Lukas Jigberg 15 timmar

In [201]:

```
#Download data to files
!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

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import os
```

```
--2022-02-16 18:51:46--  https://spamassassin.apache.org/old/publiccorpus/
20021010_easy_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)... 151.101.2.1
32, 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: 1677144 (1.6M) [application/x-bzip2]
Saving to: ‘20021010_easy_ham.tar.bz2.10’

20021010_easy_ham.t 100%[===================>]   1.60M  --.-KB/s    in 0.0
7s

2022-02-16 18:51:46 (24.1 MB/s) - ‘20021010_easy_ham.tar.bz2.10’ saved [16
77144/1677144]

--2022-02-16 18:51:46--  https://spamassassin.apache.org/old/publiccorpus/
20021010_hard_ham.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)... 151.101.2.1
32, 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.bz2.10’

20021010_hard_ham.t 100%[===================>] 997.19K  --.-KB/s    in 0.0
6s

2022-02-16 18:51:46 (17.5 MB/s) - ‘20021010_hard_ham.tar.bz2.10’ saved [10
21126/1021126]

--2022-02-16 18:51:46--  https://spamassassin.apache.org/old/publiccorpus/
20021010_spam.tar.bz2
Resolving spamassassin.apache.org (spamassassin.apache.org)... 151.101.2.1
32, 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.bz2.10’

20021010_spam.tar.b 100%[===================>]   1.14M  --.-KB/s    in 0.0
6s

2022-02-16 18:51:47 (17.6 MB/s) - ‘20021010_spam.tar.bz2.10’ saved [119258
2/1192582]
```

# 1. Preprocessing

In [202]:

```
#Creates a list with all the mails
def createDataDir(folder):
  ddList = []
  entries = os.listdir(os.path.abspath(folder))
  for entryName in entries:
    ddList.append(open(folder + entryName, "r", errors='ignore').read())
  return ddList

# Splits our lists for 70% train and 30% testing
hamTrain, hamTest = train_test_split(createDataDir("easy_ham/") + createDataDir("hard_h
am/"), test_size=0.3, random_state=7)
easyhamTrain,easyhamTest = train_test_split(createDataDir("easy_ham/"), test_size=0.3,
random_state=7)
hardhamTrain,hardhamTest = train_test_split(createDataDir("hard_ham/"), test_size=0.3,
random_state=7)
spamTrain,spamTest = train_test_split(createDataDir("spam/"), test_size=0.3, random_sta
te=7)
```

# 2. Model and Accuracy Test

I choose to test:\ Multinomial Naive Bayes \ Bernoulli Naive Bayes

In [203]:

```
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB

#Train the model with the given data, then predict
def trainPredict(mailTrain,mailTest):

  cv = CountVectorizer()
  mNB = MultinomialNB()
  bNB = BernoulliNB()

  train = cv.fit_transform(mailTrain + spamTrain)

  mNB.fit(train, ['ham']*len(mailTrain) + ['spam']*len(spamTrain))
  bNB.fit(train, ['ham']*len(mailTrain) + ['spam']*len(spamTrain))

  mPredict = mNB.predict( cv.transform(mailTest) )
  bPredict = bNB.predict( cv.transform(mailTest) )

  return (mPredict,bPredict)



x = trainPredict(hamTrain, hamTest)

#Print out the results
hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - Ham:", hamAmount, spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - Ham:", hamAmount, spamAmount)

x = trainPredict(hamTrain, spamTest)

hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - Spam:", hamAmount , spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - Spam:", hamAmount , spamAmount)
```

```
Multinomial - Ham: ['ham' 'spam'] [837   4]
Bernoulli - Ham: ['ham' 'spam'] [840   1]
Multinomial - Spam: ['ham' 'spam'] [ 17 134]
Bernoulli - Spam: ['ham' 'spam'] [115  36]
```

Ham Accuracy: \ Multinomial Ham - 99.64% \ Multinomial Spam - 86.75%

Bernoulli Ham - 99.88%\ Bernoulli Spam - 23.84%

## Model discussion

Both models where great at classifying whether the hamTest contained any spam. But they are still not perfect, multi missing 3 and berno missing 1.

However, when looking at the Spam emails did the result worsen and differ more. Multi classified 20 mails as ham and had a accuracy of 87%. Berno did much worse classifying 115 of the emails as ham when they were all spam.

As mentioned in the lab description will we improve these models a bit later, but for now can we look at why the result between the two models varied. Multi works by looking at frequency of words / features to calculate its result. Berno does not take the amount of words into account, it is designed for binary/boolean features. \ The big issue with Berno is that it explicitly penalizes the non-occurrence of a feature that is 'supposed' to appear for that label. Berno might classify spam as ham because many spam mails contain the required features/words for a ham mail.

Nonetheless, our model would definitely not be ready nor use. We need to improve their accuracy.

# More model tests

In [204]:

```
x = trainPredict(easyhamTrain,easyhamTest)

hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - EasyHam:", hamAmount, spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - EasyHam:", hamAmount, spamAmount)

x = trainPredict(easyhamTrain,spamTest)

hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - Spam:", hamAmount, spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - Spam:", hamAmount, spamAmount)
```

```
Multinomial - EasyHam: ['ham' 'spam'] [764   2]
Bernoulli - EasyHam: ['ham' 'spam'] [764   2]
Multinomial - Spam: ['ham' 'spam'] [ 27 124]
Bernoulli - Spam: ['ham' 'spam'] [86 65]
```

Easy Ham Accuracy: \ Multinomial Ham - 99.99% \ Multinomial Spam - 87.41%

Bernoulli Ham - 100% \ Bernoulli Spam - 50.99%

\

Very similar to the result we got from the 'all ham' model. Multi preforms okay and Berno is really bad, mislabelling half the spam emails, however still better compared to 'all ham'.

In [205]:

```
x = trainPredict(hardhamTrain, hardhamTest)

hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - HardHam:", hamAmount, spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - HardHam:", hamAmount, spamAmount)

x = trainPredict(hardhamTrain, spamTest)

hamAmount, spamAmount = np.unique( x[0], return_counts=True)
print("Multinomial - Spam:", hamAmount, spamAmount)
hamAmount, spamAmount = np.unique( x[1], return_counts=True)
print("Bernoulli - Spam:", hamAmount, spamAmount)
```

```
Multinomial - HardHam: ['ham' 'spam'] [64 11]
Bernoulli - HardHam: ['ham' 'spam'] [47 28]
Multinomial - Spam: ['ham' 'spam'] [  1 150]
Bernoulli - Spam: ['ham' 'spam'] [  1 150]
```

Hard Ham Accuracy: \ Multinomial Ham - 86.66% \ Multinomial Spam - 96.69%

Bernoulli Ham - 70.66% \ Bernoulli Spam - 97.35%

\

This data is almost flipped, this time is the spam data that performs well while ham is struggling. Its also interesting that compared to the rest of the tests is Multi and Berno this time much closer in accuracy to one another.

Unsurprisingly does the models struggle with differentiating the emails this time, the point was that hard ham should look like spam. As is written in the README; \ " closer in many respects to typical spam: use of HTML, unusual HTML markup, coloured text, "spammish-sounding" phrases etc. " \ making it so that the models have less 'clear evidence' to look for. Another factor that may play a part is that we feed the model more spam than hard ham, giving it 75 ham emails and 151 spam emails. We can't really tell if this influences the model but it’s often good to try and feed the model enough data of one type. It may be that 75 is too low a number.

As for why the models are much better at differentiating spam emails this time may be because its just worse at predicting ham. The results are almost flipped from previous test, this time is spam easy to guess while ham is difficult. It can be safe to assume that the models are just very eager to classify an email as a specific type, ham or spam. This time it really wants to classify things for spam as opposed to earlier test where many emails where ham.

The thing we are looking for is a model that can differentiate the two and has good accuracy all around and isn't biased towards one type.

# Filter E-mails

# Discussion & Word counting

It can be very useful for ours to filter our emails for a number of reason.

1. Many emails (spam and ham) contain a precede of normal features. Common words like the, of, with etc. Having our model picking up and learning that these non-spam words are in spam emails will lead to inaccuracy. Mind that we can and should also remove them from ham also. If we let ham learn that these 'common words' are a clear sign that it’s not spam, will it classify every somewhat smart spam as ham and since there is not even a mention of these words in spam (we removed them) will it make the model extremely sure of it. Again, leading to inaccuracy.
2. Removing words and features used once such as abnormal words and confusing characters set ( 'Thalassophobia' or 'coco@rad.az' ) might improve our model as it focuses less on words that are, more or less, random. If we don't our model might classify a feature that we know is pointless as an indicator to a ham or spam mail.

In [206]:

```
#Counts the word usage
def mostFeatures(mailSet, max):
  cv = CountVectorizer().fit(mailSet)
  featuresSummed = cv.transform(mailSet).sum(axis=0)
  words_freq = [(word, featuresSummed[0, idx]) for word, idx in cv.vocabulary_.items()]
  words_freq = sorted(words_freq, key = lambda word: word[1], reverse=True)
  return words_freq[0:max]

#Returns the amount of words used <=max times
def leastFeatures(frequency, max):
  count=0
  for elem in frequency:
    if elem[1] <= max:
      count+=1
  return count

#Returns a list of the least used words
def _leastFeatures(frequency, max):
  least = []
  while max!=0:
    least.append(frequency.pop())
    max-=1
  return least

print("Hams most used words: ",mostFeatures(hamTrain, 20))
print("Spams most used words: ",mostFeatures(spamTrain,20))
print("Spams words used once: ",leastFeatures(mostFeatures( spamTrain,2000000 ),1 ) )
print("Hams words used once: ",leastFeatures(mostFeatures( hamTrain,2000000 ),1 ) )
print("Some of the least used words: ",_leastFeatures(mostFeatures( hamTrain+spamTrain,
200000 ), 20 ))
```

```
Hams most used words:  [('com', 45658), ('the', 24306), ('http', 23206),
('to', 22047), ('2002', 17275), ('td', 17081), ('from', 17046), ('net', 14
806), ('for', 13915), ('width', 13279), ('with', 13087), ('by', 12752),
('www', 11982), ('of', 11609), ('localhost', 11381), ('and', 11350), ('i
d', 11043), ('3d', 10965), ('example', 10794), ('received', 10603)]
Spams most used words:  [('3d', 7691), ('font', 7490), ('to', 4640), ('co
m', 4129), ('td', 4018), ('the', 3889), ('from', 3018), ('for', 2854), ('y
ou', 2830), ('br', 2828), ('and', 2735), ('size', 2604), ('2002', 2487),
('tr', 2370), ('with', 2335), ('width', 2293), ('of', 2239), ('by', 2154),
('http', 2133), ('face', 1881)]
Spams words used once:  18431
Hams words used once:  31276
Some of the least used words:  [('7c6644', 1), ('7c306356', 1), ('01c24db
a', 1), ('c89a33c0', 1), ('0460', 1), ('wv0vghbwi8lg', 1), ('acjnusiab0tdx
mtxr', 1), ('0706fea9', 1), ('c89a5ad0', 1), ('74ab201c24dba', 1), ('qvp00
78', 1), ('117dd2e06d', 1), ('maa31139', 1), ('98b8943f99', 1), ('e17oir
2', 1), ('17oir2', 1), ('3496', 1), ('0008hs', 1), ('17oir9', 1), ('17ois
7', 1)]
```

Creating some statistics can more easily see what we predicted earlier. \ There are a lot of single used words, which isn't that weird when there are words like 'bott_right' and 'header_4'. \ Spam and Ham share some of their most frequent words. \

We can also see that Ham have more words only once and that its most used words are used much more than spams most used words.

# Using SkLearn

CountVectorizer has inbuilt filter options that will help machine learning models. Max_df is a parameter that removes words that appear in x% or x amount of the documents. We can use it to remove common words like the, with, com etc. Min_df works similarly as it removes words that only appear in x% or x amount of documents. We can use it to remove words only used in 1,2.. documents, as they wouldn't really help our model. \ SkLearn also have a language filter, they point to it being unstable but we can atleast try it and see if we get any difference.

Below are the functions from before now with the CountVectorizer parameters max and min DF (Document Frequency) \ We do a test by asking for the most used words in ham, after removing the words that most documents used.

In [207]:

```
#Same function as before, upgraded to alter cv
def _trainPredict(mailTrain,mailTest, maxDf, minDf,lang):

  cv = CountVectorizer(min_df=minDf, max_df=maxDf, stop_words=lang)
  mNB = MultinomialNB()
  bNB = BernoulliNB()

  train = cv.fit_transform(mailTrain + spamTrain)

  mNB.fit(train, ['ham']*len(mailTrain) + ['spam']*len(spamTrain))
  bNB.fit(train, ['ham']*len(mailTrain) + ['spam']*len(spamTrain))

  mPredict = mNB.predict( cv.transform(mailTest) )
  bPredict = bNB.predict( cv.transform(mailTest) )

  return (mPredict,bPredict)

#Same function as before, upgraded to alter cv
def _mostFeatures(mailSet, max, maxDf, minDf):
  cv = CountVectorizer(min_df=minDf, max_df=maxDf).fit(mailSet)
  featuresSummed = cv.transform(mailSet).sum(axis=0)
  words_freq = [(word, featuresSummed[0, idx]) for word, idx in cv.vocabulary_.items()]
  words_freq = sorted(words_freq, key = lambda word: word[1], reverse=True)
  return words_freq[0:max]

print("Hams most used words: ",_mostFeatures(hamTrain, 20,0.8,0.2))
```

```
Hams most used words:  [('net', 14806), ('www', 11982), ('and', 11350),
('list', 10444), ('fork', 8458), ('xent', 8272), ('09', 7438), ('sep', 707
0), ('is', 6495), ('10', 6069), ('that', 5807), ('it', 5803), ('mailto', 4
755), ('you', 4722), ('on', 4590), ('admin', 4506), ('jm', 4461), ('this',
4031), ('oct', 3943), ('aug', 3811)]
```

Important to note is that we don't remove the most used words, but the words used in most documents. That is why 'net' and 'www' appears on the list, but the words in-between on the old list 'with' 'for' etc, are removed because they were in more documents.

We can now redo our previous test, but tweak the data to only allow certain words to pass. Which % or amount we should use for our parameters is impossible to say precisely, so we can try and adept to get the best result. \ ! Did do manually testing of finding decent parameters, somewhat pointless showing all the failed/not good enough attempts !

Prevoius Results:

Multinomial - EasyHam: [764 2]\ Bernoulli - EasyHam: [764 2]\ Multinomial - Spam: [ 27 124]\ Bernoulli - Spam: [86 65]

Multinomial - HardHam: [64 11] \ Bernoulli - HardHam: [47 28]\ Multinomial - Spam: [ 1 150]\ Bernoulli - Spam: [ 1 150]

In [208]:

```
#To save space put the 'print results' in a function
def saveSpace(mailTrain,mailTest, maxDf, minDf,lang, pName):

  x = _trainPredict(mailTrain,mailTest, maxDf, minDf, lang)
  hamAmount, spamAmount = np.unique( x[0], return_counts=True)
  print("Multinomial -",pName,":", hamAmount, spamAmount)
  hamAmount, spamAmount = np.unique( x[1], return_counts=True)
  print("Bernoulli -",pName,":", hamAmount, spamAmount)

print("EasyHam E-mails | 90% maxDF | 5 amount minDF")
saveSpace(easyhamTrain,easyhamTest,0.9,5,None,"EasyHam")
saveSpace(easyhamTrain,spamTest,0.9,5,None,"Spam")

print( )

print("HardHam E-mails | 80% maxDF | 5 amount minDF")
saveSpace(hardhamTrain,hardhamTest,0.8,5,None,"HardHam")
saveSpace(hardhamTrain,spamTest,0.8,5,None,"Spam")

print( )

print("Ham E-mails | 80% maxDF | 10 amount minDF")
saveSpace(hamTrain,hamTest,0.8,10,None,"Ham")
saveSpace(hamTrain,spamTest,0.8,10,None,"Spam")

print( )

print("Ham E-mails | 80% maxDF | 10 amount minDF")
saveSpace(hamTrain,hamTest,0.8,10,"english","Ham")
saveSpace(hamTrain,spamTest,0.8,10,"english","Spam")
```

```
EasyHam E-mails | 90% maxDF | 5 amount minDF
Multinomial - EasyHam : ['ham' 'spam'] [760   6]
Bernoulli - EasyHam : ['ham' 'spam'] [762   4]
Multinomial - Spam : ['ham' 'spam'] [  5 146]
Bernoulli - Spam : ['ham' 'spam'] [  2 149]

HardHam E-mails | 80% maxDF | 5 amount minDF
Multinomial - HardHam : ['ham' 'spam'] [58 17]
Bernoulli - HardHam : ['ham' 'spam'] [55 20]
Multinomial - Spam : ['ham' 'spam'] [  1 150]
Bernoulli - Spam : ['ham' 'spam'] [  3 148]

Ham E-mails | 80% maxDF | 10 amount minDF
Multinomial - Ham : ['ham' 'spam'] [829  12]
Bernoulli - Ham : ['ham' 'spam'] [815  26]
Multinomial - Spam : ['ham' 'spam'] [  3 148]
Bernoulli - Spam : ['ham' 'spam'] [  2 149]

Ham E-mails | 80% maxDF | 10 amount minDF
Multinomial - Ham : ['ham' 'spam'] [828  13]
Bernoulli - Ham : ['ham' 'spam'] [816  25]
Multinomial - Spam : ['ham' 'spam'] [  3 148]
Bernoulli - Spam : ['ham' 'spam'] [  2 149]
```

There seems to be a border where we can't improve the Ham E-mails no more without worsening its other ability’s. But what an improvement. EasyHam is much better at differentiating what is spam this time, only classifying 5,2 incorrectly.

All ham has also improved a lot, receiving a total of 841 emails and labelling 12 and 25 as spam. Not perfect, but a lot better.

It might be that we must add more parameters or change or model more fundamentally. But we can do that after we tinker with it in the last exercise.

It isn't any big change when we try it with english filter.

AllHam Before: \ [837 4] - 99.52%\ [840 1] - 99.88%\ [ 17 134] - 88.74%\ [115 36] - 23.84%

AllHam After: \ [828 13] - 98.45%\ [816 25] - 97.02%\ [ 3 148] - 98.01%\ [ 2 149] - 98.67%

# Filter and discussion

## Remove Headers and Footers

Most emails contain a header, or rather a block of code and text before the actual content of the email start. Most emails are then ended with another block of code. We can remove these as they do not serve any purpose. We must first find a header and footer indicator.

There seems to be a gap in most mails where the real message starts or begins to start. There are some outliers but most follow this format.\ Footer is trickier, there is a 'footer' in many mails, but that belongs top the html code of that particular email. Some emails don't have a footer and just ubruptly ends, usually with a message like 'unsubsribe' 'generic market name' etc. In the end did we use the same logic and look for a gap at the bottom of the page, its not ideal, but finding the most common factor footer in all emails is a lab of its own.

In [209]:

```
#Remove parts of string
def removeHeadFoot(dir):
  stringMails = []
  for mail in dir:
    headlessMail = mail.split('\n\n', 1)[-1]
    headlessMail.split('\n\n').pop()
    '\n\n'.join(headlessMail)
    stringMails.append(headlessMail)

  return stringMails

#Reloads the data and filters it
easyHam = removeHeadFoot(createDataDir("easy_ham/"))
hardHam = removeHeadFoot(createDataDir("hard_ham/"))
spam = removeHeadFoot(createDataDir("spam/"))

#Resplits the data
hamTrain, hamTest = train_test_split(easyHam + hardHam, test_size=0.3, random_state=7)
easyhamTrain,easyhamTest = train_test_split(easyHam, test_size=0.3, random_state=7)
hardhamTrain,hardhamTest = train_test_split(hardHam, test_size=0.3, random_state=7)
spamTrain,spamTest = train_test_split(spam, test_size=0.3, random_state=7)


print("EasyHam E-mails | 90% maxDF | 5 amount minDF")
saveSpace(easyhamTrain,easyhamTest,0.9,5,None,"EasyHam")
saveSpace(easyhamTrain,spamTest,0.9,5,None,"Spam")

print( )

print("HardHam E-mails | 80% maxDF | 5 amount minDF")
saveSpace(hardhamTrain,hardhamTest,0.8,5,None,"HardHam")
saveSpace(hardhamTrain,spamTest,0.8,5,None,"Spam")

print( )

print("Ham E-mails | 80% maxDF | 5 amount minDF")
saveSpace(hamTrain,hamTest,0.8,10,"english","Ham")
saveSpace(hamTrain,spamTest,0.8,10,"english","Spam")
```

```
EasyHam E-mails | 90% maxDF | 5 amount minDF
Multinomial - EasyHam : ['ham' 'spam'] [763   3]
Bernoulli - EasyHam : ['ham' 'spam'] [750  16]
Multinomial - Spam : ['ham' 'spam'] [ 29 122]
Bernoulli - Spam : ['ham' 'spam'] [ 42 109]

HardHam E-mails | 80% maxDF | 5 amount minDF
Multinomial - HardHam : ['ham' 'spam'] [55 20]
Bernoulli - HardHam : ['ham' 'spam'] [51 24]
Multinomial - Spam : ['ham' 'spam'] [  4 147]
Bernoulli - Spam : ['ham' 'spam'] [  3 148]

Ham E-mails | 80% maxDF | 5 amount minDF
Multinomial - Ham : ['ham' 'spam'] [829  12]
Bernoulli - Ham : ['ham' 'spam'] [817  24]
Multinomial - Spam : ['ham' 'spam'] [ 12 139]
Bernoulli - Spam : ['ham' 'spam'] [ 37 114]
```

## Header and Footer value

The results get worse when we remove the header and the footer. If we look at 'Ham' above, can we see that it become worse at classifying spam in the spam test, by quite a lot. After testing different values for the parameters was it clear that it was bad to remove the header and the footer. They apparently held valuable information that was important in deciphering if a an E-mail was spam or ham. A wild guess to what that information might be, a certain mail address 'windowz@notScam.com' or a code bit of some kind.

## Split

As always splitting the data is necessary to get a result that we can judge. The important thing to remember is that we can affect the data by splitting the mails inappropriately. A clear example would be if all the 'smart' spam was trained and the test only consisted of 'dumb' spam. The chance of this happening is really low, but the idea is what important, giving our model weird training material that is not close to the real 'target' only messes with it. So, to improve our skip can we try to remove outliers and maybe double check or mail data to see if it is appropriate.

Another factor that can mess with the result is something that was touched on earlier. The amount of spam and ham we feed the model to train can differ quite a lot. We have for example many easyHam mails, but only a few hardHam mails. Feeding our bot with unequally sized training data can alter the result. Considering that Easy is more than 8 times bigger then Hard is it plausible that the model bases more on EasyHam when doing its training. That might be good depending on what the goal of the User is but we are looking for a model that is good for both normal and abnormal ham. Do we want to be really accurate can we try to match the ratio of easy/hard ham to a real life mail inbox.

## Spam Training Only

It would pick up on the wrong details and would start classifying every mail or many as spam. Common words used a lot 'Hey!' 'we' etc would be seen as indicators for a spam mail. Our model similarly to before would favour only 1 side, in this case Spam and would not find clear indicators to what a ham mail is.

# Conclussion

The model is definently not ready for customer use. Simply letting 1 email get discarded or sorted incorrect can doom a mail account. That being said it might be that this model can never get ready. We feed the model a lot of hard to read emails from hard ham that might be to difficult to see a pattern in. Some of those mails also looks like what a normal person would consider spam. The problem might be were we draw the line at spam, not wanting spam but allowing weird hard ham emails. The line is also of course different for every person so creating a 1 model to work for all inboxes is also extremely hard if we want to keep the same level 'weird acceptable mail'.

There are ways of course, given more times and more tricks can one surely make a Naïve Bayes model that sort with 100% accuracy.