In [4]:

**import** pandas **as** pd

sms\_spam **=** pd**.**read\_csv('SMSSpamCollection', sep**=**'\t', header**=None**, names**=**['Label', 'SMS'])

print(sms\_spam**.**shape) sms\_spam**.**head()

Out[4]:

(5572, 2)

**Label SMS**

**0** ham Go until jurong point, crazy.. Available only ...

**1** ham Ok lar... Joking wif u oni...

**2** spam Free entry in 2 a wkly comp to win FA Cup fina...

**3** ham U dun say so early hor... U c already then say...

**4** ham Nah I don't think he goes to usf, he lives aro...

In [5]:

Out[5]:

In [6]:

Label

sms\_spam['Label']**.**value\_counts(normalize**=True**)

ham 0.865937

spam 0.134063

Name: proportion, dtype: float64

*# Randomize the dataset*

data\_randomized **=** sms\_spam**.**sample(frac**=**1, random\_state**=**1)

*# Calculate index for split*

training\_test\_index **=** round(len(data\_randomized) **\*** 0.8)

*# Split into training and test sets*

training\_set **=** data\_randomized[:training\_test\_index]**.**reset\_index(drop**=True**) test\_set **=** data\_randomized[training\_test\_index:]**.**reset\_index(drop**=True**)

print(training\_set**.**shape) print(test\_set**.**shape)

(4458, 2)

(1114, 2)

training\_set['Label']**.**value\_counts(normalize**=True**)

In [7]:

Out[7]:

In [8]:

Out[8]:

In [9]:

Label

ham 0.86541

spam 0.13459

Name: proportion, dtype: float64

test\_set['Label']**.**value\_counts(normalize**=True**)

Label

ham 0.868043

spam 0.131957

Name: proportion, dtype: float64

*# Before cleaning*

training\_set**.**head(3)

|  |  |  |  |
| --- | --- | --- | --- |
| Out[9]: |  | **Label** | **SMS** |
|  | **0** | ham | Yep, by the pretty sculpture |
|  | **1** | ham | Yes, princess. Are you going to make me moan? |
|  | **2** | ham | Welp apparently he retired |

In [10]:

*# After cleaning*

training\_set['SMS'] **=** training\_set['SMS']**.**str**.**replace( '\W', ' ') *# Removes punctuation*

training\_set['SMS'] **=** training\_set['SMS']**.**str**.**lower()

training\_set**.**head(3)

|  |  |  |  |
| --- | --- | --- | --- |
| Out[10]: |  | **Label** | **SMS** |
|  | **0** | ham | yep, by the pretty sculpture |
|  | **1** | ham | yes, princess. are you going to make me moan? |
|  | **2** | ham | welp apparently he retired |

In [11]:

training\_set['SMS'] **=** training\_set['SMS']**.**str**.**split()

vocabulary **=** []

**for** sms **in** training\_set['SMS']:

**for** word **in** sms:

vocabulary**.**append(word)

vocabulary **=** list(set(vocabulary))

In [12]:

Out[12]:

In [13]:

word\_counts\_per\_sms **=** {'secret': [2,1,1],

'prize': [2,0,1],

'claim': [1,0,1],

'now': [1,0,1],

'coming': [0,1,0],

'to': [0,1,0],

'my': [0,1,0],

'party': [0,1,0],

'winner': [0,0,1]

}

word\_counts **=** pd**.**DataFrame(word\_counts\_per\_sms) word\_counts**.**head()

11860

len(vocabulary)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Out[13]: | **secret** | **prize** | **claim** | **now** | **coming** | **to** | **my** | **party** | **winner** |
|  | **0** 2 | 2 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  | **1** 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
|  | **2** 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |

In [18]:

word\_counts\_per\_sms **=** {unique\_word: [0] **\*** len(training\_set['SMS']) **for** unique\_word

**for** index, sms **in** enumerate(training\_set['SMS']):

**for** word **in** sms:

word\_counts\_per\_sms[word][index] **+=** 1

In [17]:

word\_counts **=** pd**.**DataFrame(word\_counts\_per\_sms) word\_counts**.**head()

Out[17]:

**out? rental x49. txt> helen, out-**

**-if**

**dat.. both! comes.. tahan ... driving... concern exp**

**0** 0 0 0 0 0 0 0 0 0 0 ... 0 0

**1** 0 0 0 0 0 0 0 0 0 0 ... 0 0

**2** 0 0 0 0 0 0 0 0 0 0 ... 0 0

**3** 0 0 0 0 0 0 0 0 0 0 ... 0 0

**4** 0 0 0 0 0 0 0 0 0 0 ... 0 0

# 5 rows × 11860 columns

C C

In [16]:

training\_set\_clean **=** pd**.**concat([training\_set, word\_counts], axis**=**1) training\_set\_clean**.**head()

Out[16]:

**Label SMS out? rental x49. txt> helen, out-**

**-if**

**dat.. both! ... driving... concern e**

**0** ham

[yep,, by, the, pretty, sculpture]

0 0 0 0 0 0 0 0 ... 0 0

**1** ham

[yes,, princess., are, you, going, to, make, m...

0

0

0

0

0

0

0

0 ...

0

0

**2** ham

[welp, apparently, he, retired]

0 0 0 0 0 0 0 0 ... 0 0

**3** ham [havent.]

0

0

0

0

0

0

0

0 ...

0

0

**4** ham

[i, forgot, 2, ask, ü,

all, smth..,

there's, a...

0 0 0 0 0 0 0 0 ... 0 0

# 5 rows × 11862 columns

C C

In [19]:

*# Isolating spam and ham messages first*

spam\_messages **=** training\_set\_clean[training\_set\_clean['Label'] **==** 'spam'] ham\_messages **=** training\_set\_clean[training\_set\_clean['Label'] **==** 'ham']

*# P(Spam) and P(Ham)*

p\_spam **=** len(spam\_messages) **/** len(training\_set\_clean) p\_ham **=** len(ham\_messages) **/** len(training\_set\_clean)

*# N\_Spam*

n\_words\_per\_spam\_message **=** spam\_messages['SMS']**.**apply(len) n\_spam **=** n\_words\_per\_spam\_message**.**sum()

|  |  |  |
| --- | --- | --- |
|  |  | *# N\_Ham*  n\_words\_per\_ham\_message **=** ham\_messages['SMS']**.**apply(len) |
|  | n\_ham **=** n\_words\_per\_ham\_message**.**sum() |
|  | *# N\_Vocabulary*  n\_vocabulary **=** len(vocabulary) |
|  | *# Laplace smoothing*  alpha **=** 1 |
|  |  |
| In | [20]: | *# Initiate parameters*  parameters\_spam **=** {unique\_word:0 **for** unique\_word **in** vocabulary} |
|  | | parameters\_ham **=** {unique\_word:0 **for** unique\_word **in** vocabulary}  *# Calculate parameters*  **for** word **in** vocabulary:  n\_word\_given\_spam **=** spam\_messages[word]**.**sum() *# spam\_messages already defined*  p\_word\_given\_spam **=** (n\_word\_given\_spam **+** alpha) **/** (n\_spam **+** alpha**\***n\_vocabulary) parameters\_spam[word] **=** p\_word\_given\_spam  n\_word\_given\_ham **=** ham\_messages[word]**.**sum() *# ham\_messages already defined*  p\_word\_given\_ham **=** (n\_word\_given\_ham **+** alpha) **/** (n\_ham **+** alpha**\***n\_vocabulary) parameters\_ham[word] **=** p\_word\_given\_ham |

In [21]:

**import** re

**def** classify(message): '''

message: a string '''

message **=** re**.**sub('\W', ' ', message) message **=** message**.**lower()**.**split()

p\_spam\_given\_message **=** p\_spam p\_ham\_given\_message **=** p\_ham

**for** word **in** message:

**if** word **in** parameters\_spam:

p\_spam\_given\_message **\*=** parameters\_spam[word]

**if** word **in** parameters\_ham:

p\_ham\_given\_message **\*=** parameters\_ham[word]

print('P(Spam|message):', p\_spam\_given\_message) print('P(Ham|message):', p\_ham\_given\_message)

**if** p\_ham\_given\_message **>** p\_spam\_given\_message: print('Label: Ham')

**elif** p\_ham\_given\_message **<** p\_spam\_given\_message: print('Label: Spam')

**else**:

print('Equal proabilities, have a human classify this!')

In [22]:

classify('WINNER!! This is the secret code to unlock the money: C3421.')

P(Spam|message): 1.1680023632078457e-26 P(Ham|message): 6.088544142463393e-28

Label: Spam

In [23]:

classify("Sounds good, Tom, then see u there")

P(Spam|message): 2.234299283967944e-26 P(Ham|message): 8.376346103813855e-22 Label: Ham

In [24]:

**def** classify\_test\_set(message): '''

message: a string '''

message **=** re**.**sub('\W', ' ', message) message **=** message**.**lower()**.**split()

p\_spam\_given\_message **=** p\_spam p\_ham\_given\_message **=** p\_ham

**for** word **in** message:

**if** word **in** parameters\_spam:

p\_spam\_given\_message **\*=** parameters\_spam[word]

**if** word **in** parameters\_ham:

p\_ham\_given\_message **\*=** parameters\_ham[word]

**if** p\_ham\_given\_message **>** p\_spam\_given\_message:

**return** 'ham'

**elif** p\_spam\_given\_message **>** p\_ham\_given\_message:

**return** 'spam'

**else**:

**return** 'needs human classification'

In [25]:

test\_set['predicted'] **=** test\_set['SMS']**.**apply(classify\_test\_set) test\_set**.**head()

Out[25]:

**Label SMS predicted**

**0** ham Later i guess. I needa do mcat study too. ham

**1** ham But i haf enuff space got like 4 mb... ham

**2** spam Had your mobile 10 mths? Update to latest Oran... spam

**3** ham All sounds good. Fingers . Makes it difficult ... ham

**4** ham All done, all handed in. Don't know if mega sh... ham

In [26]:

correct **=** 0

total **=** test\_set**.**shape[0]

**for** row **in** test\_set**.**iterrows(): row **=** row[1]

**if** row['Label'] **==** row['predicted']: correct **+=** 1

print('Correct:', correct)

print('Incorrect:', total **-** correct) print('Accuracy:', correct**/**total)

Correct: 1090

Incorrect: 24

Accuracy: 0.9784560143626571

In [ ]: