CMPE 409 Machine Translation

Part-of-Speech (POS) Tagging

Murat ORHUN

Istanbul Bilgi University

April 20, 2022

Introduction

- Part-of-speech tagging is the process of converting a sentence, in the form of a list of words, into a list of tuples, where each tuple is of the form (word, tag).
- The tag is a part-of-speech tag, and signifies whether the word is a noun, adjective, verb, and so on.
- Necessary step for chunking, grammar analysis and word sens disambiguation

Introduction Cont.

- Default tagging
- Trainable taggers
- Evaluate
- All taggers in NLTK are in the nltk.tag package
- Backoff chaining
- Third part libraries

Default Tagging

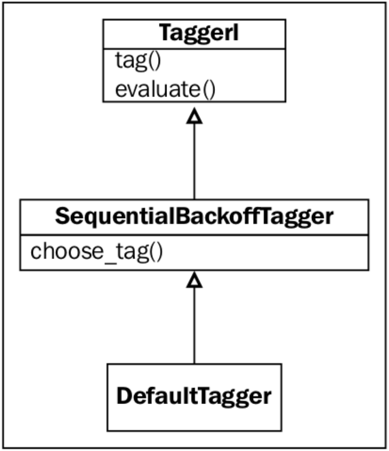
- Default tagging provides a baseline for part-of-speech tagging
- It simply assigns the same part-of-speech tag to every token.
- **DefaultTagger** class

Default Tagging

```
>>> from nltk.tag import DefaultTagger
>>> tagger = DefaultTagger('NN')
>>> tagger.tag(['Hello', 'World'])
[('Hello', 'NN'), ('World', 'NN')]
```

- Dafault tagge may take list of tags
- the tag() method takes list of words (tokenized)
- tag() method return list of **tuple**s.

Default Tagging



Part-of-speech tags used in the Penn Treebank Project

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun

Part-of-speech tags used in the Penn Treebank Project

19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Evaluation

```
>>> from nltk.corpus import treebank
>>> test_sents = treebank.tagged_sents()[3000:]
>>> tagger.evaluate(test_sents)
0.14331966328512843
```

We may have different result if we change tags of the tagger

Tagging sentences

```
>>> tagger.tag_sents([['Hello', 'world', '.'],
['How', 'are', 'you', '?']])
[[('Hello', 'NN'), ('world', 'NN'), ('.', 'NN')],
[('How', 'NN'), ('are', 'NN'), ('you', 'NN'), ('?','NN')]]
```

In this case we use the "tag_sents()" method

Untagging sentences

```
>>> from nltk.tag import untag
>>> untag([('Hello', 'NN'), ('World', 'NN')])
['Hello', 'World']
```

Tegh **untag** takes list of tagged **tuple**s

Unigarm Tagger

```
>>> from nltk.tag import UnigramTagger
>>> from nltk.corpus import treebank
>>> treebank.sents()[0]
['Pierre', 'Vinken', ',', '61', 'years', 'old', ',',
'will', 'join','the', 'board', 'as', 'a', 'nonexecutive'
, 'director', 'Nov.', '29','.']
```

```
text= treebank.sents()[0]
```

Note: treebank.sents(): gives list of sentences

Unigarm Tagger

```
>>> from nltk.tag import UnigramTagger
>>> from nltk.corpus import treebank
text= treebank.sents()[0]
treebank.tagged_sents()
train_sents= tagged_sents()[:3000]

#create the tagger with tagged words
tagger = UnigramTagger(train_sents)
```

Note: treebank.tagged_sents() and treebank.sents()

Unigarm Tagger

```
>>> from nltk.tag import UnigramTagger
>>> from nltk.corpus import treebank
text= treebank.sents()[0]
train_sents= tagged_sents()[:3000]
#create the tagger with tagged words
tagger = UnigramTagger(train_sents)
result= tagger.tag(text)
```

Note: See creation of the tagger, then the "text"

Unigarm Tagger

Part of tagged words

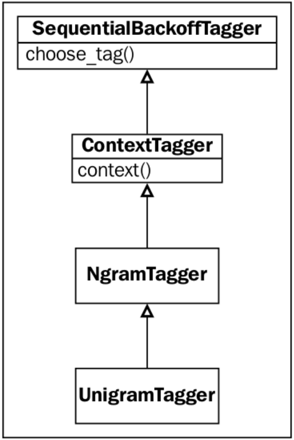
```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','),
('61', 'CD'),('years', 'NNS'), ('old', 'JJ'),
(',', ','), ('will', 'MD'), ('join','VB'),
('the', 'DT'), ('board', 'NN'),
('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'),
('director', 'NN'), ('Nov.', 'NNP'), ('29',
'CD'), ('.', '.')]
```

Discuss outputs...

Evaluate Unigram Tagger Part of tagged words

```
>>> test_sents = treebank.tagged_sents()[3000:]
>>> tagger.evaluate(test_sents)
0.8588819339520829
```

Unigram Model



Unigram Model

All taggers that inherit from ContextTagger can take a pre-built model instead of training

```
>>> tagger = UnigramTagger(model={'Pierre': 'NN'})
>>> tagger.tag(treebank.sents()[0])
[('Pierre', 'NN'), ('Vinken', None), (',', None),
('61', None),('years', None), ('old', None),........
```

Further Reading

- **Backoff tagging** is one of the core features of **SequentialBackoffTagger**.
- It allows you to chain taggers together so that if one tagger doesn't know how to tag a word
- it can pass the word on to the next backoff tagger

Further Reading

```
>>> tagger1 = DefaultTagger('NN')
>>> tagger2 = UnigramTagger(train_sents, backoff=tagger1)
>>> tagger2.evaluate(test_sents)
0.8758471832505935
```

Note: **backoff=tagger1**

Saving and Loading a tagger

```
>>> import pickle
>>> f = open('tagger.pickle', 'wb')
>>> pickle.dump(tagger, f)
>>> f.close()
>>> f = open('tagger.pickle', 'rb')
>>> tagger = pickle.load(f)
```

Note: we use **pickle** here

Bigram and Trigram taggers

```
>>> from nltk.tag import BigramTagger, TrigramTagger
>>> bitagger = BigramTagger(train_sents)
>>> bitagger.evaluate(test_sents)
0.11310166199007123

>>> tritagger = TrigramTagger(train_sents)
>>> tritagger.evaluate(test_sents)
0.0688107058061731
```

Note: See performance

Combine ngram taggers

```
def backoff_tagger(train_sents, tagger_classes,
                                                   backoff=None):
        for cls in tagger_classes:
            backoff = cls(train_sents, backoff=backoff)
    return backoff
```

Note: This method is defined in "tag_util.py" file
https://github.com/japerk/nltk3-cookbook

Combine ngram taggers

```
>>> backoff = DefaultTagger('NN')
>>> tagger = backoff_tagger(train_sents,
                            [UnigramTagger,
                             BigramTagger,
                             TrigramTagger],
                             backoff=backoff)
>>> tagger.evaluate(test_sents)
0.8806820634578028
```

Note: This method is defined in "tag_util.py" file
https://github.com/japerk/nltk3-cookbook

Quadgram tagger

The NgramTagger class can be used by itself to create a tagger that uses more than three ngrams for its context key.

```
>>> from nltk.tag import NgramTagger
>>> quadtagger = NgramTagger(4, train_sents)
>>> quadtagger.evaluate(test_sents)
0.058234405352903085
```

Note: see performance

Combine Quadgram tagger

```
>>> from taggers import QuadgramTagger
>>> quadtagger = backoff_tagger(train_sents,
                          [UnigramTagger,
                           BigramTagger,
                           TrigramTagger,
                           QuadgramTagger],
                          backoff=backoff)
>>> quadtagger.evaluate(test_sents)
0.8806388948845241
```

Note: **QuadgramTagger** is defined in "**taggers.py**" file
https://github.com/japerk/nltk3-cookbook

Creating a model of likely word tags

```
>>> from tag_util import word_tag_model
>>> from nltk.corpus import treebank
>>> model = word_tag_model(treebank.words(),
                           treebank.tagged_words())
>>> tagger = UnigramTagger(model=model)
>>> tagger.evaluate(test_sents)
0.559680552557738
```

Note: word_tag _model has been implmented in tag_util.py file
https://github.com/japerk/nltk3-cookbook

Creating a model of likely word tags with backoff chaining

```
>>> default_tagger = DefaultTagger('NN')
>>> likely_tagger = UnigramTagger(model=model,
                                  backoff=default_tagger)
>>> tagger = backoff_tagger(train_sents,
                            [UnigramTagger,
                             BigramTagger,
                             TrigramTagger],
                             backoff=likely_tagger)
>>> tagger.evaluate(test_sents)
0.8806820634578028
```

Note: See performance again

Tagging with Regular Expression

```
patterns = [
(r'^\d+$', 'CD'),
(r'.*ing$', 'VBG'), # gerunds, i.e. wondering
(r'.*ment$', 'NN'), # i.e. wonderment
(r'.*ful$', 'JJ') # i.e. wonderful
]
```

Note: this pattern can be found in tag_util.py
https://github.com/japerk/nltk3-cookbook

Tagging with Regular Expression

```
>>> from tag_util import patterns
>>> from nltk.tag import RegexpTagger
>>> tagger = RegexpTagger(patterns)
>>> tagger.evaluate(test_sents)
0.037470321605870924
```

Note: this pattern can be found in tag _util.py
https://github.com/japerk/nltk3-cookbook

Using WordNet for Tagging

WordNet tag	Treebank tag
n	NN
a	JJ
s	JJ
r	RB
v	VB

Using WordNet for Tagging

```
from nltk.tag import SequentialBackoffTagger
from nltk.corpus import wordnet
from nltk.probability import FreqDist

class WordNetTagger(SequentialBackoffTagger):
  '''
  >>> wt = WordNetTagger()
  >>> wt.tag(['food', 'is', 'great'])
  [('food', 'NN'), ('is', 'VB'), ('great', 'JJ')]
  '''
  def __init__(self, *args, **kwargs):
    SequentialBackoffTagger.__init__(self, *args, **kwargs)

    self.wordnet_tag_map = {
      'n': 'NN',
      's': 'JJ',
      'a': 'JJ',
      'r': 'RB',
```

Using WordNet for Tagging

```
def choose_tag(self, tokens, index, history):
  word = tokens[index]
  fd = FreqDist()

  for synset in wordnet.synsets(word):
    fd[synset.pos()] += 1

  return self.wordnet_tag_map.get(fd.max())
```

Using WordNet for Tagging

```
>>> from taggers import WordNetTagger
>>> wn_tagger = WordNetTagger()
>>> wn_tagger.evaluate(train_sents)
0.17914876598160262
```

Note: Discuss the result

Using WordNet with backoff chaining

```
>>> from tag_util import backoff_tagger
>>> from nltk.tag import UnigramTagger, BigramTagger,
                                         TrigramTagger
>>> tagger = backoff_tagger(train_sents, [UnigramTagger,
                                            BigramTagger,
                                            TrigramTagger],
                                         backoff=wn_tagger)
>>> tagger.evaluate(test_sents)
0.8848262464925534
```

Note: Discuss the result

Tagging Proper Names

```
from nltk.tag import SequentialBackoffTagger
from nltk.corpus import names

class NamesTagger(SequentialBackoffTagger):
  def __init__(self, *args, **kwargs):
    SequentialBackoffTagger.__init__(self, *args, **kwargs)
    self.name_set = set([n.lower() for n in names.words()])

    def choose_tag(self, tokens, index, history):
      word = tokens[index]

      if word.lower() in self.name_set:
        return 'NNP'
      else:
        return None
```

This code can be found in taggers.py :
https://github.com/japerk/nltk3-cookbook

Tagging Proper Names

```
>>> from taggers import NamesTagger
>>> nt = NamesTagger()
>>> nt.tag(['Jacob'])
[('Jacob', 'NNP')]
```

Try this with some Turkish names.

Other Taggers

- Affix Tagger
- Prefix
- Suffix
- Brill Tagger
- TnT tagger

Training a tagger with NLTK-Trainer

- there are many different ways to train taggers
- Which one is good?
- Training experiments can be tedious
- use the NLTK-Trainer

https://nltk-trainer.readthedocs.io/en/latest/

Training a tagger with NLTK-Trainer

The simplest way to run train_tagger.py is with the name of an NLTK corpus.

```
python train_tagger.py treebank
...........
...............
dumping TrigramTagger to /Users/jacob/nltk_data/
taggers/treebank_aubt.pickle
```

Training a tagger with NLTK-Trainer

Read chapter 4 of the reference.

Recourse

- Jacob Perkins, **Python 3 Text Processing with NLTK 3 Cookbook, Packt Publishing**,ISBN: 9781782167853