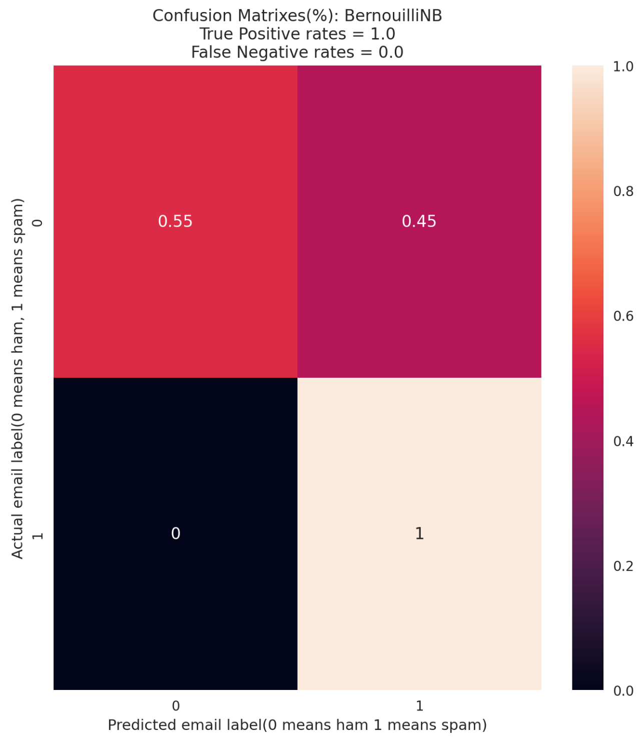
```python
#Re-estimate my classifier using fit_prior parameter set to false
print("Fit_prior = False")
print("Easy_ham & Spam (body):")
Naive_Bayes_MNB_BNB(easytrain2,easytest2,easy_spamtrain2,easy_spamtest2,fit_prior=False)

print(" ")
print("Hard_ham & Spam (body):")
Naive_Bayes_MNB_BNB(hardtrain2,hardtest2,hard_spamtrain2,hard_spamtest2,fit_prior=False)
print(" ")
```

```
Fit_prior = False
Easy_ham & Spam (body):
Multinomial Naive Bayes(fit_prior): The total points: 763, The number of mislabeled points: 16
Multinomial Naive Bayes(fit_prior): The accuracy score is 0.979030
Bernoulli Naive Bayes(fit_prior): The total points: 763, The number of mislabeled points: 85
Bernoulli Naive Bayes(fit_prior): The accuracy score is 0.888598

Hard_ham & Spam (body):
Multinomial Naive Bayes(fit_prior): The total points: 188, The number of mislabeled points: 21
Multinomial Naive Bayes(fit_prior): The accuracy score is 0.888298
Bernoulli Naive Bayes(fit_prior): The total points: 188, The number of mislabeled points: 22
Bernoulli Naive Bayes(fit_prior): The accuracy score is 0.882979
```









Question:
What does this parameter mean?

Answer:
By looking at the function documentation, we can know:
fit_prior(bool, default=True):
Whether to learn class prior probabilities or not. If false, a uniform prior will be used and will not do any regularization.

Question:
How does this alter the predictions? Discuss why or why not.

Answer:
Changing the parameters of fit_prior will not affect the results. By comparing the results before and after the changes under the easy ham & spam email and hard ham & spam email datasets, we will find that the final results do not change much. We have three guesses of why this situation appear.

First is probably because the features between ham email and spam email are sufficiently different for classification, changing fit_prior doesn't affect the final result.

Second is because when specifying the probability of not learning the class prior, class_prior will adjust the prior based on the data. So even if the no-learning prior model is specified, it will be adjusted according to class_prior.

Third is because the score of the model is calculated as the average accuracy of the given test data and label, so even if the prior probability is not learned using the standard prior will not change the score of the model.