205229133 ¶

Natural Language Processing Lab

Lab9. Building Bigram Tagger

In [1]:

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

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\ELCOT\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\ELCOT\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[1]:

```
True
```

EXERCISE-1

In [2]:

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

Out[2]:

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

EXERCISE-2

In [3]:

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

STEP 1: Prepare data sets

In [4]:

```
len(tagsen)
```

Out[4]:

```
57340
```

In [5]:

```
br_train=tagsen[0:50000]
br_test=tagsen[50000:]
br_test[0]
```

Out[5]:

```
[('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

In [6]:

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

STEP 3: Evaluate

In [7]:

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

Out[7]:

```
0.9111006662708622
```

STEP 4: Explore

1. How big are your training data and testing data? Answer in terms of the number of total words in them.

In [8]:

```
total_train=[len(l) for l in br_train]
sum(total_train)
```

Out[8]:

```
1039920
```

In [9]:

```
total_test=[len(l) for l in br_test]
sum(total_test)
```

Out[9]:

```
121272
```

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?

In [10]:

```
t1.evaluate(br_test)
```

Out[10]:

```
0.8897849462365591
```

In [11]:

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

Out[11]:

```
0.9111006662708622
```

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

In [12]:

```
br_train[0]
```

Out[12]:

```
[('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'),
 ('.', '.')]
```

In [13]:

```
br_train[1277]
```

Out[13]:

```
[('``', '``'),
 ('I', 'PPSS'),
 ('told', 'VBD'),
 ('him', 'PPO'),
 ('who', 'WPS'),
 ('I', 'PPSS'),
 ('was', 'BEDZ'),
 ('and', 'CC'),
 ('he', 'PPS'),
 ('was', 'BEDZ'),
 ('quite', 'QL'),
 ('cold', 'JJ'),
 ('.', '.')]
```

In [14]:

```
br_train[1277][11]
```

Out[14]:

```
('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:

In [15]:

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

In [16]:

```
br_train_flat[:40]
```

Out[16]:

```
[('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'),
 ('.', '.'),
 ('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')]
```

In [17]:

```
br_train_flat[13]
```

Out[17]:

```
('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?

In [18]:

```
fd = nltk.FreqDist(br_train_flat)
cfd = nltk.ConditionalFreqDist(br_train_flat)
```

In [19]:

```
cfd['cold'].most_common()
```

Out[19]:

```
[('JJ', 110), ('NN', 8), ('RB', 2)]
```

b. When the POS tag of the preceding word (call it POSn-1) is AT, what is the likelihood of 'cold' being a noun? How about it being an adjective?

In [20]:

```
br_train_2grams = list(nltk.ngrams(br_train_flat,2))
br_train_cold=[a[1] for (a, b) in br_train_2grams if b[0] == 'cold']
fdist = nltk.FreqDist(br_train_cold)
[tag for (tag, _) in fdist.most_common()]
```

Out[20]:

```
['AT',
 'IN',
 'CC',
 'QL',
 'BEDZ',
 'JJ',
 ',',
 'DT',
 'PP$',
 'RP',
 '``',
 'NN',
 'VBN',
 'VBD',
 'CS',
 'BEZ',
 'DOZ',
 'RB',
 'PPSS',
 'BE',
 'VB',
 'VBZ',
 'NP$',
 'BEDZ*',
 '--',
 'DTI',
 'WRB',
 'BED']
```

c. When POSn-1 is JJ, what is the likelihood of 'cold' being a noun? How about it being an adjective?

In [21]:

```
br_pre = [(w2+"/"+t2, t1) for ((w1,t1),(w2,t2)) in br_train_2grams]
br_pre_cfd = nltk.ConditionalFreqDist(br_pre)
br_pre
```

Out[21]:

```
[('Fulton/NP-TL', 'AT'),
 ('County/NN-TL', 'NP-TL'),
 ('Grand/JJ-TL', 'NN-TL'),
 ('Jury/NN-TL', 'JJ-TL'),
 ('said/VBD', 'NN-TL'),
 ('Friday/NR', 'VBD'),
 ('an/AT', 'NR'),
 ('investigation/NN', 'AT'),
 ('of/IN', 'NN'),
 ("Atlanta's/NP$", 'IN'),
 ('recent/JJ', 'NP$'),
 ('primary/NN', 'JJ'),
 ('election/NN', 'NN'),
 ('produced/VBD', 'NN'),
 ('``/``', 'VBD'),
 ('no/AT', '``'),
 ('evidence/NN', 'AT'),
 ("''/''", 'NN'),
```

d. Can you find any POSn-1 that favors NN over JJ for the following word 'cold'?

In [22]:

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

Out[22]:

```
[('AT', 4), ('JJ', 2), (',', 1), ('DT', 1)]
```

In [23]:

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

Out[23]:

```
[('AT', 38),
 ('IN', 14),
 ('CC', 8),
 ('QL', 7),
 ('BEDZ', 7),
 ('JJ', 4),
 ('DT', 3),
 (',', 3),
 ('PP$', 3),
 ('``', 2),
 ('NN', 2),
 ('VBN', 2),
 ('VBD', 2),
 ('CS', 1),
 ('BEZ', 1),
 ('DOZ', 1),
 ('RB', 1),
 ('PPSS', 1),
 ('BE', 1),
 ('VB', 1),
 ('VBZ', 1),
 ('NP$', 1),
 ('BEDZ*', 1),
 ('--', 1),
 ('RP', 1),
 ('DTI', 1),
 ('WRB', 1),
 ('BED', 1)]
```

6. Based on what you found, how is your bigram tagger expected to tag 'cold' in the following sentences?

In [24]:

```
bigram_tagger = nltk.BigramTagger(br_train)
```

a. I was very cold.

In [25]:

```
text1 = word_tokenize("I was very cold.")
bigram_tagger.tag(text1)
```

Out[25]:

```
[('I', 'PPSS'), ('was', 'BEDZ'), ('very', 'QL'), ('cold', 'JJ'), ('.', '.')]
```

b. I had a cold.

In [26]:

```
text2 = word_tokenize("I had cold.")
bigram_tagger.tag(text2)
```

Out[26]:

```
[('I', 'PPSS'), ('had', 'HVD'), ('cold', None), ('.', None)]
```

c. I had a severe cold.

In [27]:

```
text3 = word_tokenize("I had a severe cold.")
bigram_tagger.tag(text3)
```

Out[27]:

```
[('I', 'PPSS'),
 ('had', 'HVD'),
 ('a', 'AT'),
 ('severe', 'JJ'),
 ('cold', 'JJ'),
 ('.', '.')]
```

d. January was a cold month.

In [28]:

```
text4 = word_tokenize("January was a cold month")
bigram_tagger.tag(text4)
```

Out[28]:

```
[('January', None),
 ('was', None),
 ('a', None),
 ('cold', None),
 ('month', None)]
```

7. Verify your prediction by having the tagger actually tag the four sentences. What did you find?

In [ ]:

```

```

8. Have the tagger tag the following sentences, all of which contain the word 'so':

a. I failed to do so.

In [29]:

```
text5 = word_tokenize("I failed to do so")
bigram_tagger.tag(text5)
```

Out[29]:

```
[('I', 'PPSS'), ('failed', 'VBD'), ('to', 'TO'), ('do', 'DO'), ('so', 'RB')]
```

b. I was happy, but so was my enemy

In [30]:

```
text6 = word_tokenize("I was happy,but so was my enemy")
bigram_tagger.tag(text6)
```

Out[30]:

```
[('I', 'PPSS'),
 ('was', 'BEDZ'),
 ('happy', 'JJ'),
 (',', ','),
 ('but', 'CC'),
 ('so', 'RB'),
 ('was', 'BEDZ'),
 ('my', 'PP$'),
 ('enemy', 'NN')]
```

c. So, how was the exam?

In [31]:

```
text7 = word_tokenize("So, how was the exam?")
bigram_tagger.tag(text7)
```

Out[31]:

```
[('So', 'RB'),
 (',', ','),
 ('how', 'WRB'),
 ('was', 'BEDZ'),
 ('the', 'AT'),
 ('exam', None),
 ('?', None)]
```

d. The students came in early so they can get good seats.

In [32]:

```
text8 = word_tokenize("The students came in early so they can get good seats")
bigram_tagger.tag(text8)
```

Out[32]:

```
[('The', 'AT'),
 ('students', 'NNS'),
 ('came', 'VBD'),
 ('in', 'IN'),
 ('early', 'JJ'),
 ('so', 'CS'),
 ('they', 'PPSS'),
 ('can', 'MD'),
 ('get', 'VB'),
 ('good', 'JJ'),
 ('seats', 'NNS')]
```

e. She failed the exam, so she must take it again.

In [33]:

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

Out[33]:

```
[('She', 'PPS'),
 ('failed', 'VBD'),
 ('the', 'AT'),
 ('exam', None),
 (',', None),
 ('so', None),
 ('she', None),
 ('must', None),
 ('take', None),
 ('it', None),
 ('again', None)]
```

f. That was so incredible.

In [34]:

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

Out[34]:

```
[('That', 'DT'), ('was', 'BEDZ'), ('so', 'QL'), ('incredible', 'JJ')]
```

g. Wow, so incredible.

In [35]:

```
text11 = word_tokenize("Wow, so incredible")
bigram_tagger.tag(text11)
```

Out[35]:

```
[('Wow', None), (',', None), ('so', None), ('incredible', None)]
```

9. Examine the tagger's performance on the sentences, focusing on the word 'so'. For each of them, decide if the tagger's output is correct, and explain how the tagger determined the POS tag.

In [ ]:

```
```

10.Based on what you have observed so far, offer a critique on the bigram tagger. What are its strengths and what are its limitations?

In [ ]:

```
```