DIT406: Assignment 4

Calvin Smith, Emir Zivcic

Chalmers University of Technology/University of Gothenburg

November 30, 2021

1. Preprocessing

a) Creating dataframes and cleaning data

We have constructed three different datasets, one for each of the files that we downloaded:

- Easy ham: Non spam-emails that are supposed to be easy to separate from spam-emails. Contains a total of 2551 emails.
- Hard ham: Non spam-emails that are supposed to be harder to separate from spam-emails. Contains a total of 250 emails
- Spam: spam-emails. Contains a total of 501 emails.

Each row in our datasets consists one email and its associated label (**easy ham**,**hard ham**, **spam**). Initially, we haven't altered the data or removed any entries.

b) Training and test splitting

We will use a standard 80/20 split when divding our data into training and testing data. However, in order to get the most accurate and comparable results we want to create balanced datasets before running our Naive bayes algorithm. Thus, we have constructed training and testing data where we have the same number of ham and spam messages. In total, we have divided the data into four different sets, where each set contains 50% ham- and 50% spam emails:

- Easy vs Spam (training): 802 emails.
- Easy vs Spam (testing): 200 emails.
- Hard vs Spam (training): 400 emails.
- Hard vs Spam (testing): 100 emails.

2. Differences between Multinomial and Bernoulli Naive Bayes

The major difference between Multinomial Naive Bayes and Bernoulli Naive Bayes is the fact that Multinomial takes into account occurrences of words while Bernoulli only takes into account if the word exists or not. Bernoulli can also be used with a threshold through the **binarize** parameter when creating the model. We chose instead to specify the **binary=True** parameter in our **CountVectorizer()** which gives us a matrix where counts are only 0 or 1 depending on if a word exists in an email or not.

3. Naive Bayes

Spam vs Easy Ham

The following figures shows the results after running our Naive Bayes classifiers on easy ham vs spam data. The methods perform similarly, where both models correct spam emails with 100% accuracy but the Bernoulli model is slightly worse at classifying ham emails.

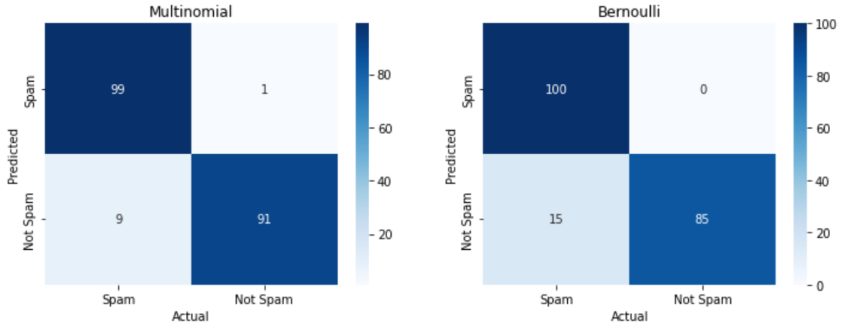

Figure 1: Spam vs easy ham

```
Multinomial %:  0.96
True Spam:  1.00
True Ham:  0.93
Bernoulli %:  0.925
True Spam:  1.00
True Ham:  0.87
```

Figure 2: Spam vs easy ham rates

Spam vs Hard Ham

The following figures show the results from running our Naive Bayes classifiers on hard ham vs spam data. The results are similar to the previous section, but

the overall predictions are worse compared to easy ham vs spam. However, this is expected since hard ham was supposed to be harder to distinguish from spam.

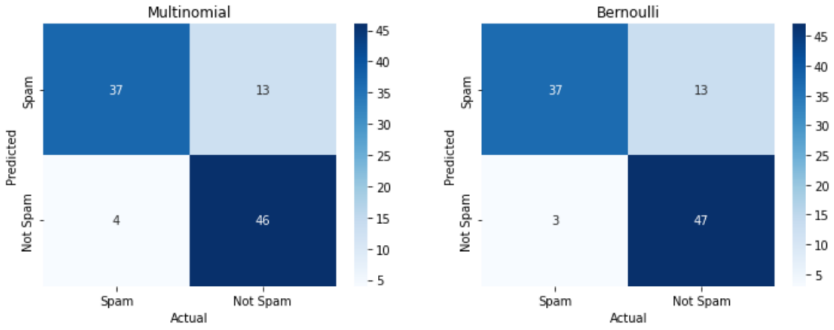

Figure 3: Spam vs hard ham

```
Multinomial %:  0.85
True Spam:  0.79
True Ham:  0.95
Bernoulli %:  0.82
True Spam:  0.78
True Ham:  0.88
```

Figure 4: Spam vs hard ham rates

4. Common/Uncommon words

a)

If certain words are very frequent in both our classes, they will not contribute much to our classification. Removing them can produce better results, since we end up with words that are more representative of each class. Similarly, words that are very uncommon, for example words that only occur once in one of the classes and doesn't occur in the other class, can produce skewed results. Since any email containing a word that only occurs in one class, will automatically be classified as the same class (since the conditional probability of the word belonging to the other class will be zero.)

To find common/uncommon words we created a function that takes some text data as input together with the number of common and uncommon words that you want to produce. The function applies CountVectorizer() to the data, transforms the resulting matrix into a dataframe, with each column representing a word and each row representing an email. We then sum over each column and sort the resulting list.

```
def words_list(X,n_common,n_uncommon):
```

```
vect = CountVectorizer()
X_counts = vect.fit_transform(X.text)

counts = pd.DataFrame(X_counts.toarray(), columns=vect.get_feature_names_out())
common = easy_counts.sum(axis = 0).sort_values(ascending = False).head(n_common)
uncommon = easy_counts.sum(axis = 0).sort_values(ascending = True).head(n_uncommon)

return_list = []
for i in range(len(common)):
  return_list.append(common.index[i])

for j in range(len(uncommon)):

  return_list.append(uncommon.index[j])

return return_list, common, uncommon
```

We can use this function to find the most common and uncommon words in our three datasets **easy_ham.df**, **hard_ham.df** and **spam.df**. Then, we will extract the words that are common for both spam emails and ham emails (easy ham and hard ham respectively) and include the words that are uncommon for both spam and ham emails. The resulting list will then be passed to **CountVectorizer()** by using stop_words = list before applying the Naive Bayes classifier.

b)

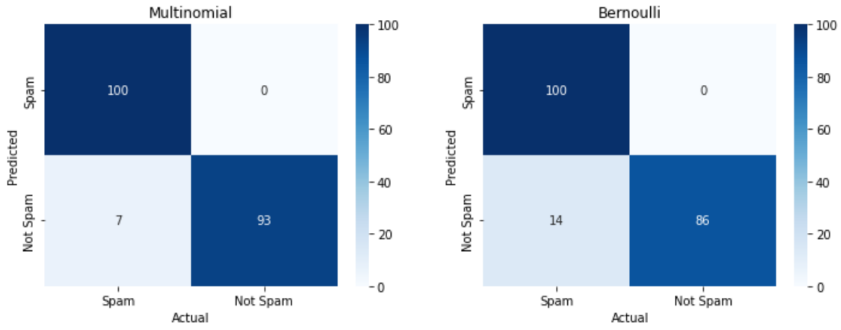

Figure 5: Spam vs easy ham rates after removing common/uncommon words only

```
Multinomial %:  0.965
True Spam:  1.00
True Ham:  0.93
Bernoulli %:  0.93
True Spam:  1.00
True Ham:  0.88
```

Figure 6: Spam vs hard ham rates after removing common/uncommon words only

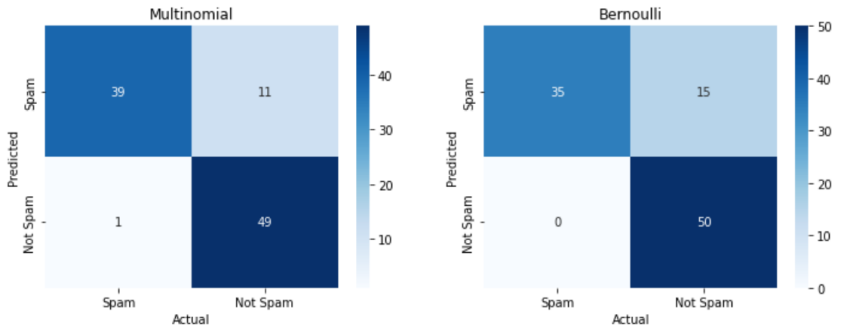

Figure 7: Spam vs hard ham rates after removing common/uncommon words only

```
Multinomial %:  0.88
True Spam:  0.82
True Ham:  0.97
Bernoulli %:  0.85
True Spam:  0.77
True Ham:  1.00
```

Figure 8: Spam vs hard ham rates after removing common/uncommon words only

5. Filtering

We have tested two ways of filtering out the headers and footers of the data. The first method comes from looking at how the emails are formatted and (perhaps naively) noticing that in a lot of the emails, the actual message starts and ends with a quotation mark (”). So, all we need to do to filter out everything besides the message is to find the index of the first and and last quotation mark and keep everything in between. For this we created a function **filter_out**:

```
def filter_out(data):

  df_out = pd.DataFrame()
```

```
  for i in range(len(data)):

    txt = data.text.iloc[i]
    start = txt.find('"')
    finish = txt.rfind('"')
    out = txt[start+1:finish]
    #df_out = df_out.append(out)
    df_out = df_out.append({'text' : out},ignore_index = True)

  df_out['label'] = [i for i in data.label]
  return df_out
```

The second method utilizes an email and regular expression module. We chose to continue with this method since it utilizes modules that are well-developed for parsing emails.

```
def extract_body(e):
    body = ""
    b = email.message_from_string(e)
    if b.is_multipart():
        for part in b.walk():
            ctype = part.get_content_type()
            if ctype == 'text/plain':
                return part.get_payload(decode=True)
            if ctype == 'text/html':
                body = part.get_payload(decode=True)
                return re.sub('<[^<]+?>', '', body.decode('latin-1'))
    else:
        return b.get_payload(decode=True)
```

We chose to not use **sklearn's** own **stop_words = 'english'** since it is general and might remove words that are highly informative to our task.

a)

The following tables show the result of running our Naive Bayes classifiers on data where we have filtered out headers and footers and removed common/uncommon words.

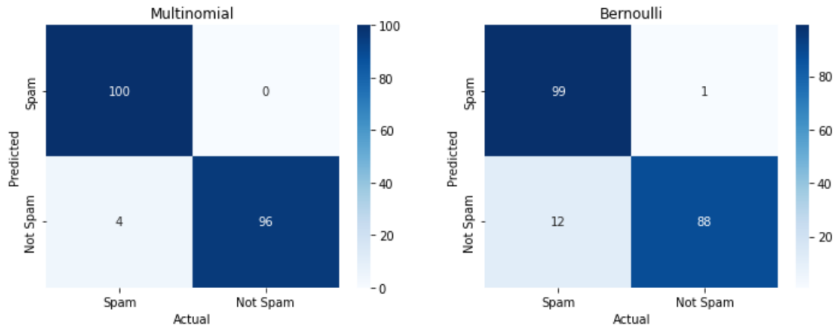

Figure 9: Spam vs easy ham after filtering and removal of common/uncommon words

```
Multinomial %:  0.98
True Spam:  1.00
True Ham:  0.96
Bernoulli %:  0.935
True Spam:  0.99
True Ham:  0.89
```

Figure 10: Spam vs easy ham rates after filtering and removal of common/uncommon words

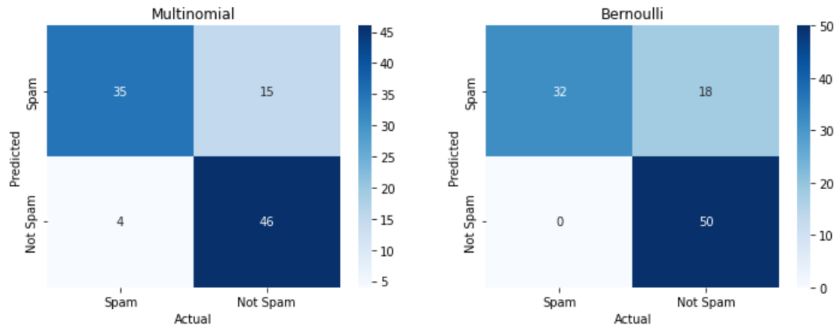

Figure 11: Spam vs hard ham after filtering and removal of common/uncommon words

```
Multinomial %:  0.81
True Spam:  0.75
True Ham:  0.90
Bernoulli %:  0.82
True Spam:  0.74
True Ham:  1.00
```

Figure 12: Spam vs hard ham rates after filtering and removal of common/uncommon words

b)

There are two main issue to consider when splitting your data into training and testing, in particular when using a Naive Bayes classifier. The first one is balance between classes. The goal of balancing your data should be so that it reflects the "true" balance of the population you are classifying. Since the true balance is almost never known and limited to the data that you have, the safest bet is to have equally many observations from each class. The Naive Bayes classification uses the following (simplified) formula to compute probability estimates of an email being spam or ham:

$$P(spam|email) \propto P(spam)P(email|spam)$$

$$P(ham|email) \propto P(ham)P(email|ham)$$

where the prior probabilities $P(spam)$ and $P(ham)$ are estimated as the proportion of spam and ham emails in the training data respectively. Thus, having for example a training set with 90% spam emails and 10% ham emails will probably result in a biased classification towards the most prevalent class in the data (unless the actual balance of ham vs spam is a ratio of 90/10). If we instead use a balanced data set, we can simply disregard the prior.

The second issue is when splitting your data results in words appearing in the testing data but not in the training data. Say for example that in our training data the word "free" appears in 1000 emails which are all classified as spam. In our testing data we have a ham email that says "are you free tonight?". The Naive Bayes classifier will predict this email as a spam email because none of the emails in our training data contain the word "free". A remedy for this is to use a smoothing parameter that accounts for words that are not in the training data (for example adding 1 to each count in the data).

c)

We would expect the classifier to perform very well when classifying spam messages as spam messages but it would probably wrongly classify some ham messages as spam messages depending on how extreme the ratio of spam/ham in the training data.

6. Summarizing results

The following tables are summaries of the results we have obtained from our different runs.

Type:	Normal	Stop	Filter + Stop
True Spam	1.0	1.0	1.0
True ham	0.93	0.93	0.96

Table 1: Summary of results when using Multinomial NB and easy ham vs spam

Type:	Normal	Stop	Filter + Stop
True Spam	1.0	1.0	0.99
True ham	0.87	0.88	0.89

Table 2: Summary of results when using Binomial NB and easy ham vs spam

Type:	Normal	Stop	Filter + Stop
True Spam	0.79	0.82	0.75
True ham	0.95	0.97	0.90

Table 3: Summary of results when using Multinomial NB and hard ham vs spam

Type:	Normal	Stop	Filter + Stop
True Spam	0.78	0.77	0.74
True ham	0.88	1.0	1.0

Table 4: Summary of results when using Binomial NB and hard ham vs spam

The results do not differ significantly when running combinations of unfiltered data and data that has been stripped of the most common words between email groups. The only major difference that can be seen in the result is that **Bernoulli Naive Bayes** performs worse than **Multinomial Naive Bayes** and that the prediction rate is lower for **hard_ham** prediction and higher for **easy_ham** predictions.

Perhaps **Bernoulli** would perform better if a higher threshold was found instead of simply allowing words with low occurrences to be **True**.

Where **Bernoulli** performs worse specifically seems to be with predicting emails as ham when they are actually spam. This can be seen in figure 9 where **Bernoulli** classified