Ques no 1

In [5]:
```
import nltk
from nltk.corpus import brown
from nltk import FreqDist

```

50 most frequently occurring words of a text that are not stopwords

In [12]:
```
stopwords = nltk.corpus.stopwords.words('english')
def most_frequent_content_words(text):
    content_words = [w.lower() for w in text if w.lower() not in stopwords a
    fd = nltk.FreqDist(content_words)
    return [w for w, num in fd.most_common(50)]
text=brown.words()
print (most_frequent_content_words(text))
```

```
['one', 'would', 'said', 'new', 'could', 'time', 'two', 'may', 'first', 'like',
'man', 'even', 'made', 'also', 'many', 'must', 'af', 'back', 'years', 'much',
'way', 'well', 'people', 'mr.', 'little', 'state', 'good', 'make', 'world', 'st
ill', 'see', 'men', 'work', 'long', 'get', 'life', 'never', 'day', 'another',
'know', 'last', 'us', 'might', 'great', 'old', 'year', 'come', 'since', 'go',
'came']
```

Ques no 2

Changing using slice and concatenation

In [14]:
```
s = 'colorless'
s = s[:4] + 'u' + s[4:]
s
```

Out[14]:
```
'colourless'
```

Ques no 3

Tokenizing

In [49]:
```
import re
sent="She sells sea shells by the sea shore"
words=nltk.word_tokenize(sent)
words

```

Out[49]: ['She', 'sells', 'sea', 'shells', 'by', 'the', 'sea', 'shore']

Words beginning with 'sh'

In [50]:
```
print([w for w in words if w.startswith('sh')])


```

['shells', 'shore']

Words with length greater than 4

In [51]:
```
print([w for w in words if len(w) > 4])
```

['sells', 'shells', 'shore']

Ques no 4

Parts of Speech involved

In [22]:
```
sent = 'They wind back the clock, while we chase after the wind.'
nltk.pos_tag(nltk.word_tokenize(sent))
```

Out[22]:
```
[('They', 'PRP'),
 ('wind', 'VBP'),
 ('back', 'RB'),
 ('the', 'DT'),
 ('clock', 'NN'),
 (',', ','),
 ('while', 'IN'),
 ('we', 'PRP'),
 ('chase', 'VBP'),
 ('after', 'IN'),
 ('the', 'DT'),
 ('wind', 'NN'),
 ('.', '.')]
```

Pronounciations involved

In [46]:
```
words=nltk.word_tokenize(sent)
alphabetical_words=[word.lower() for word in words if word.isalpha()]

pronounciations = nltk.corpus.cmudict.dict()
for word in alphabetical_words:
   print(word,pronounciations[word])
```

```
they [['DH', 'EY1']]
wind [['W', 'AY1', 'N', 'D'], ['W', 'IH1', 'N', 'D']]
back [['B', 'AE1', 'K']]
the [['DH', 'AH0'], ['DH', 'AH1'], ['DH', 'IY0']]
clock [['K', 'L', 'AA1', 'K']]
while [['W', 'AY1', 'L'], ['HH', 'W', 'AY1', 'L']]
we [['W', 'IY1']]
chase [['CH', 'EY1', 'S']]
after [['AE1', 'F', 'T', 'ER0']]
the [['DH', 'AH0'], ['DH', 'AH1'], ['DH', 'IY0']]
wind [['W', 'AY1', 'N', 'D'], ['W', 'IH1', 'N', 'D']]
```

Ques no 5

In [24]:
```
raw = """DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives
... a mandate from the masses, not from some farcical aquatic ceremony."""
tokens = nltk.word_tokenize(raw)
```

Porter stemmer

In [25]:
```
>>> porter = nltk.PorterStemmer()
>>> [porter.stem(t) for t in tokens]
```

Out[25]:
```
['denni',
 ':',
 'listen',
 ',',
 'strang',
 'women',
 'lie',
 'in',
 'pond',
 'distribut',
 'sword',
 'is',
 'no',
 'basi',
 'for',
 'a',
 'system',
 'of',
 'govern',
 '.',
 'suprem',
 'execut',
 'power',
 'deriv',
 'from',
 'a',
 'mandat',
 'from',
 'the',
 'mass',
 ',',
 'not',
 'from',
 'some',
 'farcic',
 'aquat',
 'ceremoni',
 '.']
```

Lancaster stemmer

In [26]:
```
lancaster = nltk.LancasterStemmer()
[lancaster.stem(t) for t in tokens]
```

Out[26]:
```
['den',
 ':',
 'list',
 ',',
 'strange',
 'wom',
 'lying',
 'in',
 'pond',
 'distribut',
 'sword',
 'is',
 'no',
 'bas',
 'for',
 'a',
 'system',
 'of',
 'govern',
 '.',
 'suprem',
 'execut',
 'pow',
 'der',
 'from',
 'a',
 'mand',
 'from',
 'the',
 'mass',
 ',',
 'not',
 'from',
 'som',
 'farc',
 'aqu',
 'ceremony',
 '.']
```

Ques 6

In [33]:
```
brown_tagged_sents = brown.tagged_sents(categories='news')
brown_sents = brown.sents(categories='news')
```

In [34]:
```
size = int(len(brown_tagged_sents) * 0.9)
size
```

Out[34]:
```
4160
```

Splitting train and test data

In [35]:
```
train_sents = brown_tagged_sents[:size]
test_sents = brown_tagged_sents[size:]
```

```
### Tagging a train data
```

In [40]:
```
bigram_tagger = nltk.BigramTagger(train_sents)
bigram_tagger.tag(brown_sents[2000])
```

Out[40]:
```
[('While', 'CS'),
 ('availability', 'NN'),
 ('of', 'IN'),
 ('mortgage', 'NN'),
 ('money', 'NN'),
 ('has', 'HVZ'),
 ('been', 'BEN'),
 ('a', 'AT'),
 ('factor', 'NN'),
 ('in', 'IN'),
 ('encouraging', 'VBG'),
 ('apartment', 'NN'),
 ('construction', 'NN'),
 (',', ','),
 ('the', 'AT'),
 ('generally', 'RB'),
 ('high', 'JJ'),
 ('level', 'NN'),
 ('of', 'IN'),
 ('prosperity', 'NN'),
 ('in', 'IN'),
 ('the', 'AT'),
 ('past', 'NN'),
 ('few', None),
 ('years', None),
 ('plus', None),
 ('rising', None),
 ('consumer', None),
 ('income', None),
 ('are', None),
 ('among', None),
 ('the', None),
 ('factors', None),
 ('that', None),
 ('have', None),
 ('encouraged', None),
 ('builders', None),
 ('to', None),
 ('concentrate', None),
 ('in', None),
 ('the', None),
 ('apartment-building', None),
 ('field', None),
 ('.', None)]
```

Tagging a test data

In [41]:
```
unseen_sent = brown_sents[4200]
bigram_tagger.tag(unseen_sent)
```

Out[41]:
```
[('and', 'CC'),
 ('it', 'PPS'),
 ('was', 'BEDZ'),
 ('filled', None),
 ('then', None),
 ('as', None),
 ('now', None),
 ('by', None),
 ('quarreling', None),
 ('tribes', None),
 ('with', None),
 ('no', None),
 ('political', None),
 ('or', None),
 ('historical', None),
 ('unity', None),
 ('.', None)]
```

Accuracy of test data

In [42]:
```
bigram_tagger.evaluate(test_sents)
```

Out[42]:
```
0.10206319146815508
```

Accuracy of train data

In [43]:
```
bigram_tagger.evaluate(train_sents)
```

Out[43]:
```
0.7884137382485832
```

Why training accuracy is better than test accuracy?

The tagging accuracy is higher with training data than the test data. The bigram tagger manages to tag every word in a sentence it saw during training, but does badly on an unseen sentence since when it encounters a new word , it is unable to assign a tag to the unseen word. So, the tag 'None' is assigned as seen in the above example in unseen_sent.

As a result, it cannot tag the following word even if it was seen during training, because it never saw it during training with the tag i.e. 'None' on the previous word. Consequently, the tagger fails to tag the rest of

the sentence as seen in the example above. Hence, its overall accuracy score is very low for unseen data.

As n gets larger, the specificity of the contexts increases, as does the chance that the data we wish to tag contains contexts that were not present in the training data. This is known as the sparse data problem in NLP.

In [ ]:

```

```