Building a Spam Filter with Naive Bayes

n this project, we're going to build a spam filter for SMS messages using the multinomial Naive Bayes algorithm. Our goal is to write a program that classifies new messages with an accuracy greater than 80% — so we expect that more than 80% of the new messages will be classified correctly as spam or ham (non-spam).

To train the algorithm, we'll use a dataset of 5,572 SMS messages that are already classified by humans. The dataset was put together by Tiago A. Almeida and José María Gómez Hidalgo, and it can be downloaded from the The UCI Machine Learning Repository. The data collection process is described in more details on this page, where you can also find some of the papers authored by Tiago A. Almeida and José María Gómez Hidalgo.

Exploring the Dataset

We'll now start by reading in the dataset

In [72]:
```
import pandas as pd
import numpy as np
spam_sms = pd.read_csv("SMSSpamCollection", sep="\t",names=['Label','SMS'], header=None)
spam_sms
```

Out[72]:

	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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?

	Label	SMS
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [73]:
```
spam_sms['Label'].unique()
```

Out[73]:
```
array(['ham', 'spam'], dtype=object)
```

In [74]:
```
spam_sms['Label'].value_counts(normalize=True)
```

Out[74]:
```
ham      0.865937
spam     0.134063
Name: Label, dtype: float64
```

It is obvious we see that about 87% of the messages are ham, and 13% are spam. This sample looks representative, since in practice most messages that people receive are ham.

Training and Testing the dataset

We're now going to split our dataset into a training and a test set, where the training set accounts for 80% of the data, and the test set for the remaining 20%.

In [75]:
```
# Randomize the dataset
data_randomized =  spam_sms.sample(frac=1,random_state=1)# Use the frac=1 parameter to randomize the entire dataset.
# Use the random_state=1 parameter to make sure your results are reproducible.
```

In [76]:
```
# Calculate index for split
training_test_index = round(len(data_randomized) * 0.8)

# Training/Test split

training_set = data_randomized[:training_test_index].reset_index(drop=True)
training_set
```

Out[76]:

	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
3	ham	Havent.
4	ham	I forgot 2 ask ü all smth.. There's a card on ...
...	...	...
4453	ham	Sorry, I'll call later in meeting any thing re...
4454	ham	Babe! I fucking love you too !! You know? Fuck...
4455	spam	U've been selected to stay in 1 of 250 top Bri...
4456	ham	Hello my boytoy ... Geeee I miss you already a...
4457	ham	Wherre's my boytoy ? :-(

4458 rows × 2 columns

In [77]:

```
testing_set = data_randomized[training_test_index:].reset_index(drop=True)
testing_set
```

Out[77]:

	Label	SMS
0	ham	Later i guess. I needa do mcat study too.
1	ham	But i haf enuff space got like 4 mb...
2	spam	Had your mobile 10 mths? Update to latest Oran...
3	ham	All sounds good. Fingers . Makes it difficult ...
4	ham	All done, all handed in. Don't know if mega sh...
...	...	...
1109	ham	We're all getting worried over here, derek and...
1110	ham	Oh oh... Den muz change plan liao... Go back h...

	Label	SMS
1111	ham	CERI U REBEL! SWEET DREAMZ ME LITTLE BUDDY!! C...
1112	spam	Text & meet someone sexy today. U can find a d...
1113	ham	K k:) sms chat with me.

1114 rows × 2 columns

Find the percentage of spam and ham in both the training and the test set

We'll now analyze the percentage of spam and ham messages in the training and test sets. We expect the percentages to be close to what we have in the full dataset, where about 87% of the messages are ham, and the remaining 13% are spam.

In [78]:
```
testing_set["Label"].value_counts(normalize=True)
```

Out[78]:
```
ham     0.868043
spam    0.131957
Name: Label, dtype: float64
```

In [79]:
```
training_set["Label"].value_counts(normalize=True)
```

Out[79]:
```
ham     0.86541
spam    0.13459
Name: Label, dtype: float64
```

The results look good! We'll now move on to cleaning the dataset

Cleaning Data

To calculate all the probabilities required by the algorithm, we'll first need to perform a bit of data cleaning to bring the data in a format that will allow us to extract easily all the information we need.

Letter Case and Punctuation

We'll begin with removing all the punctuation and bringing every letter to lower case from SMS column.

In [80]:
```
# Cleaning  data by removing punctuation

training_set['SMS'] = training_set['SMS'].str.replace('\W', ' ')
# transform every letter in every word to lower case.
training_set['SMS'] = training_set['SMS'].str.lower()
training_set.head()
```

Out[80]:

	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
3	ham	havent
4	ham	i forgot 2 ask ü all smth there s a card on ...

Creating the Vocabulary

Let's now move to creating the vocabulary, which in this context means a list with all the unique words in our training set.

In [81]:
```
training_set['SMS'] = training_set['SMS'].str.split()
training_set.head(10)
```

Out[81]:

	Label	SMS
0	ham	[yep, by, the, pretty, sculpture]
1	ham	[yes, princess, are, you, going, to, make, me,...
2	ham	[welp, apparently, he, retired]
3	ham	[havent]
4	ham	[i, forgot, 2, ask, ü, all, smth, there, s, a,...
5	ham	[ok, i, thk, i, got, it, then, u, wan, me, 2, ...
6	ham	[i, want, kfc, its, tuesday, only, buy, 2, mea...

	Label	SMS
7	ham	[no, dear, i, was, sleeping, p]
8	ham	[ok, pa, nothing, problem]
9	ham	[ill, be, there, on, lt, gt, ok]

In [82]:
```
Vocabulary = []
for sms in training_set['SMS']:
    for word in sms:
        Vocabulary.append(word)
```

In [83]:
```
# Transform the vocabulary list into a set. This will remove the duplicates from the Vocabulary list.
Vocabulary = set(Vocabulary)
# Transform the vocabulary set back into a list
Vocabulary = list(Vocabulary )
len(Vocabulary)
```

Out[83]: 7783

Final Training Set

We're now going to use the vocabulary we just created to make the data transformation we want.

In [84]:
```
word_counts_per_sms = {unique_word: [0] * len(training_set['SMS']) for unique_word in Vocabulary}
```

In [85]:
```
for index, sms in enumerate(training_set['SMS']):
    for word in sms:
        word_counts_per_sms[word][index] += 1
```

In [24]:
```
word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head(10)
```

Out[24]:

	relaxing	fresh	snowboarding	freaking	hangin	vilikkam	choose	agent	smart	points	...	50perwksub	872	kudi	brilliantly	feels	09061743
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

	relaxing	fresh	snowboarding	freaking	hangin	vilikkam	choose	agent	smart	points	...	50perwksub	872	kudi	brilliantly	feels	090617438
1	0	0	0	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	0	0	0	
3	0	0	0	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	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

10 rows × 7783 columns

In [26]:

```
training_data = pd.concat([training_set,word_counts], axis=1)
training_data.head()
```

Out[26]:

	Label	SMS	relaxing	fresh	snowboarding	freaking	hangin	vilikkam	choose	agent	...	50perwksub	872	kudi	brilliantly	feels	09061
0	ham	[yep, by, the, pretty, sculpture]	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1	ham	[yes, princess, are, you, going, to, make, me,...	0	0	0	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	0	0	0	
3	ham	[havent]	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

	Label	SMS	relaxing	fresh	snowboarding	freaking	hangin	vilikkam	choose	agent	...	50perwksub	872	kudi	brilliantly	feels	09061
4	ham	[i, forgot, 2, ask, ü, all, smth, there, s, a,...	0	0	0	0	0	0	0	0	...	0	0	0	0	0	

5 rows × 7785 columns

Firstly, calculating Constants

We're now done with cleaning the training set, and we can begin creating the spam filter. The Naive Bayes algorithm will need to answer these two probability questions to be able to classify new messages:

$$P(Spam|w_1, w_2, \ldots, w_n) \propto P(Spam) \cdot \prod_{i=1}^{n} P(w_i|Spam)$$

$$P(Ham|w_1, w_2, \ldots, w_n) \propto P(Ham) \cdot \prod_{i=1}^{n} P(w_i|Ham)$$

Also, to calculate P(wi|Spam) and P(wi|Ham) inside the formulas above, we'll need to use these equations:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + \alpha \cdot N_{Vocabulary}}$$

Some of the terms in the four equations above will have the same value for every new message. We can calculate the value of these terms once and avoid doing the computations again when a new messages comes in. Below, we'll use our training set to calculate:

```
P(Spam) and P(Ham)
NSpam, NHam, NVocabulary
```

We'll also use Laplace smoothing and set $\alpha = 1$.

In [55]:
```python
# Spliting dataset into Spam and Ham.
spam_message = training_data[training_set["Label"]=="spam"]
ham_message = training_data[training_set["Label"] =='ham']

# Calculating P(Spam) and P(Ham).
P_spam = len(spam_message)/ len(training_data)
P_Ham = len(ham_message)/ len(training_data)
```

In [56]:
```python
# Calculating N_Spam.
N_Spam = spam_message['SMS'].apply(len).sum()
N_Ham = ham_message['SMS'].apply(len).sum()
N_Vocabulary = len(Vocabulary)
print(N_Spam,'\n',N_Ham,'\n',N_Vocabulary, sep="")

# Laplace Smoothing
alpha=1
```

```
15190
57237
7783
```

Calculating Parameters

Now that we have the constant terms calculated above, we can move on with calculating the parameters $P(w_i|Spam)$ and $P(w_i|Ham)$. Each parameter will thus be a conditional probability value associated with each word in the vocabulary.

The parameters are calculated using the formulas:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + \alpha \cdot N_{Vocabulary}}$$

In [90]:

```
# Initiate parameters
parameters_spam =  {new_word:0 for new_word in Vocabulary}
parameters_Ham = {new_word:0 for new_word in Vocabulary}

for word in Vocabulary:
    n_word_given_spam =  spam_message[word].sum() # spam_messages already defined in a cell above.
    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_message[word].sum()
    p_word_given_Ham = ( n_word_given_Ham +alpha )/ (N_Ham + alpha* N_Vocabulary)
    parameters_Ham[word] = p_word_given_Ham
```

Classifying A New Message

Now that we have all our parameters calculated, we can start creating the spam filter. The spam filter can be understood as a function that:

- Takes in as input a new message (w1, w2, ..., wn).
- Calculates P(Spam|w1, w2, ..., wn) and P(Ham|w1, w2, ..., wn).
- Compares the values of P(Spam|w1, w2, ..., wn) and P(Ham|w1, w2, ..., wn), and:
  - If P(Ham|w1, w2, ..., wn) > P(Spam|w1, w2, ..., wn), then the message is classified as ham.
  - If P(Ham|w1, w2, ..., wn) < P(Spam|w1, w2, ..., wn), then the message is classified as spam.
  - If P(Ham|w1, w2, ..., wn) = P(Spam|w1, w2, ..., wn), then the algorithm may request human help.

In [92]:

```
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_spam_given_message > p_ham_given_message:
        print("Label : Spam")
    if p_ham_given_message > p_spam_given_message:
        print("Label : Ham")
    else:
        print('Equal proabilities, have a human classify this!')
```

In [93]:
```
classify('WINNER!! This is the secret code to unlock the money: C3421.')
```

```
P(Spam|message): 1.3481290211300841e-25
P(Ham|message): 1.9368049028589875e-27
Label : Spam
Equal proabilities, have a human classify this!
```

In [94]:
```
classify("Sounds good, Tom, then see u there")
```

```
P(Spam|message): 2.4372375665888117e-25
P(Ham|message): 3.687530435009238e-21
Label : Ham
```

Measuring the Spam Filter's Accuracy

The two results above look promising, but let's see how well the filter does on our test set, which has 1,114 messages.

We'll start by writing a function that returns classification labels instead of printing them.

In [128…
```
def classify_test_data(message):
    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_spam_given_message > p_ham_given_message:
        return("spam")
    elif p_ham_given_message > p_spam_given_message:
        return("ham")
    else:
        print('Equal proabilities, have a human classify this!')
```

Now that we have a function that returns labels instead of printing them, we can use it to create a new column in our test set.

In [129…
```
testing_set["Predicted"] = testing_set['SMS'].apply(classify_test_data)
testing_set.head()
```

```
Equal proabilities, have a human classify this!
```

Out[129…

	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

Now we can compare the predicted values with the actual values to measure how good our spam filter is with classifying new messages. To make the measurement, we'll use accuracy as a metric:

Accuracy = number of correctly classified messages / total number of classified messages

In [133…

```
correct = 0
total = len(testing_set)

for row in testing_set.iterrows():
    row = row[1]
    if row['Label'] == row['Predicted']:
        correct += 1

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

```
Correct: 1100
Incorrect: 14
Accuracy: 0.9874326750448833
```

The accuracy is close to 98.74%, which is really good. Our spam filter looked at 1,114 messages that it hasn't seen in training, and classified 1,100 correctly.