1. Read the following instructions and try to answer questions.
2. I strongly recommend to use Google Colab interface to implement and run your Python cod e.
   A. https://colab.google/ (Colab website, Use your Google account to join and sign in.)
   B. Get started with Google Colaboratory https://youtu.be/inN8seMm7UI?si=KUdMKhWLjDqzzhh8
   C. You may find many useful information about Colab on the internet. (Maybe you need t o get used to use the Jupyter Notebook interface.)
3. Our goal is to make a spam message filter according to specific words.
4. Data preparation

```
import pandas as pd
import urllib.request

urllib.request.urlretrieve("https://raw.githubusercontent.com/ukairia777
/tensorflow-nlp-
tutorial/main/10.%20RNN%20Text%20Classification/dataset/spam.csv",
filename="spam.csv")
df = pd.read_csv('spam.csv', encoding='latin1', usecols=[0, 1])

df.columns = ['Label', 'Message']

print(df.head())
```

```
  Label                                            Message
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...
```

   A. You can download the dataset directly in you Python code like above.
   B. Use `print(df.head())` function to see the first few lines of the data.
5. Function to split messages into words

```
import re

# Function to split messages into words
def tokenize(message):
    message = message.lower()  # Convert to lowercase
    words = re.findall(r'\b\w+\b', message)  # Extract words
    return words
```

6. Separate spam and non-spam messages

```
# Separate spam and non-spam messages
spam_messages = df[df['Label'] == 'spam']['Message']
non_spam_messages = df[df['Label'] == 'ham']['Message']
```

7. Count the number of spam messages

```
# Count the number of spam messages
spam_count = spam_messages.shape[0]

# Print the result
print(f"Total number of spam messages: {spam_count}")
```

```
Total number of spam messages: 747
```

8. **Question 1**: How to get the number of non-spam(ham) messages?

Your answer here

```
non_spam_count = non_spam_messages.shape[0]
print(f"Total number of non_spam messages: {non_spam_count}")
```

9. Calculate word frequency

```
from collections import Counter

# Calculate word frequency
spam_words = Counter()
non_spam_words = Counter()

for message in spam_messages:
    spam_words.update(tokenize(message))

for message in non_spam_messages:
    non_spam_words.update(tokenize(message))
```

10. Print the most common words

```
# Print the most common words
print("Most common words in spam messages:")
print(spam_words.most_common(30))

print("\nMost common words in non-spam messages:")
print(non_spam_words.most_common(30))
```

```
Most common words in spam messages:
[('to', 688), ('a', 377), ('call', 355), ('â', 299), ('you', 297), ('your', 264), ('free', 224), ('2', 206), ('the', 206),

Most common words in non-spam messages:
[('i', 2940), ('you', 1943), ('to', 1554), ('the', 1122), ('a', 1056), ('u', 1018), ('and', 857), ('in', 818), ('me', 772),
```

11. **Question 2**: Do you think is it meaningful to use the most common words in each message category to make a spam filter? Explain why.

Your answer here

No, it's meaningless. Because a comparison of the 30 most common words in spam and the 30 most common words in non-spam messages resulted in 12 identical words. Then, if a sentence is made up of those 12 words, it is impossible to distinguish whether it is a spam message or a

non-spam message. Therefore, this method has limitations in the way it tries to make a sample filter.

12. Identify words that are more frequent in spam than in non-spam

```
# Identify words that are more frequent in spam than in non-spam
# Compare the frequency of each word in spam and non-spam
unique_words = set(list(spam_words.keys()) +
list(non_spam_words.keys()))
spam_dominant_words = {}

for word in unique_words:
    spam_freq = spam_words[word]
    non_spam_freq = non_spam_words[word]
    # Select the word if it is significantly more frequent in spam
    if spam_freq > non_spam_freq * 2:  # This ratio can be adjusted.
        spam_dominant_words[word] = spam_freq

# Print the results
print("Words more frequent in spam than in non-spam:")
for word, freq in sorted(spam_dominant_words.items(), key=lambda x:
x[1], reverse=True):
    print(f"{word}: {freq}")
```

```
Words more frequent in spam than in non-spam:
å: 299
free: 224
txt: 163
mobile: 127
stop: 121
claim: 113
reply: 104
www: 98
```

13. **Question 3**: If you want to Identify words that are a way more frequent in spam than in no n-spam, how are you going to modify the previous code?

Your answer here

First of all, the probabilities of the word appearing in spam messages and non-spam messages were calculated as ‘p_w’ and ‘q_w’.

If the non-spam messages do not contain a word, then the value of ‘q_w’ is 0, then the probability that the incoming message based on BAYES’ Theorem is spam is 1. I don't think it's meaningless if the probability is 1. So I use the condition “q_w!=0” to exclude the condition that the probability is 1.

```
unique_words = set(list(spam_words.keys()) +
list(non_spam_words.keys()))

spam_dominant_words = {}
for word in unique_words:
    spam_freq = spam_words[word]#the frequency of word in spam message
    non_spam_freq = non_spam_words[word]#the frequency of word in non-
spam message
```

```
    p_w = spam_freq / spam_count #the probability that the spam message
contains the word
    q_w = non_spam_freq / non_spam_count #the probability that the non-
spam message contains the word


    if p_w > q_w * 2 and q_w !=0 : # This ratio can be adjusted.
        spam_dominant_words[word] = spam_freq
for word, freq in sorted(spam_dominant_words.items(), key=lambda x:
x[1], reverse=True):
    print(f"{word}: {freq}")
```

14. **Question 4**: How many words do you want to pick to make your spam filter? What are the words?

Your answer here

Use the BAYES' theorem and then pick words through the code. Combine words just before the probability becomes 1 to create a new filter of your own. .
The number of words in my spam filter varies depending on how I adjust the ratio of the condition in question 3.

If p_w > q_w * 1 and q_w!=0. -> the number of words : 20
{to, a, call, å, your, free, 2, the, for, now, or, u, txt, is, ur, on, 4, have, from , mobile}

If p_w > q_w * 2 and q_w!=0. -> the number of words : 17
{to, a, call, å, your, free, 2, for, now, or, txt, ur, on, 4, from , mobile, text}

If p_w > q_w * 10 and q_w!=0. -> the number of words : 9
{å, free, txt, mobile, text, stop, 1, reply, www}

```
p_num = 0
q_num = 0
p_w = 1.0
q_w = 1.0
p_total = 1.0
q_total = 1.0
r_w = 0.0
spam_my_words={}
for word, freq in sorted(spam_dominant_words.items(), key=lambda x:
x[1], reverse=True):
  p_num = spam_words[word] #the frequency of word in spam message
  q_num = non_spam_words[word]#the frequency of word in non-spam message
```

```
  if q_num !=0:
    p_w = p_num / spam_count
    q_w = q_num / non_spam_count
    p_total *= p_w
    q_total *= q_w
    r_w = p_total / (p_total + q_total) #Generalized BAYES' Theorem
    if(r_w<1.0):
      spam_my_words[word] = freq
    else :
      break
  else:
    continue
print(len(spam_my_words))
print(spam_my_words)
```

15. Refer to the following codes if you think you need.

```
# Function to check if a word is in a message
def is_word_in_message(word, message):
    message = message.lower()  # Convert to lowercase
    words = set(re.findall(r'\b\w+\b', message))  # Extract words and
convert to a set
    return word in words
```

```
# Specify the word to search for
search_word = 'free'.lower()  # Replace with your specific word

# Filter spam messages
spam_messages = df[df['Label'] == 'spam']['Message']

# Count how many spam messages contain the word
count = sum(is_word_in_message(search_word, message) for message in
spam_messages)

# Print the result
print(f"The word '{search_word}' appears in {count} spam messages.")
```

16. **Question 5**: Using the multi-words filtering method covered in the class, implement a function that determines if a given input message is a spam or not. (Suppose the decision threshold is set to 90%.)

Your answer here

```
def create_word_frequency_dict(words): #Remove the duplicated words and
calculate the number of words
    word_frequency_dict = {}
```

```
    for word in words:
        if word in word_frequency_dict:
            word_frequency_dict[word] += 1
        else:
            word_frequency_dict[word] = 1

    return word_frequency_dict


def make_filter():
  p_num = 0
  q_num = 0
  p_w = 0.0
  q_w = 0.0
  p_total = 1.0
  q_total = 1.0
  r_w = 0.0
  spam_my_words={}
  for word, freq in sorted(spam_dominant_words.items(), key=lambda x:
x[1], reverse=True):
    p_num = spam_words[word] #the frequency of word in spam message
    q_num = non_spam_words[word] #the frequency of word in non-spam
message
    if q_num !=0:
      p_w = p_num / spam_count
      q_w = q_num / non_spam_count
      p_total *= p_w
      q_total *= q_w
      r_w = p_total / (p_total + q_total) #Generalized BAYES' Theorem
      if(r_w<1.0):
        spam_my_words[word] = freq
      else :
        break

    else:
      continue

  return spam_my_words

def is_spam(input_message):
  spam_my_words = make_filter() #make your spam-filter based on BAYES'
Theorem

  p_num = 0
```

```
  q_num = 0
  p_w = 0.0
  q_w = 0.0
  p_total = 1.0
  q_total = 1.0
  r_w = 0.0

  for word in word_frequency_dict.keys():
    if word in spam_my_words:
      p_num = sum(is_word_in_message(word, message) for message in
spam_messages)
      q_num = sum(is_word_in_message(word, message) for message in
non_spam_messages)
      p_w = p_num / spam_count
      q_w = q_num / non_spam_count
      p_total *= p_w
      q_total *= q_w
    else:
      continue

    r_w = p_total / (p_total + q_total) #Generalized BAYES' Theorem
    print(word, p_total,q_total, r_w)
  return r_w > 0.9


input_message = input() #enter your specific message

# Tokenize the input message
tokenized_words = tokenize(input_message)

# Create a dictionary of word frequencies
word_frequency_dict = create_word_frequency_dict(tokenized_words)

if(is_spam(input_message)):
  print(input_message+" is spam message.")
else:
  print(input_message+" is not spam message.")
```

The result :
Input message : “I'm at work. Please call”
word = “call”
p_total = 0.4390896921017403

```
q_total = 0.04580310880829015
r_w = 0.9055397219296133

I'm at work. Please call
call 0.4390896921017403 0.04580310880829015 0.9055397219296133
I'm at work. Please call is spam message.
```

17. **Question 6**: Change the number of words you used and test the performance.

Your answer here
I tested through two scenarios in three situations with different word counts.
Spam messages and non-spam messages given as the first data were tested to see if they were recognized correctly again.

As the number of words decreased, the number of messages misidentified as non-spam increased, but the number of messages misidentified as spam decreased.

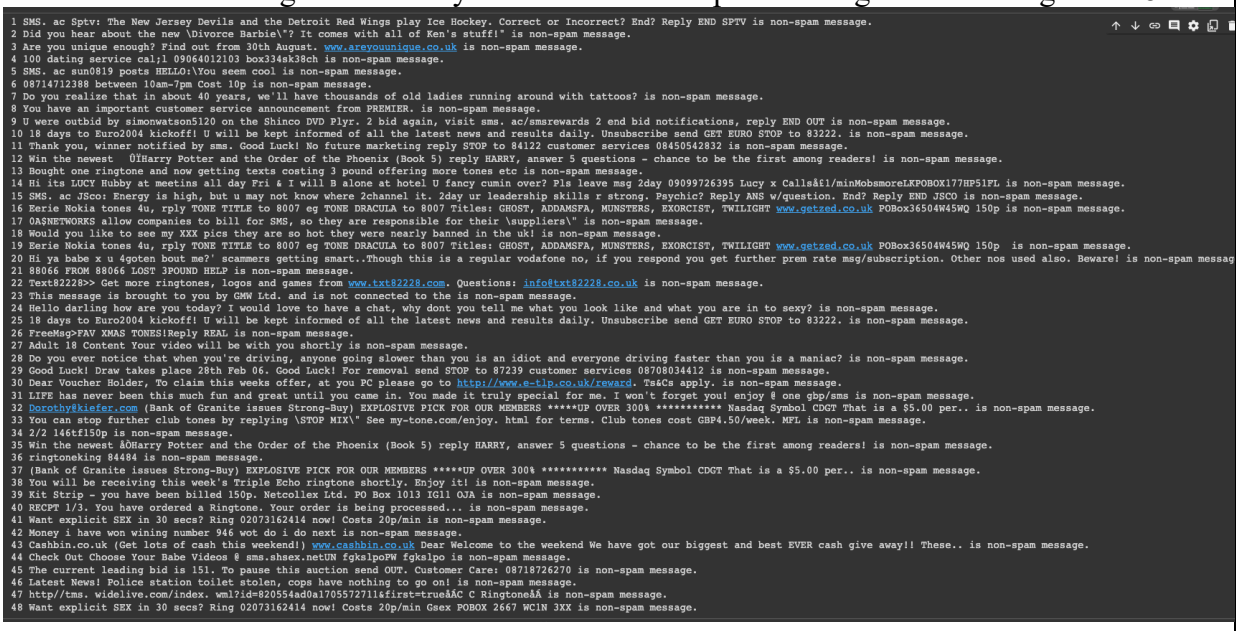
Scenario 1: Check the number of messages that are spam messages based on the number of words but are output as non-spam messages

Using the next code

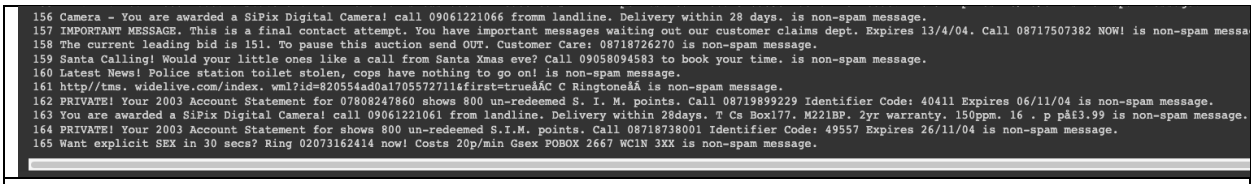
```
count=1
for input_message in spam_messages:
  # Tokenize the input message
  tokenized_words = tokenize(input_message)
  # Create a dictionary of word frequencies
  word_frequency_dict = create_word_frequency_dict(tokenized_words)


  if(is_spam(input_message)==False):
    print(count, input_message+" is non-spam message.")
    count+=1
```

If p_m > q_m * 1 and p_m > 0.1: -> The number of spam_mywords: 20
The number of messages incorrectly identified as non-spam messages when doing this: 47

If p_m > q_m * 2 and p_m > 0.1: -> The number of spam_dominant_words: 17
The number of messages incorrectly identified as non-spam messages when doing this: 48



If p_m > q_m * 10 and p_m > 0.1: -> The number of spam_dominant_words: 9
The number of messages incorrectly identified as non-spam messages when doing this: 165

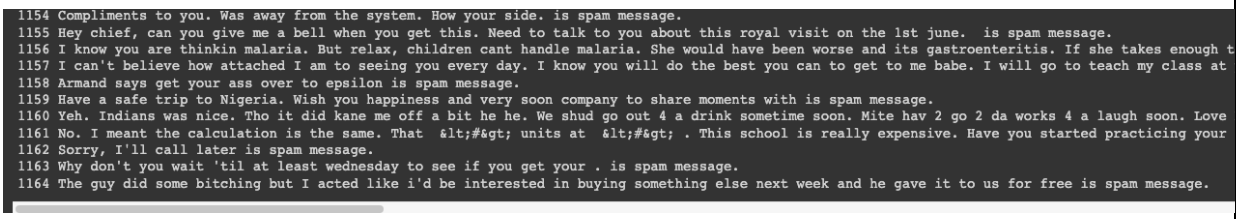


Scenario 2: Check the number of messages that are non-spam messages based on the number of words but are output as spam messages

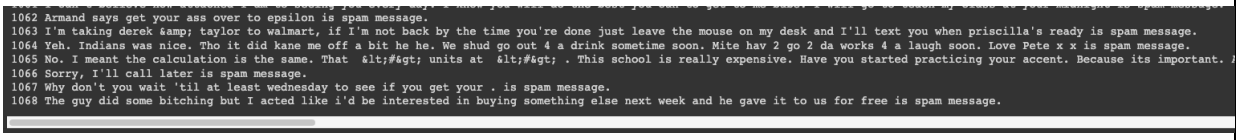
Using the next code

```
count=1
for input_message in non_spam_messages:
  # Tokenize the input message
  tokenized_words = tokenize(input_message)
  # Create a dictionary of word frequencies
  word_frequency_dict = create_word_frequency_dict(tokenized_words)


  if(is_spam(input_message)==True):
    print(count, input_message+" is spam message.")
    count+=1
```

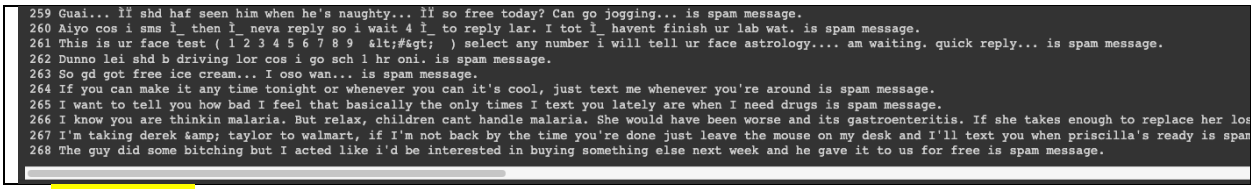
If p_m > q_m * 1 and p_m > 0.1: -> The number of spam_dominant_words: 20
The number of messages incorrectly identified as spam messages when doing this: 1164



If p_m > q_m * 2 and p_m > 0.1: -> The number of spam_dominant_words: 17
The number of messages incorrectly identified as spam messages when doing this: 1068



If p_m > q_m * 10 and p_m > 0.1: -> The number of spam_dominant_words: 9
The number of messages incorrectly identified as spam messages when doing this: 268



18. **Question 7**: Analyze(discuss) the advantages and disadvantages of the spam filter method implemented through this assignment.

Your answer here

[Advantages]
1. Identification of Spam-Dominant Words: Your approach identifies words that significantly appear more frequently in spam messages. This can be effective in capturing distinctive characteristics of spam.

2. Bayesian Probability Integration: The incorporation of Bayesian probability provides a principled way to update probabilities based on evidence. This allows the filter to adapt and potentially improve over time.

3. Customization: The filter allows customization of thresholds and conditions for considering a word as spam-dominant. This flexibility allows for tuning based on the characteristics of the data.

4. Iterative Bayesian Probability Calculation: The iterative calculation of Bayesian probabilities for each word, considering both spam and non-spam messages, provides a dynamic and evolving approach to spam detection.

[Disadvantages]
1. Stopping Condition: The decision to stop updating probabilities when the Bayesian probability (r_w) becomes greater than 1.0 may lead to premature termination. It could potentially miss opportunities for refining the filter further.

2. Sensitivity to Thresholds: The effectiveness of the filter may be sensitive to the chosen thresholds and conditions for considering a word as spam-dominant. Fine-tuning these parameters could be challenging.

3. Word Frequency Calculation: The accuracy of the filter heavily depends on the accurate calculation of word frequencies. Errors in tokenization or frequency counting could lead to incorrect probabilities.

4. Limited Feature Set: The filter relies primarily on word frequencies and Bayesian probabilities. Incorporating additional features or more sophisticated machine learning techniques could enhance the filter's performance.

5. Assumption of Independence: The assumption of independence among words may not always hold true in natural language. This can limit the effectiveness of the filter, especially in capturing context or relationships between words.

6. Limited Generalization: Depending on the training data, the filter may have limitations in generalizing to diverse and unseen messages. Overfitting to the training set could be a concern.