Categorizing and POS Tagging with NLTK Python

Natural language processing is a sub-area of computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (native) languages. This is nothing but how to program computers to process and analyze large amounts of natural language data.

A part-of-speech tagger, or POS-tagger, processes a sequence of words and attaches a part of speech tag to each word. To do this first we have to use tokenization concept (Tokenization is the process by dividing the quantity of text into smaller parts called tokens.)

In [ ]:

```
import nltk
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
from nltk.tokenize import word_tokenize
text = word_tokenize("Hello welcome to the world of to learn Categorizing and POS Tagging with NLTK and Python")
nltk.pos_tag(text)
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

Out[ ]:

```
[('Hello', 'NNP'),
 ('welcome', 'NN'),
 ('to', 'TO'),
 ('the', 'DT'),
 ('world', 'NN'),
 ('of', 'IN'),
 ('to', 'TO'),
 ('learn', 'VB'),
 ('Categorizing', 'NNP'),
 ('and', 'CC'),
 ('POS', 'NNP'),
 ('Tagging', 'NNP'),
 ('with', 'IN'),
 ('NLTK', 'NNP'),
 ('and', 'CC'),
 ('Python', 'NNP')]
```

Tagged Corpora

Representing Tagged Tokens

A tagged token is represented using a tuple consisting of the token and the tag. We can create one of these special tuples from the standard string representation of a tagged token, using the function str2tuple():

In [ ]:

```
tagged_token = nltk.tag.str2tuple('Learn/VB')
tagged_token
```

Out[ ]:

```
('Learn', 'VB')
```

In [ ]:

```
tagged_token[0]
```

Out[ ]:

```
'Learn'
```

In [ ]:

```
tagged_token[1]
```

Out[ ]:

```
'VB'
```

Reading Tagged Corpora

Several of the corpora included with NLTK have been tagged for their part-of-speech.

Here's an example of what you might see if you opened a file from the Brown Corpus with a text editor:

In [ ]:

```
nltk.download('brown')
nltk.download('universal_tagset')
nltk.corpus.brown.tagged_words()
```

```
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package universal_tagset to /root/nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
```

Out[ ]:

```
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

In [ ]:

```
nltk.corpus.brown.tagged_words(tagset='universal')
```

Out[ ]:

```
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

Part of Speech Tagset

Tagged corpora use many different conventions for tagging words.

TagMeaningEnglish

ExamplesADJadjectivenew, good, high, special, big, localADPadpositionon, of, at, with, by, into, underADVadverbreally, already, still, early, nowCONJconjunctionand, or, but, if, while, althoughDETdeterminer, articles, a, some, most, every, no, whichNOUNnounyear, home, costs, time, AfricaNUMnumeraltwenty-four, fourth, 1991, 14:24PRTparticleat, on, out, over per, that, up, withPRONpronounhe, their, her, its, my, I, usVERBverbis, say, told, given, playing, would. punctuation marks. ,;!Xotherersatz, esprit, dunno, gr8, university

In [ ]:

```
from nltk.corpus import brown
brown_news_tagged = brown.tagged_words(categories='adventure', tagset='universal')
tag_fd = nltk.FreqDist(tag for (word, tag) in brown_news_tagged)
tag_fd.most_common()
```

Out[ ]:

```
[('NOUN', 13354),
 ('VERB', 12274),
 ('.', 10929),
 ('DET', 8155),
 ('ADP', 7069),
 ('PRON', 5205),
 ('ADV', 3879),
```

```
('ADJ', 3364),
('PRT', 2436),
('CONJ', 2173),
('NUM', 466),
('X', 38)]
```

Nouns

Nouns generally refer to people, places, things, or concepts, for example.: woman, Scotland, book, intelligence. The simplified noun tags are N for common nouns like a book, and NP for proper nouns like Scotland.

In [ ]:

```
word_tag_pairs = nltk.bigrams(brown_news_tagged)
noun_preceders = [a[1] for (a, b) in word_tag_pairs if b[1] == 'NOUN']
fdist = nltk.FreqDist(noun_preceders)
[tag for (tag, _) in fdist.most_common()]
```

Out[ ]:

```
['DET',
 'ADJ',
 'NOUN',
 'ADP',
 '.',
 'VERB',
 'CONJ',
 'NUM',
 'ADV',
 'PRON',
 'PRT',
 'X']
```

POS Tagging

Part-of-Speech or PoS tagging, then it may be defined as the process of assigning one of the parts of speech to the given word.

The parts of speech include nouns, verb, adverbs, adjectives, pronouns, conjunction and their sub-categories.

NN stands for noun, NNP stands for proper noun, VB for verb and so on. For e.g. In "Can you please buy me an Arizona tea" , Arizona is a proper noun.

There are various PoS tagging techniques:

1.Lexical based Methods

2.Rule based Methods

3.Probablisitic Methods

4.Deep Learning Methods

In [35]:

```
from nltk import pos_tag, word_tokenize
```

In [36]:

```
text = "Can you please buy me an Arizona tea"
tokens = word_tokenize(text)
tokens
```

Out[36]:

```
['Can', 'you', 'please', 'buy', 'me', 'an', 'Arizona', 'tea']
```

In [37]:

```
t = pos_tag(tokens)
```

```
t
```

Out[37]:

```
[('Can', 'MD'),
 ('you', 'PRP'),
 ('please', 'VB'),
 ('buy', 'VB'),
 ('me', 'PRP'),
 ('an', 'DT'),
 ('Arizona', 'NNP'),
 ('tea', 'NN')]
```

CRF (Conditional Random Fields)

CRF or Conditional Random Fields is a discriminant model for sequences data similar to MEMM(Maximum Entropy Markov Model). It models the dependency between each state and the entire input sequences.CRF overcomes the label bias issue by using global normalizer.

In CRF, a set of feature functions are defined to extract features for each word in a sentence. Some examples of feature functions are: is the first letter of the word capitalised, what the suffix and prefix of the word, what is the previous word, is it the first or the last word of the sentence, is it a number etc.

Uses of CRF

Gene prediction, NLP Part of Speech (POS) Tagging, NLP Named Entity Recognition (NER).

Importing the dataset and analyzing it.

In [38]:

```
tagged_sentence = nltk.corpus.treebank.tagged_sents(tagset='universal')
print("Number of Tagged Sentences ",len(tagged_sentence))
```

```
Number of Tagged Sentences  3914
```

In [39]:

```
tagged_words=[tup for sent in tagged_sentence for tup in sent]
print("Total Number of Tagged words", len(tagged_words))
```

```
Total Number of Tagged words 100676
```

In [40]:

```
vocab=set([word for word,tag in tagged_words])
print("Vocabulary of the Corpus",len(vocab))
```

```
Vocabulary of the Corpus 12408
```

In [41]:

```
tags=set([tag for word,tag in tagged_words])
print("Number of Tags in the Corpus ",len(tags))
```

```
Number of Tags in the Corpus  12
```

Splitting the dataset.

In [42]:

```
from sklearn.model_selection import train_test_split
train_set, test_set = train_test_split(tagged_sentence,test_size=0.2,random_state=1234)
```

Train Size

In [43]:

```
len(train_set)
```

Out[43]:

```
3131
```

Test Size

In [44]:

```
len(test_set)
```

Out[44]:

```
783
```

Creating the feature function

In [46]:

```
import re
def features(sentence,index):
  ### sentence is of the form [w1,w2,w3,..], index is the position of the word in the sen
tence
  return {
    'is_first_capital':int(sentence[index][0].isupper()),
  'is_first_word': int(index==0),
    'is_last_word':int(index==len(sentence)-1),
    'is_complete_capital': int(sentence[index].upper()==sentence[index]),
    'prev_word':'' if index==0 else sentence[index-1],
    'next_word':'' if index==len(sentence)-1 else sentence[index+1],
    'is_numeric':int(sentence[index].isdigit()),
    'is_alphanumeric': int(bool((re.match('^(?=.*[0-9]$)(?=.*[a-zA-Z])',sentence[index]))
)),
    'prefix_1':sentence[index][0],
    'prefix_2': sentence[index][:2],
    'prefix_3':sentence[index][:3],
    'prefix_4':sentence[index][:4],
    'suffix_1':sentence[index][-1],
    'suffix_2':sentence[index][-2:],
    'suffix_3':sentence[index][-3:],
    'suffix_4':sentence[index][-4:],
    'word_has_hyphen': 1 if '-' in sentence[index] else 0
}
def untag(sentence):
  return [word for word,tag in sentence]
def prepareData(tagged_sentences):
  X,y=[],[]
  for sentences in tagged_sentences:
    X.append([features(untag(sentences), index) for index in range(len(sentences))])
    y.append([tag for word,tag in sentences])
  return X,y
```

In [47]:

```
X_train,y_train=prepareData(train_set)
print(len(X_train))

X_test,y_test=prepareData(test_set)
print(len(X_test))
```

```
3131
783
```

Installing.

In [48]:

```
!pip install sklearn-crfsuite
```

```
Collecting sklearn-crfsuite
  Downloading https://files.pythonhosted.org/packages/25/74/5b7befa513482e6dee1f3dd68171a
6c9dfc14c0eaa00f885ffeba54fe9b0/sklearn_crfsuite-0.3.6-py2.py3-none-any.whl
Collecting python-crfsuite>=0.8.3
  Downloading https://files.pythonhosted.org/packages/95/99/869dde6dbf3e0d07a013c8eebfb0a
3d30776334e0097f8432b631a9a3a19/python_crfsuite-0.9.7-cp36-cp36m-manylinux1_x86_64.whl (7
43kB)
     |████████████████████████████████| 747kB 3.8MB/s
Requirement already satisfied: tabulate in /usr/local/lib/python3.6/dist-packages (from s
klearn-crfsuite) (0.8.7)
Requirement already satisfied: tqdm>=2.0 in /usr/local/lib/python3.6/dist-packages (from
sklearn-crfsuite) (4.41.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from sklear
n-crfsuite) (1.15.0)
Installing collected packages: python-crfsuite, sklearn-crfsuite
Successfully installed python-crfsuite-0.9.7 sklearn-crfsuite-0.3.6
```

Fitting the model

In [50]:

```
from sklearn_crfsuite import CRF
crf = CRF(
  algorithm='lbfgs',
  c1=0.01,
  c2=0.1,
  max_iterations=100,
  all_possible_transitions=True
)
crf.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/base.py:197: FutureWarning: From version 0
.24, get_params will raise an AttributeError if a parameter cannot be retrieved as an ins
tance attribute. Previously it would return None.
  FutureWarning)
```

Out[50]:

```
CRF(algorithm='lbfgs', all_possible_states=None, all_possible_transitions=True,
    averaging=None, c=None, c1=0.01, c2=0.1, calibration_candidates=None,
    calibration_eta=None, calibration_max_trials=None, calibration_rate=None,
    calibration_samples=None, delta=None, epsilon=None, error_sensitive=None,
    gamma=None, keep_tempfiles=None, linesearch=None, max_iterations=100,
    max_linesearch=None, min_freq=None, model_filename=None, num_memories=None,
    pa_type=None, period=None, trainer_cls=None, variance=None, verbose=False)
```

Evaluating

In [51]:

```
from sklearn_crfsuite import metrics
from sklearn_crfsuite import scorers
y_pred=crf.predict(X_test)
y_pred_train=crf.predict(X_train)

print("F1 score on Test Data ")
print(metrics.flat_f1_score(y_test, y_pred,average='weighted',labels=crf.classes_))

print("F1 score on Training Data ")
metrics.flat_f1_score(y_train, y_pred_train,average='weighted',labels=crf.classes_)
```

```
F1 score on Test Data
0.9738471726864286
F1 score on Training Data
```

Out[51]:

0.9963402924209424

In [52]:

```
print(metrics.flat_classification_report(y_test, y_pred, labels=crf.classes_, digits=3))
```

```
              precision    recall  f1-score   support

         ADP      0.979     0.985     0.982      1869
        NOUN      0.966     0.977     0.972      5606
        CONJ      0.994     0.994     0.994       480
        VERB      0.964     0.960     0.962      2722
         ADJ      0.911     0.874     0.892      1274
           .      1.000     1.000     1.000      2354
           X      1.000     0.997     0.998      1278
         NUM      0.991     0.993     0.992       671
         DET      0.994     0.995     0.994      1695
         ADV      0.927     0.909     0.918       585
        PRON      0.998     0.998     0.998       562
         PRT      0.979     0.982     0.980       614

    accuracy                          0.974     19710
   macro avg      0.975     0.972     0.974     19710
weighted avg      0.974     0.974     0.974     19710
```