```
import nltk.
  from nltk.tokenize import .sent_tokenize,word_tokenize.
  nltk.download('punkt')
  nltk.download('averaged_perceptron_tagger')
```

Exercise:-1

```
import nltk
text = word_tokenize("And now for something
                      completely different")
nltk.pos_tag(text)
```

Exercise-2

```
from nltk.corpus .import brown
  tagsen = brown.tagged .sents()
```

step:-1 Prepare data sets.

```
len (tagsen)
br_train = tagsen [0:50000]
br_test = tagsen [50000:]
 br_test [0]
```

step:-2

```
t0 = nltk.DefaultTagger ('NN')
t1 = .nltk. UnigramTagger.(br_train,backoff=t0)
t2 = nltk. BigramTagger(br_train,
                        backoff = t1)
```

Step:-3

```
t2.evaluate(br_test)
```

Natural Language Processing Lab
Lab9. Building Bigram Tagger

In this lab, you will build a bigram tagger and test it out. We will use the Brown Corpus and its native tagset.

EXERCISE-1

```
import nltk
text = word_tokenize("And now for something completely different")
nltk.pos_tag(text)
```

Output:

```
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'),
('completely', 'RB'), ('different', 'JJ')]
```

Question: Write down the expansion for CC, RB, ..., JJ in the above output.

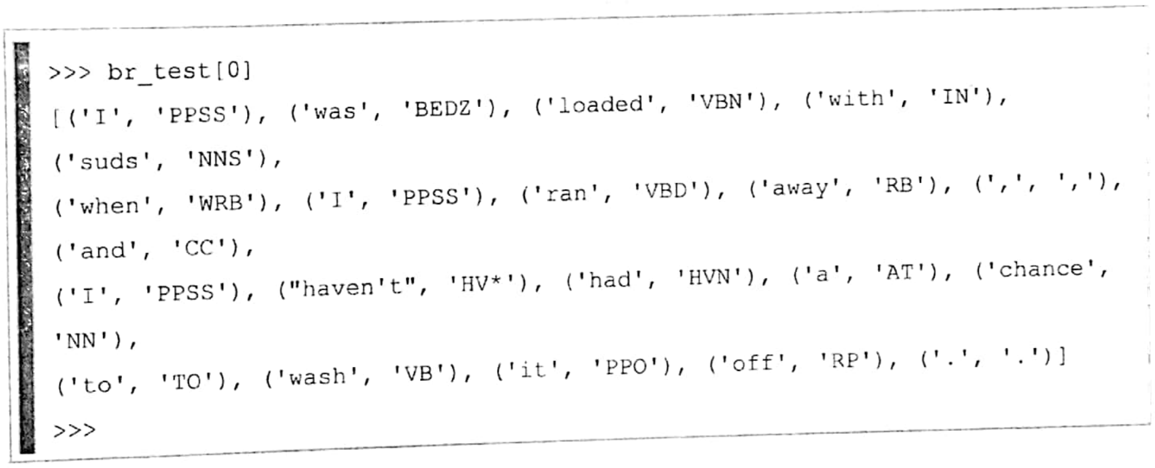
EXERCISE-2

Type the following lines and load Brown corpus into the list tagsen.

```
from nltk.corpus import brown
tagsen = brown.tagged_sents()
```

STEP 1: Prepare data sets

There are a total of 57,340 POS-tagged sentences in the Brown Corpus. Among them, assign the first 50,000 to your list of training sentences. Then, assign the remaining sentences to your list of testing sentences. The first of your testing sentences should look like this:



```
>>> br_test[0]
[('I', 'PPSS'), ('was', 'BEDZ'), ('loaded', 'VBN'), ('with', 'IN'),
('suds', 'NNS'),
('when', 'WRB'), ('I', 'PPSS'), ('ran', 'VBD'), ('away', 'RB'), (',', ','),
('and', 'CC'),
('I', 'PPSS'), ("haven't", 'HV*'), ('had', 'HVN'), ('a', 'AT'), ('chance',
'NN'),
('to', 'TO'), ('wash', 'VB'), ('it', 'PPO'), ('off', 'RP'), ('.', '.')]
>>>
```

STEP 2: Build a bigram tagger

Following the steps shown in this chapter, build a bigram tagger with two back-off models. The first one on the stack should be a default tagger that assigns `'NN'` by default.

STEP 3: Evaluate

Evaluate your bigram tagger on the test sentences. You should be getting the accuracy score of **0.911.** If not, something went wrong: go back and re-build your tagger.

Step: 3

t2 . evaluate (br_test)

Step: 4

(1) total_train = [len (1) for 1 in br_train]
sum (total_train)

total_test = [len(1) . for 1 in br_test]
sum (total_test)

2) t1 . evaluate (br_test)
t2 evaluate (br_test)

3) br_train [0]
4) br_train [1277]
5) br_train [1277] [1]

4) br_train_flat = [(word, tag) . for sent in br_train
for (word, tag) in sent]

br_train_flat [:40]

br_train_flat [13]

5) (a) fd = nltk. FreqDist (br_train_flat)
cfd = nltk . ConditionalFreqDist (br_train_flat)

cfd['cold'] . most_common()

(b) br_train_2gram = list (nltk. ngrams (br_train_flat, 2))

br_train_cold = [a[1] . for (a, b) in br_train_2gram
if b[0] == 'cold']
fdist = nltk. FreqDist (br_train_cold)
[tag . for (tag, _) . in fdist . most_common()]

Scanned with CamScanner

STEP 4: Explore

Now, explore your tagger to answer the questions below.

1. How big are your training data and testing data? Answer in terms of the number of total words in them.
2. What is the performance of each of the two back-off taggers? How much improvement did you get: (1) going from the default tagger to the unigram tagger, and (2) going from the unigram tagger to the bigram tagger?
3. Recall that 'cold' is ambiguous between `JJ` 'adjective' and `NN` 'singular noun'. Let's explore the word in the training data. The problem with the training data, through, is that it is a list of tagged *sentences*, and it's difficult to get to the tagged words which are one level below:

```
>>> br_train[0]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-
TL'),
('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'),
('investigation',
'NN'), ('of', 'IN'), ("Atlanta's", 'NP$'), ('recent', 'JJ'), ('primary',
'NN'),
('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'AT'),
('evidence',
'NN'), ("''", "''"), ('that', 'CS'), ('any', 'DTI'), ('irregularities',
'NNS'),
('took', 'VBD'), ('place', 'NN'), ('.', '.')]
>>> br_train[1277]          # 1278th sentence
[('``', '``'), ('I', 'PPSS'), ('told', 'VBD'), ('him', 'PPO'), ('who',
'WPS'),
('I', 'PPSS'), ('was', 'BEDZ'), ('and', 'CC'), ('he', 'PPS'), ('was',
'BEDZ'),
('quite', 'QL'), ('cold', 'JJ'), ('.', '.')]
>>> br_train[1277][11]      # 1278th sentence, 12th word
('cold', 'JJ')
>>>
```

4. To be able to compile tagged-word-level statistics, we will need a flat list of tagged words, without them being organized into sentences. How to do this? You can use multi-loop list comprehension to construct it:

```
>>> br_train_flat = [(word, tag) for sent in br_train for (word, tag) in
sent]
                # [x for innerlist in outerlist for x in innerlist]
>>> br_train_flat[:40]
[('The', 'AT'), ('Fulton', 'NP-TL'), ('County', 'NN-TL'), ('Grand', 'JJ-
TL'),
```

c) br_pre = [(W2+"/"+t2, t1) for ((W1, t1), (W2, t2)) in br_train_2grams]

br_pre_cfd = nltk.ConditionalFreqDist(br_pre)

br_pre

d) br_pre_cfd['cold/NN'].most_common()

br_pre_cfd['cold/JJ'].most_common()

6) bigram_tagger = nltk.BigramTagger(br_train)

(a) text1 = word_tokenize("I was very cold.")

bigram_tagger.tag(text1)

(b) text2 = word_tokenize("I had cold.")

bigram_tagger.tag(text2)

(c) text3 = word_tokenize("I had a severe cold.")

bigram_tagger.tag(text3)

(d) text4 = word_tokenize("January was a cold month")

bigram_tagger.tag(text4)

8) (a) text5 = word_tokenize("I failed to do so")

bigram_tagger.tag(text5)

(b) text6 = word_tokenize("I was happy, but so was my enemy")

bigram_tagger.tag(text6)

(c) text7 = word_tokenize("So, how was the exam?")

bigram_tagger.tag(text7)

(d) text8 = word_tokenize("The students came in early so they can get good seats")

bigram_tagger.tag(text8)

Scanned with CamScanner

```
('Jury', 'NN-TL'), ('said', 'VBD'), ('Friday', 'NR'), ('an', 'AT'),
('investigation',
'NN'), ('of', 'IN'), ("Atlanta's", 'NP$'), ('recent', 'JJ'), ('primary',
'NN'),
('election', 'NN'), ('produced', 'VBD'), ('``', '``'), ('no', 'AT'),
('evidence',
'NN'), ("''", "''"), ('that', 'CS'), ('any', 'DTI'), ('irregularities',
'NNS'),
('took', 'VBD'), ('place', 'NN'), ('.', '.'), ('The', 'AT'), ('jury',
'NN'),
('further', 'RBR'), ('said', 'VBD'), ('in', 'IN'), ('term-end', 'NN'),
('presentments',
'NNS'), ('that', 'CS'), ('the', 'AT'), ('City', 'NN-TL'), ('Executive',
'JJ-TL'),
('Committee', 'NN-TL'), (',', ','), ('which', 'WDT'), ('had', 'HVD')]
>>> br_train_flat[13]       # 14th word
('election', 'NN')
>>>
```

5. Now, exploring this list of `(word, POS)` pairs from the training data, answer the questions below.
   a. Which is the more likely tag for 'cold' overall?
   b. When the POS tag of the preceding word (call it POS$_{n-1}$) is `AT`, what is the likelihood of 'cold' being a noun? How about it being an adjective?
   c. When POS$_{n-1}$ is `JJ`, what is the likelihood of 'cold' being a noun? How about it being an adjective?
   d. Can you find any POS$_{n-1}$ that favors `NN` over `JJ` for the following word 'cold'?
6. Based on what you found, how is your bigram tagger expected to tag 'cold' in the following sentences?
   a. *I was very cold.*
   b. *I had a cold.*
   c. *I had a severe cold.*
   d. *January was a cold month.*
7. Verify your prediction by having the tagger actually tag the four sentences. What did you find?
8. Have the tagger tag the following sentences, all of which contain the word 'so':
   a. *I failed to do so.*
   b. *I was happy, but so was my enemy.*
   c. *So, how was the exam?*
   d. *The students came in early so they can get good seats.*
   e. *She failed the exam, so she must take it again.*
   f. *That was so incredible.*

e)

```
text9= word_tokenize("She failed the exam, so she
                      must take it again")

bigram_tagger.tag(text9)
```

f.

```
text10 = word_tokenize("That was so incredible")
bigram_tagger.tag(text 10)
```

g)

```
text 11 = word_tokenize("Wow, so incredible")
bigram_tagger.tag(text 11)
```