109-2

Natural Language Processing(Lab 1)

(紅色框線為 lab 實作)

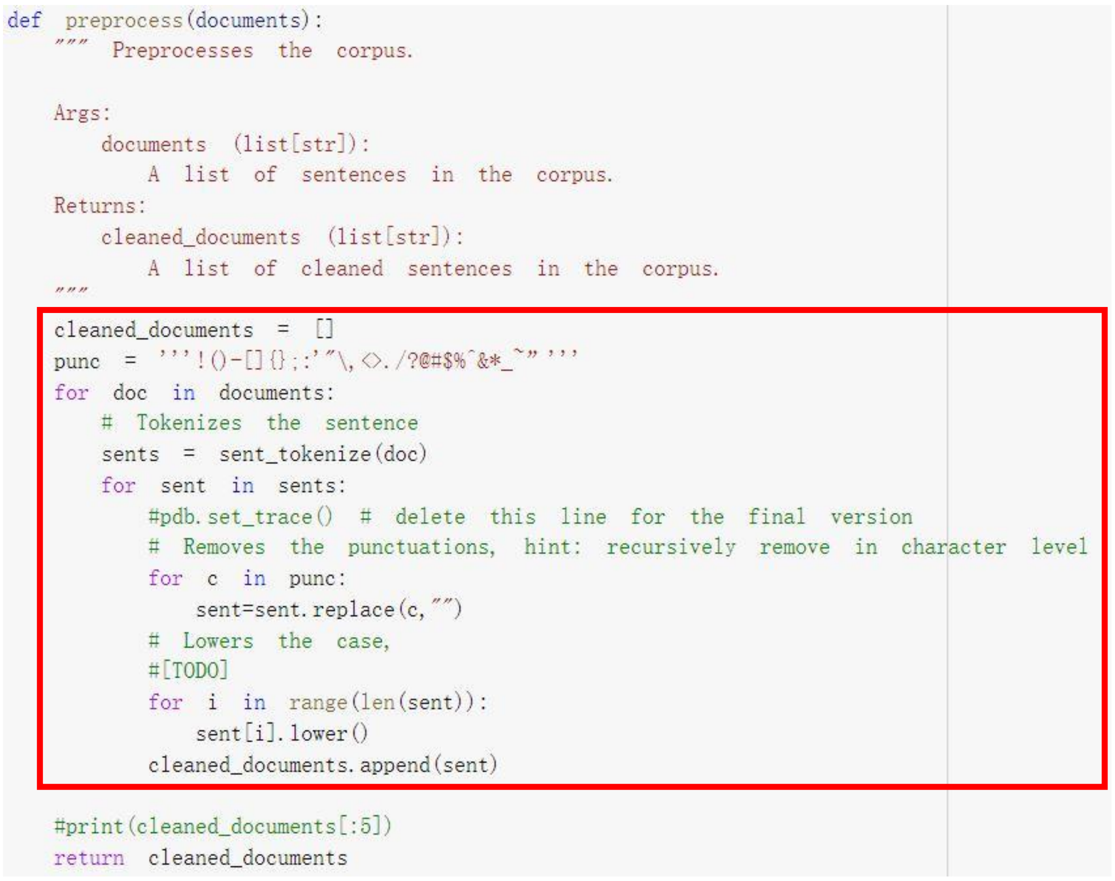
1. Dataset : Tweets
2. 進行 data preprocessing

   Use NLTK to tokenize data(sent_tokenize)

   (先用 nltk 的 sent_tokenize 幫助進行斷句)

   Remove punctuations and lower the cases

   (將標點符號去除,並將所有的詞都轉成小寫)

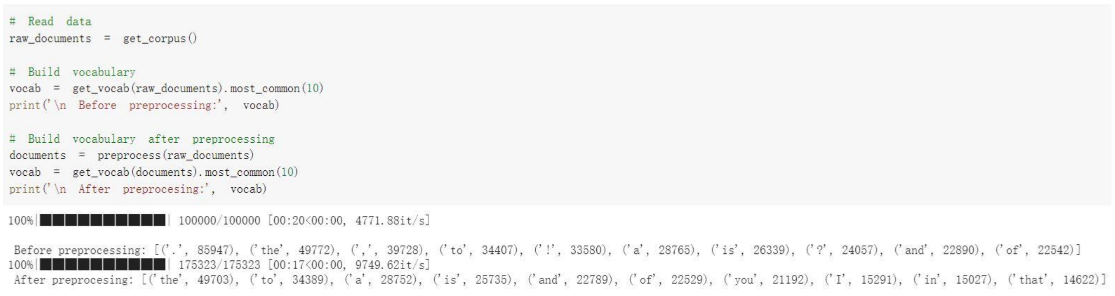


```
def preprocess(documents):
    """ Preprocesses the corpus.

    Args:
        documents (list[str]):
            A list of sentences in the corpus.
    Returns:
        cleaned_documents (list[str]):
            A list of cleaned sentences in the corpus.
    """
    cleaned_documents = []
    punc = '''!()-[]{};:'"\,<>./?@#$%^&*_~”'''
    for doc in documents:
        # Tokenizes the sentence
        sents = sent_tokenize(doc)
        for sent in sents:
            #pdb.set_trace() # delete this line for the final version
            # Removes the punctuations, hint: recursively remove in character level
            for c in punc:
                sent=sent.replace(c,"")
            # Lowers the case,
            #[TODO]
            for i in range(len(sent)):
                sent[i].lower()
            cleaned_documents.append(sent)

    #print(cleaned_documents[:5])
    return cleaned_documents
```

將進行 preprocessing 前後的 top-10 common word 列出來



```
# Read data
raw_documents = get_corpus()

# Build vocabulary
vocab = get_vocab(raw_documents).most_common(10)
print('\n Before preprocessing:', vocab)

# Build vocabulary after preprocessing
documents = preprocess(raw_documents)
vocab = get_vocab(documents).most_common(10)
print('\n After preprocesing:', vocab)
```

```
100%|█████████████| 100000/100000 [00:20<00:00, 4771.88it/s]

 Before preprocessing: [('.', 85947), ('the', 49772), (',', 39728), ('to', 34407), ('!', 33580), ('a', 28765), ('is', 26339), ('?', 24057), ('and', 22890), ('of', 22542)]
100%|█████████████| 175323/175323 [00:17<00:00, 9749.62it/s]
 After preprocesing: [('the', 49703), ('to', 34389), ('a', 28752), ('is', 25735), ('and', 22789), ('of', 22529), ('you', 21192), ('I', 15291), ('in', 15027), ('that', 14622)]
```

3.建立 N-gram Model

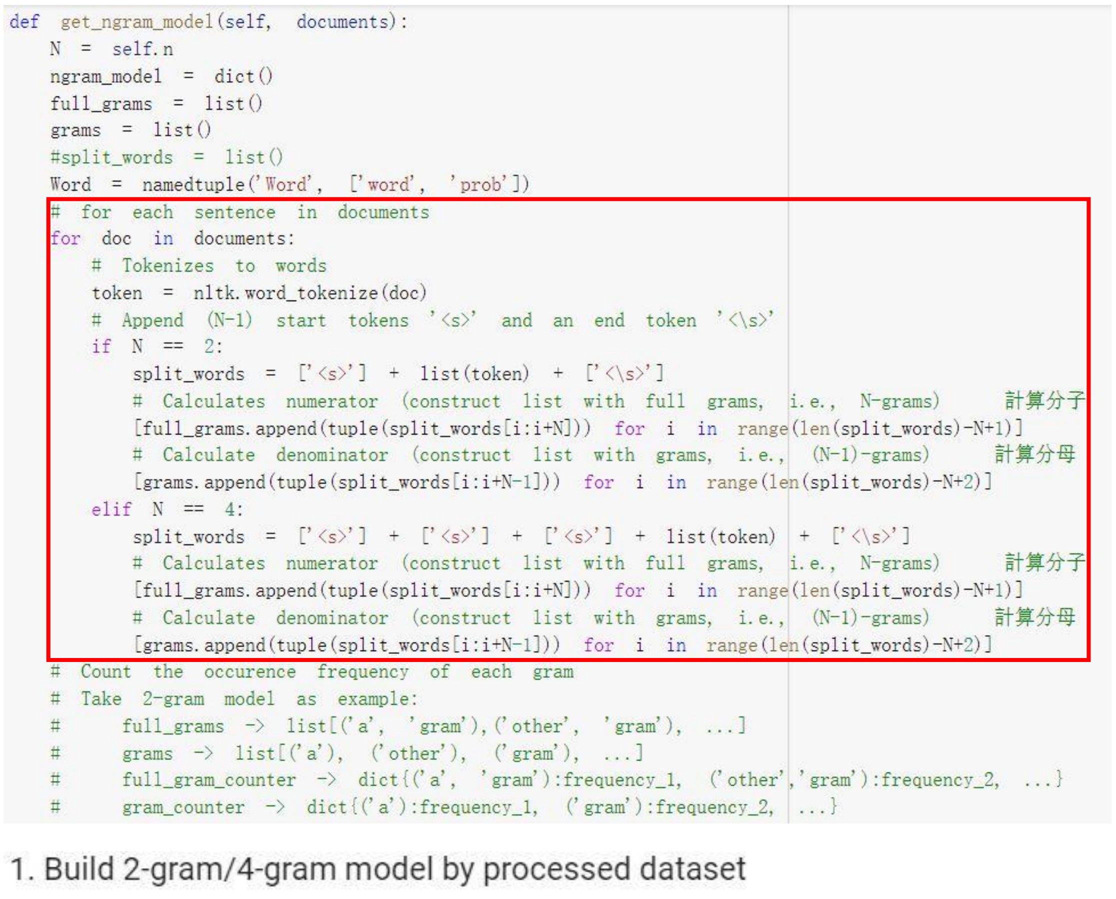
Use NLTK to tokenize data(word_tokenize)

(先用 nltk 的 word_tokenize 幫助進行斷詞)

分別實現 bigram 和 4 gram 的 language model 去產生 sequence

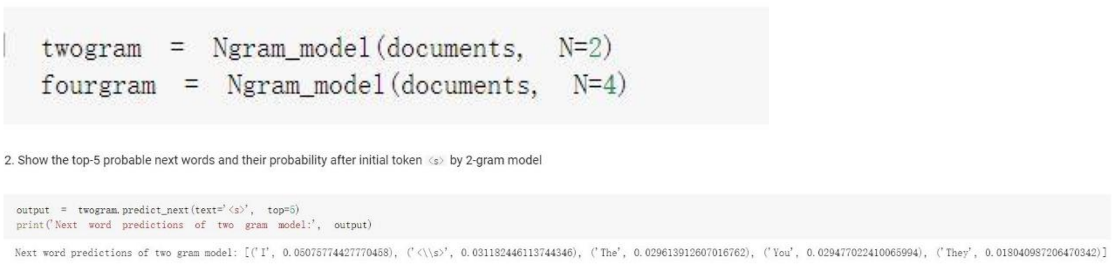
Bigram 型式:<s>+…　……　…+</s>

4 gram 型式:<s><s><s>+… …　…+</s>



```
def  get_ngram_model(self,  documents):
    N  =  self.n
    ngram_model  =  dict()
    full_grams  =  list()
    grams  =  list()
    #split_words  =  list()
    Word  =  namedtuple('Word',  ['word',  'prob'])
    #  for  each  sentence  in  documents
    for  doc  in  documents:
        #  Tokenizes  to  words
        token  =  nltk.word_tokenize(doc)
        #  Append  (N-1)  start  tokens  '<s>'  and  an  end  token  '<\s>'
        if  N  ==  2:
            split_words  =  ['<s>']  +  list(token)  +  ['<\s>']
            #  Calculates  numerator  (construct  list  with  full  grams,  i.e.,  N-grams)     計算分子
            [full_grams.append(tuple(split_words[i:i+N]))  for  i  in  range(len(split_words)-N+1)]
            #  Calculate  denominator  (construct  list  with  grams,  i.e.,  (N-1)-grams)     計算分母
            [grams.append(tuple(split_words[i:i+N-1]))  for  i  in  range(len(split_words)-N+2)]
        elif  N  ==  4:
            split_words  =  ['<s>']  +  ['<s>']  +  ['<s>']  +  list(token)  +  ['<\s>']
            #  Calculates  numerator  (construct  list  with  full  grams,  i.e.,  N-grams)     計算分子
            [full_grams.append(tuple(split_words[i:i+N]))  for  i  in  range(len(split_words)-N+1)]
            #  Calculate  denominator  (construct  list  with  grams,  i.e.,  (N-1)-grams)     計算分母
            [grams.append(tuple(split_words[i:i+N-1]))  for  i  in  range(len(split_words)-N+2)]
    #  Count  the  occurence  frequency  of  each  gram
    #  Take  2-gram  model  as  example:
    #      full_grams  ->  list[('a',  'gram'),('other',  'gram'),  ...]
    #      grams  ->  list[('a'),  ('other'),  ('gram'),  ...]
    #      full_gram_counter  ->  dict{('a',  'gram'):frequency_1,  ('other','gram'):frequency_2,  ...}
    #      gram_counter  ->  dict{('a'):frequency_1,  ('gram'):frequency_2,  ...}
```

1. Build 2-gram/4-gram model by processed dataset



```
twogram  =  Ngram_model(documents,  N=2)
fourgram  =  Ngram_model(documents,  N=4)
```

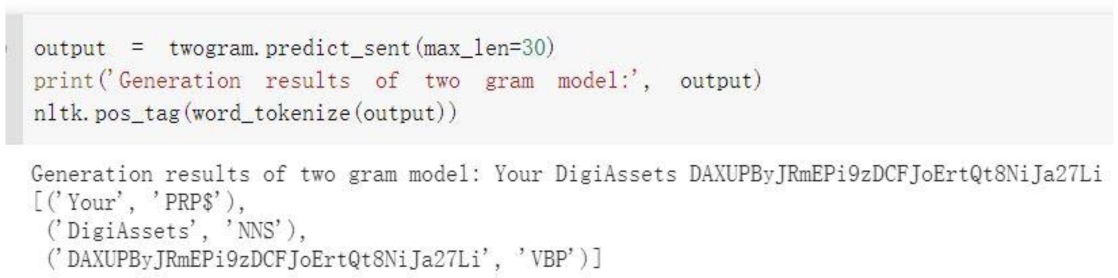
2. Show the top-5 probable next words and their probability after initial token <s> by 2-gram model

```
output  =  twogram.predict_next(text='<s>',  top=5)
print('Next  word  predictions  of  two  gram  model:',  output)
```

```
Next word predictions of two gram model: [('I', 0.05075774427770458), ('<\\s>', 0.031182446113744346), ('The', 0.029613912607016762), ('You', 0.029477022410065994), ('They', 0.018040987206470342)]
```

4.POS Tagging

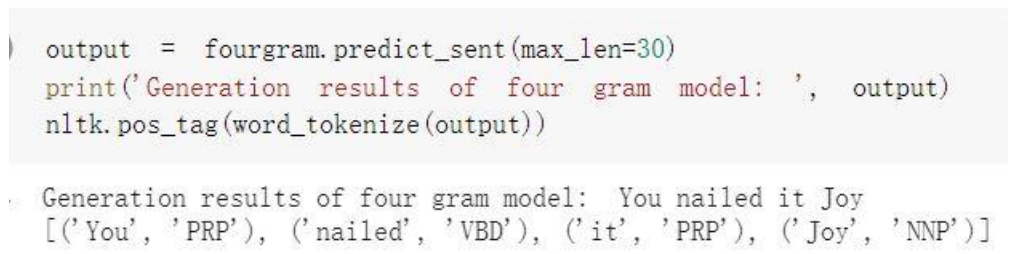
3. Generate a sentence with 2-gram model and find the POS taggings



```
output = twogram.predict_sent(max_len=30)
print('Generation results of two gram model:', output)
nltk.pos_tag(word_tokenize(output))
```

```
Generation results of two gram model: Your DigiAssets DAXUPByJRmEPi9zDCFJoErtQt8NiJa27Li
[('Your', 'PRP$'),
 ('DigiAssets', 'NNS'),
 ('DAXUPByJRmEPi9zDCFJoErtQt8NiJa27Li', 'VBP')]
```

4. Generate a sentence with 4-gram model and find the POS taggings



```
output = fourgram.predict_sent(max_len=30)
print('Generation results of four gram model: ', output)
nltk.pos_tag(word_tokenize(output))
```

```
Generation results of four gram model:  You nailed it Joy
[('You', 'PRP'), ('nailed', 'VBD'), ('it', 'PRP'), ('Joy', 'NNP')]
```