Lab 4: Bayesian Models

Group 59

Malte Carlstedt & Johan Östling

Time Spent: 12h

1. A)

The following files were downloaded from https://spamassassin.apache.org/old/publiccorpus/.

* 20021010_hard_ham.tar.bz2
* 20021010_spam.tar.bz2
* 20021010_easy_ham.tar.bz2

The files was then unzipped by: `tar -xjvf file.bz2`

1. B)

We split our data by using the following command:

```
#splits spam and ham mails to test and train sets
hamTrain, hamTest = train_test_split(listOfHam, test_size=0.3, random_state=0)
spamTrain, spamTest = train_test_split(listOfSpam, test_size=0.3,
random_state=0)
```

We chose the test size to be 30% with no scientific reason, except that it is common practice to split the data set into this proportion.

2 A)

We used the four datasets (hamtrain, spamtrain, hamtest and spamtest) for the following variables to set up our model. x is the email and y is 1 if spam and 0 if not spam. We used the variable names x and y for no particular reason other than convenience.

```
#Setting up the variables which will build the model

x_train = numpy.concatenate((hamTrain[:,0], spamTrain[:,0]))
y_train = numpy.concatenate((hamTrain[:,1], spamTrain[:,1]))
x_test = numpy.concatenate((hamTest[:,0], spamTest[:,0]))
y_test = numpy.concatenate((hamTest[:,1], spamTest[:,1]))
```

2 B)

Brief explanation of **Multinomial Naive Bayes**:

Multinomial Naive Bayes is a variant of Bayes theorem. Multinomial Naive Bayes uses a vector that calculates how **many** times a specific word appears in a text. For an example if the word "casino" is only present once in an email it might just be a colleague asking if you'd wanna join for a casino night. If the word “casino” however is present multiple times it’s more likely an spam email from a company trying to advertise their casino. A problem with Multinomial Naive Bayes is that if for example a word never occurs in our training data set, the algorithm will not specify this as an unwanted word. For example if a spam email creator knows that a certain mailservice uses Multinomial Naive Bayes they may purposely differentiate the word they are using in their email to be able to slip by the Multinomial Naive Bayes based spam protection.

Brief explanation of **Bernoulli Naive Bayes**:

Bernoulli Naive Bayes uses binary terms to classify given data. For example it classifies our mails whether or not a certain word appears in it or not. So maybe the word ‘invest’ appears in spam mails while ham mails does not. Then it determines whether the given mail is ham or spam by seeing if the mail contains the word ‘invest’ or not.

Differences between **Bernoulli** and **Multinomial** Naive Bayes:

The difference between bernoulli and multinomial is the type of values they take in. Multinomial takes in frequencies, while bernoulli only takes in binary terms. For example in the ham or spam mails, the Multinomial model counts the appearances of certain words while the Bernoulli model looks if a certain word is in the mail or not.

3

We used the sklearn.metrics.accuary_score() to get the percentage of correct predictions made from our two models.

```
# Using Naïve Baye multinomial classifier to train our datasets.
multiNB = MultinomialNB()
multiNB.fit(trained_x_vector, y_train)
multiNB_predict = multiNB.predict(x_test_vector)
print("Accuracy Multinomial:",metrics.accuracy_score(y_test, multiNB_predict))

# Using Naïve Baye bernoulli classifier to train our datasets.
bernoulliNB = BernoulliNB(binarize=1.0)
bernoulliNB.fit(trained_x_vector,y_train)
bernoulliNB_predict = bernoulliNB.predict(x_test_vector)
print("Accuracy Bernoulli:",metrics.accuracy_score(y_test,
bernoulliNB_predict))
```

i)
for spam versus easy-ham we got the following score:

Accuracy Multinomial: 0.9640130861504908 **(96%)**
Accuracy Bernoulli: 0.8909487459105779 **(89%)**

ii)
for spam versus hard-ham we got the following score:

Accuracy Multinomial: 0.915929203539823 **(92%)**
Accuracy Bernoulli: 0.8495575221238938 **(85%)**

4 A)

Most common words

It may be worth filtering out words that are common in both the ham mails and the spam mails due to the fact that these words don't add anything to our conclusion whether this is spam or not. The reason why is because if a word is very common in both ham and spam emails, then that word would be a bad classifier for identifying whether a new mail is spam or ham. For example the word 'the' which is highly common in both ham and spam mails. Our model should not use 'the' as an indicator to whether a new mail is spam or ham, since this word is so common. To figure out which words are common in both the ham and spam emails. We used a library called "Collections" to count how many times a certain word occurs in our emails.

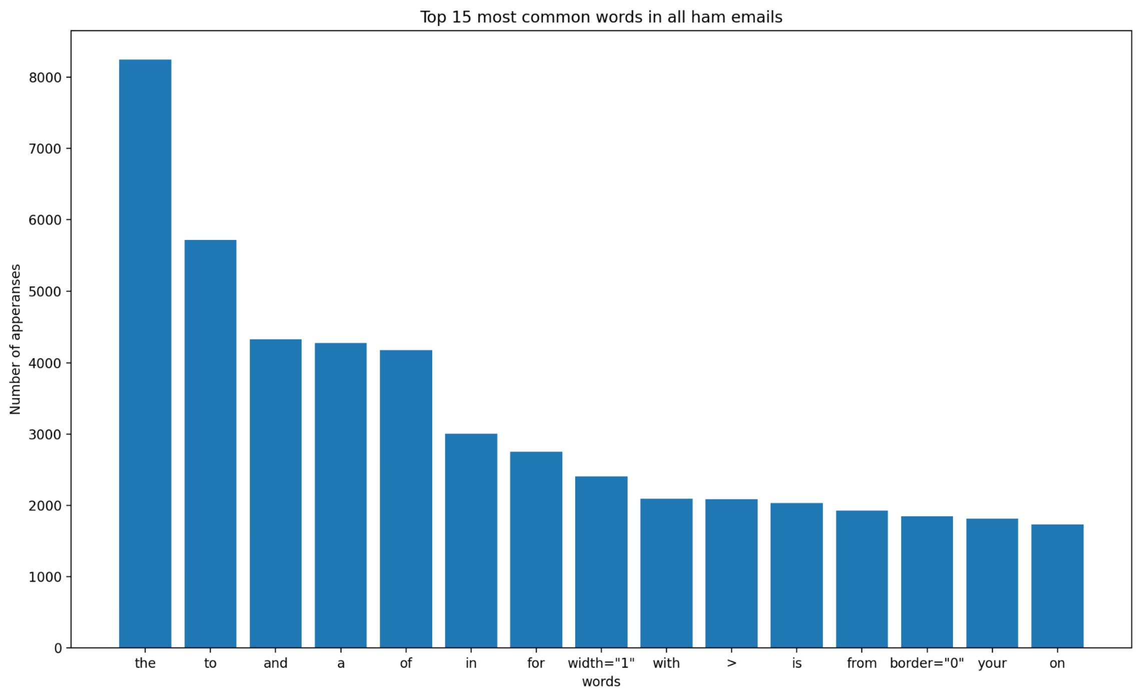
First we created a list with all words that occur in our files (emails).

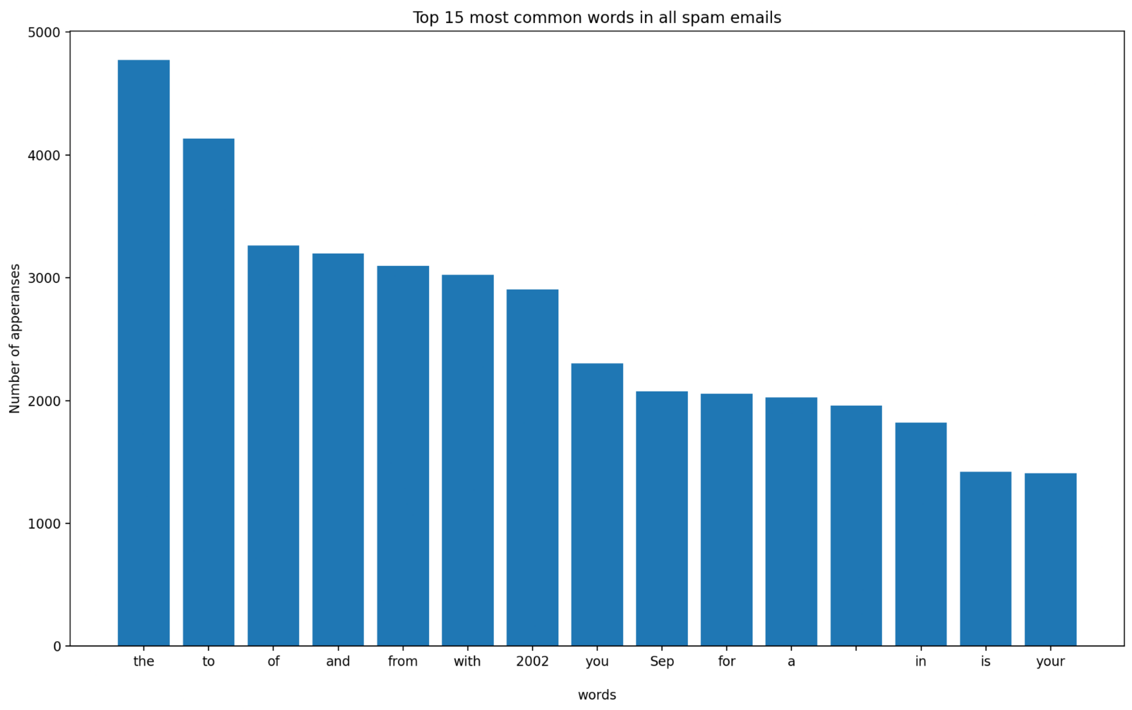
```
wordsHam = [word for email in listOfHam for word in email.split(" ")]
wordsSpam = [word for email in listOfSpam for word in email.split(" ")]
```

Using that list we count how many times each word occured.

```
word_count_spam = collections.Counter(wordsSpam)
word_count_ham = collections.Counter(wordsHam)
```

This gives us just a huge list coupled with how many times that word occured. This is not interesting for us to see exactly how many times a word occured. We therefore decided to just extract the top most common words. We settled on just extracting the top 15 words since words after that seemed to be less informative and not as common as the top 15. We decided to display the top 15 most common words of the ham emails and the 15 most common words for spam mails separately. After extracting the most common words in both types of emails we combined the results. The 15 most common words in each type of email can be seen below in a bar chart.





From the bar chart you can directly identify that the words that appear most often in the emails are regular conjunctions in the English language. Words like “the”, “to” and “from” appear very often in emails since these are used in the common language. We want to filter these words out since they do not add anything to the conclusion whether an email is spam or ham since the words appear often in both types of emails and therefore cant be concluded as ham nor spam. To be able to find which of the words that we can filter out we need to find the words that are common in both the ham mails and the spam mails. We did this by combining two lists of the most common words in each type of email and allowing no duplicates.

```
matchingWords = list(set(mostCommonWordsHam) &
set(mostCommonWordsSpam))
```

The result of this gave us that the most common words in both type of emails are: [*'is', 'a', 'for', 'in', 'to', 'from', 'of', 'with', 'the', 'and', 'your'*]. From the above conclusion we decided that these words can be filtered out. By doing this our dataset grows smaller and our algorithms get more effective.

Most uncommon words

Until now we have only discussed why the most common words can be filtered out. The opposite would be to filter out the most uncommon word. We would want to filter out the most uncommon words since those words will not help our algorithm to determine in the future whether an email is spam or ham. By then filtering these words out we can focus our model to train on contents that will help it’s learning. The words that have only occurred once or very few times in our emails may be some complicated number or url that will never occur in a new email. Thus not helping us to decide whether an email is ham or spam in the future. Filtering out words that have only occurred once or twice in our current dataset of emails will help shrinking our current dataset and thus making our algorithm quicker.

4 B)

Using our own filter

In 4a we found two lists of our most common words and most uncommon. If we combine these and use this as a filter we get the following result.

```
matchingCommonWords = list(set(mostCommonWordsHam) &
set(mostCommonWordsSpam))
matchingUncommonWords = list(set(mostUnCommonWordsHam) &
set(mostUnCommonWordsSpam))
wordsToFilter = [*matchingWords, *matchingUncommonWords]
vectorizer = CountVectorizer(stop_words=wordsToFilter)
```

Accuracy Multinomial: 0.915929203539823 (92%)
Accuracy Bernoulli: 0.8495575221238938 (85%)

Note: *This calculation was only conducted on the hard ham emails.*

Our results show that no increase in accuracy can be seen. We believe that this is because our list of common words is too small to make a difference in a dataset with over 400.000 words. Our list of common words was only 11 words long since we defined a word as common if it was in the top 15 most common words used in the emails. But if we define common as being in the top 50 most common words used in the emails, the accuracy might have increased. This number is hard for us to define so instead it is better to use a percentage parameter that classifies a word as common if it appears in x percentage of emails. Such a parameter is already built in into the countvectorizer object. Which is a reason for why it might be better to let sklearn filter out words for us instead of us telling countvectorizer which words that should be filtered out. Sorting out our uncommon words didn’t increase our accuracy either, this was however expected as stated in 4a. It might have increased the calculation speed a bit since our dataset shrinked but it didn’t increase our accuracy.

Since our filter did not increase our accuracy we chose to use sklearn's own filter.

Letting Sklearn filter out words for us

CountVectorizer has inbuilt parameters that let us choose what words that should be filtered out. We are going to set stopwords = ‘english’. This filters common English words like ‘the’.

We also use the parameters max_df which we set to 0.85. This means that if a word occurs in 85% or above of all emails, it counts as a common word and is filtered out. We chose 85% or above to make sure that only very common words are sorted out.

Then we used the parameter min_df which we set to 3. This means that if a word occurs in less than 2 or less emails, it counts as uncommon and is filtered out. We chose the value 3 to make sure that the words that are filtered out are indeed highly uncommon.

```
vectorizer = CountVectorizer(max_df = 0.85, min_df = 3,
stop_words="english")
```

Our accuracy score for multinomial and bernoulli model became:

Accuracy Multinomial: 0.9247787610619469
Accuracy Bernoulli: 0.8761061946902655
Note: *This calculation was only conducted on the hard ham emails.*

Which is a small improvement from when the model was used on the emails without any words filtered out.

5 A)

To define where the headers in the emails ended and where the footers in the emails started differed very much from every email which made it hard to find a perfect solution. We printed out several emails and saw that the emails had an html-script model. We then decided to define a header as everything in the emails which comes before the </head> tag. Which is the closing tag for the header in html. The footer was defined as everything that comes after </body> which is the closing tag for the body of the email or in other words; the content of the email.

We removed the header and footer from the emails by implementing the following code:

```python
def removeHeader(email):
  index = email.find("<html>")
  return email[index:]

def removeFooter(email):
  index = email.find("</body>")
  return email[:index]

def removeFooterAndHeader(email):
  for i in range(len(email)-1):
     email[i] = removeHeader(removeFooter(email[i]))
  return email
```

When we ran our code without filtering out any words like we did in assignment 3 we got the following result:

Accuracy Multinomial: 0.8097345132743363
Accuracy Bernoulli: 0.7876106194690266
Note: *This calculation was only conducted on the hard ham emails.*

And when we ran our code but this time we filtered out words like we did in assignment 4 we got the following result:

Accuracy Multinomial: 0.8185840707964602
Accuracy Bernoulli: 0.7920353982300885
Note: *This calculation was only conducted on the hard ham emails.*

The result did not improve when we removed headers and footers. This may be because of the difficulty of defining a perfect header and footer. This decline in accuracy may also be because some certain elements were important for our algorithm to filter out spam mails. As an example, our remove header method also removes the sender and receiver's email . It might be that the algorithm learned to differentiate which emails are from a spammer and which are from a real person.

B)

We divide our data set into a training set and test set for both the spam mails and ham mails. One problem with our way of dividing is that perhaps emails with very classifiable words end up in the test set, when we would like for it to end up in the training set.

Another problem with our dividing is that we might get that the training set includes more spam mails than we want, or the other way around, that we get fewer spam mails than we want. And this may contribute to a worse accuracy score for our models.

To avoid these issues we could try to use a random factor that chooses random emails for the training set and test set. If you only do it once, the same problem occurs and you could get unlucky and a mail that would be better suited for the training set that ends up in the test set. But if you train your model on different mails for training sets and test sets by running the code several times with a random factor for the divide-function, and save the variables for your model, we think that the model will improve. Otherwise you can try to run the code n number of times and choose the model with the best accuracy score.

C)

If our training set was mostly spam emails and our test set was mostly ham emails our training program wouldn't get enough data to compare to how an ham email should look like. The training set would then indicate more factors in an spam email as spam. When then running the test dataset our prediction would falsely accuse more ham email as spam since it hasn't compared against enough ham emails before.