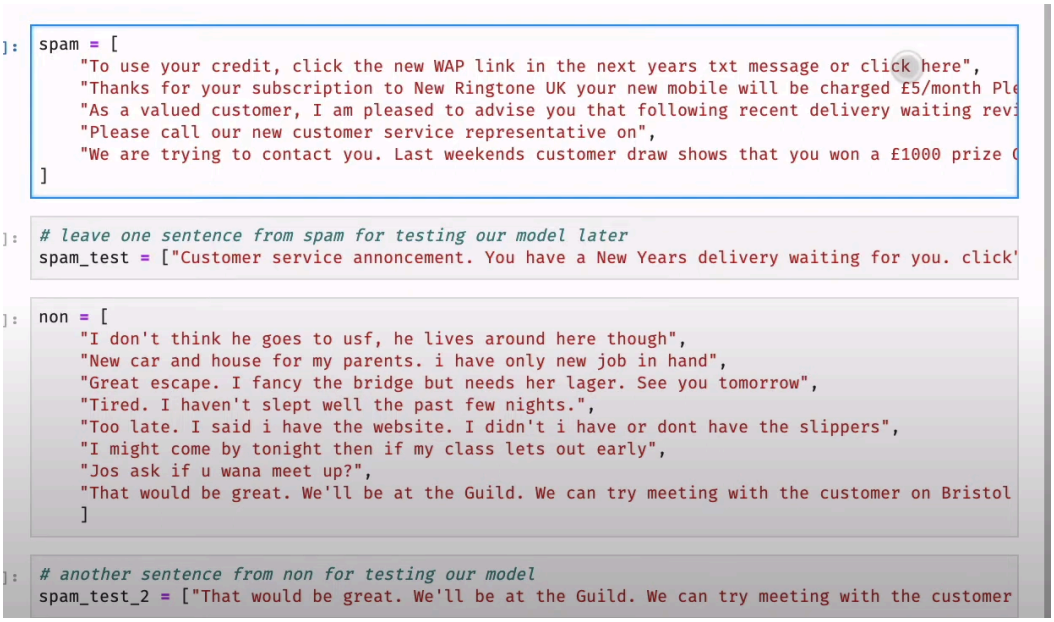
Probability And Statistic Lecture # 13

- **Spam Detection using Naive Bayes**
    - Taking a real set dataset from real world and keeping it real and keeping it short , first we work on a smaller dataset then we apply it on a bigger one
    - 

```
spam = [
    "To use your credit, click the new WAP link in the next years txt message or click here",
    "Thanks for your subscription to New Ringtone UK your new mobile will be charged £5/month Ple
    "As a valued customer, I am pleased to advise you that following recent delivery waiting revi
    "Please call our new customer service representative on",
    "We are trying to contact you. Last weekends customer draw shows that you won a £1000 prize (
]
```

```
# leave one sentence from spam for testing our model later
spam_test = ["Customer service annoncement. You have a New Years delivery waiting for you. click'
```

```
non = [
    "I don't think he goes to usf, he lives around here though",
    "New car and house for my parents. i have only new job in hand",
    "Great escape. I fancy the bridge but needs her lager. See you tomorrow",
    "Tired. I haven't slept well the past few nights.",
    "Too late. I said i have the website. I didn't i have or dont have the slippers",
    "I might come by tonight then if my class lets out early",
    "Jos ask if u wana meet up?",
    "That would be great. We'll be at the Guild. We can try meeting with the customer on Bristol
    ]
```

```
# another sentence from non for testing our model
spam_test_2 = ["That would be great. We'll be at the Guild. We can try meeting with the customer
```

        - This is taken from DATASET repository of real world called UCI
    - Spam is list of words with spam
    - Spam_test is a list of words with sentence that we are gonna test
    - Non is list of words with non spam words
    - Spam_test_2 is list of words that dont have spam messages , it is for testing our model

- **Data Preprocessing**

```
[ ]: # !pip install gensim

[ ]: from gensim.parsing.preprocessing import remove_stopwords
     from gensim.parsing.porter import PorterStemmer
     from gensim.utils import tokenize

[ ]: test_sentence = non[4]
     test_sentence = non[5]
     test_sentence = spam[1]

     print(test_sentence)

     removed_stops = remove_stopwords(test_sentence)
     print(removed_stops)

     p = PorterStemmer()
     stemmed = p.stem(removed_stops)
     print(stemmed)

     tokens = tokenize(stemmed)
     print(list(tokens))
```

- Stopwords : Those are the words that create complications in our ANALYSIS which is why we remove them from our dataset and make it less complicated
  - So we use remove_stopwords from gensim.parsing.preprocessing library
  - For example , call and calling is considered the same so we will merge it with the help of remove_stopwords
- Stemming : It is the process of considering two similar words as the same words

```
[8]: from gensim.parsing.preprocessing import remove_stopwords
     from gensim.parsing.porter import PorterStemmer
     from gensim.utils import tokenize

[10]: test_sentence = non[4]
      test_sentence = non[5]
      # test_sentence = spam[1]

      print(test_sentence)

      removed_stops = remove_stopwords(test_sentence)
      print(removed_stops)

      p = PorterStemmer()
      stemmed = p.stem(removed_stops)
      print(stemmed)

      # tokens = tokenize(stemmed)
      # print(list(tokens))

I might come by tonight then if my class lets out early
I come tonight class lets early
i come tonight class lets earli
```

  - For example you can see early and earli make the same meaning so with the help of PorterStemmer we remove those duplicates and make it less complicated for future analysis
- Tokenization : Splitting something large into smaller pieces based on some rules
  - We do that with the help of gensin library and from there we use tokenize

Create a dictionary of words

```
[ ]: def tokenize_sentence(sentence):
         p = PorterStemmer()
         removed_stops = remove_stopwords(sentence)
         stemmed = p.stem(removed_stops)
         tokens = tokenize(stemmed)
         return list(tokens)
```

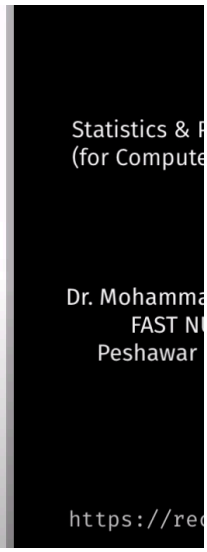
- A funciton for whatever we did above

```
]: dictionary = set()     # will have unique values only
   spams_tokenized = []
   nons_tokenized = []


   for sentence in spam:
       sentence_tokens = tokenize_sentence(sentence)
       spams_tokenized.append(sentence_tokens)
       dictionary  = dictionary.union(sentence_tokens)   # add sentence words to the dic


   for sentence in non:
       sentence_tokens = tokenize_sentence(sentence)
       nons_tokenized.append(sentence_tokens)
       dictionary  = dictionary.union(sentence_tokens)


   print("Tokenized spam: ", spams_tokenized)
   print("Tokenized non:  ", nons_tokenized)
   print("Dictionary:     ", dictionary)
```



- It is pretty obvious you can look at the print function and literally understand what is going on
- Dictionary have all the unique words just

Basic Stats

```
[14]: # These things do not depend on an individual word so let's calculate them separately

      total_word_count = len(dictionary)
      total_spam_messages = len(spams_tokenized)
      total_all_messages = len(spams_tokenized) + len(nons_tokenized)

      print("Total Number of words: ", total_word_count)

      Total Number of words:  101
```

- This after skimming of early and earlier

```
# P(spam) ... does not depend on an individual word so let's calculate that separatel

p_spam = total_spam_messages / total_all_messages

print("P(spam) = ", p_spam)
```

```
P(spam) =  0.38461538461538464
```

- Now our pre-processing is complete

```
[ ]: # Helper function to count occurances

def count_word_in_messages(word, messages):
    total_count = 0
    for msg in messages:
        if word in msg:       # notice this ensured uniqueness automatically
            total_count += 1

    return total_count
```

- This gives us the count of
  - How many times a word has appeared in the messages list provided to it as a parameter

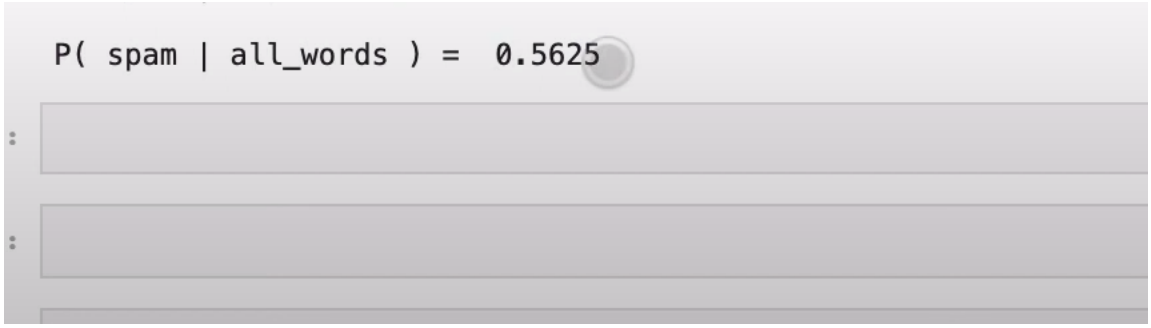
```
for test_sentence in spam_test:
    test_sentence = tokenize_sentence(test_sentence)
    print(test_sentence)

    # let's run this for each word separately
    for word in test_sentence:
        print("----------------")
        print("Runnig for word:", word)

        # Find P( w | spam)
        spam_count = count_word_in_messages(word, spams_tokenized)
        p_w_spam = spam_count / total_spam_messages
        print("P( w | spam)  = ", p_w_spam)

        # Find P( w )
        w_count = count_word_in_messages(word, spams_tokenized)
        w_count += count_word_in_messages(word, nons_tokenized)
        p_w = w_count / total_all_messages
        print("P( w )        = ", p_w)
```

```
    # Find P( spam | w )
    p_spam_w = (p_w_spam * p_spam) / p_w
    print("P( spam )     = ", p_spam)
    print("P( spam | w ) = ", p_spam_w)
    print("")
    final_prob *= p_spam_w
```

- The above is all basics probability
- Last one is BAYESIAN RULE
- It is important to stay close to mathematics while giving variables name
- 

- Now after computing the total probability of a word being spam is **0.5625**
    - This is the point where we put a threshold of whether anything greater than this should be spam or less than should not be spam
    - But we take this threshold based on experience or based on true facts or based on what percentage would seem right
    - 0.5 is closer to 50/50 we can do 0.8 which is 80%
    - We going for 0.5 here

- Revision:
    - Take a data set
    - Preprocess it
    - After preprocessing feed it to the model
    - Model gave us prediction
    - After that it gives us a quantified result as in a number
    - Then we with experience or however put a threshold on it
- Things to note :
    - This without probability would have been a mess
    - Keep clean code merged with mathematics
    - Without mathematics this code is really complex
    -