Text Mining Online | Text Analysis Online | Text Processing Online

Text Mining | Text Analysis | Text Process | Natural Language Processing

Dive Into NLTK, Part III: Part-Of-Speech Tagging and POS Tagger

This is the third article in the series "Dive Into NLTK", here is an index of all the articles in the series that have been published to date:

Part-of-speech tagging is one of the most important text analysis tasks used to classify words into their part-of-speech and label them according the tagset which is a collection of tags used for the pos tagging. Part-of-speech tagging also known as word classes or lexical categories. Here is the definition from wikipedia:

> *In corpus linguistics, part-of-speech tagging (POS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition, as well as its context—i.e. relationship with adjacent and related words in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc.*
>
> *Once performed by hand, POS tagging is now done in the context of computational linguistics, using algorithms which associate discrete terms, as well as hidden parts of speech, in accordance with a set*

of descriptive tags. POS-tagging algorithms fall into two distinctive groups: rule-based and stochastic. E. Brill's tagger, one of the first and most widely used English POS-taggers, employs rule-based algorithms.

How to use POS Tagging in NLTK

After import NLTK in python interpreter, you should use word_tokenize before pos tagging, which referred as pos_tag method:

```
>>> import nltk
>>> text = nltk.word_tokenize("Dive into NLTK: Part-of-speech tagging and POS Tagger")
>>> text
['Dive', 'into', 'NLTK', ':', 'Part-of-speech', 'tagging', 'and', 'POS', 'Tagger']
>>> nltk.pos_tag(text)
[('Dive', 'JJ'), ('into', 'IN'), ('NLTK', 'NNP'), (':', ':'), ('Part-of-speech', 'JJ'), ('tagging', 'NN'), ('and', 'CC'), ('POS', 'NNP'),
('Tagger', 'NNP')]
```

NLTK provides documentation for each tag, which can be queried using the tag, e.g., nltk.help.upenn_tagset('RB'), or a regular expression, e.g., nltk.help.upenn_brown_tagset('NN.*'):

```
>>> nltk.help.upenn_tagset('JJ')
JJ: adjective or numeral, ordinal
third ill-mannered pre-war regrettable oiled calamitous first separable
ectoplasmic battery-powered participatory fourth still-to-be-named
multilingual multi-disciplinary …
>>> nltk.help.upenn_tagset('IN')
IN: preposition or conjunction, subordinating
astride among uppon whether out inside pro despite on by throughout
below within for towards near behind atop around if like until below
next into if beside …
>>> nltk.help.upenn_tagset('NNP')
NNP: noun, proper, singular
Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos
Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA
Shannon A.K.C. Meltex Liverpool …
>>>
```

NLTK also provide batch pos tagging method for document pos tagging, which is batch_pos_tag method:

```
>>> nltk.batch_pos_tag([['this', 'is', 'batch', 'tag', 'test'], ['nltk', 'is', 'text', 'analysis', 'tool']])
```

[[('this', 'DT'), ('is', 'VBZ'), ('batch', 'NN'), ('tag', 'NN'), ('test', 'NN')], [('nltk', 'NN'), ('is', 'VBZ'), ('text', 'JJ'), ('analysis', 'NN'), ('tool', 'NN')]]
>>>

The pre-trained POS Tagger Model in NLTK

You can find the pre-trained POS Tagging Model in nltk_data/taggers:

YangtekiMacBook-Pro:taggers textminer$ pwd
/Users/textminer/nltk_data/taggers
YangtekiMacBook-Pro:taggers textminer$ ls
total 11304
drwxr-xr-x 3 textminer staff 102 7 9 14:06 hmm_treebank_pos_tagger
-rw-r–r– 1 textminer staff 750857 5 26 2013 hmm_treebank_pos_tagger.zip
drwxr-xr-x 3 textminer staff 102 7 24 2013 maxent_treebank_pos_tagger
-rw-r–r– 1 textminer staff 5031883 5 26 2013 maxent_treebank_pos_tagger.zip

The default pos tagger model using in NLTK is maxent_treebanck_pos_tagger model, you can find the code in nltk-master/nltk/tag/__init__.py:

```
# Standard treebank POS tagger
_POS_TAGGER = 'taggers/maxent_treebank_pos_tagger/english.pickle'
def pos_tag(tokens):
    """
    Use NLTK's currently recommended part of speech tagger to
    tag the given list of tokens.

        >>> from nltk.tag import pos_tag # doctest: +SKIP
        >>> from nltk.tokenize import word_tokenize # doctest: +SKIP
        >>> pos_tag(word_tokenize("John's big idea isn't all that bad.")) # doctest: +SKIP
        [('John', 'NNP'), ("'s", 'POS'), ('big', 'JJ'), ('idea', 'NN'), ('is',
        'VBZ'), ("n't", 'RB'), ('all', 'DT'), ('that', 'DT'), ('bad', 'JJ'),
        ('.', '.')]

    :param tokens: Sequence of tokens to be tagged
    :type tokens: list(str)
    :return: The tagged tokens
    :rtype: list(tuple(str, str))
    """
    tagger = load(_POS_TAGGER)
    return tagger.tag(tokens)

def batch_pos_tag(sentences):
    """
    Use NLTK's currently recommended part of speech tagger to tag the
    given list of sentences, each consisting of a list of tokens.
    """
    tagger = load(_POS_TAGGER)
    return tagger.batch_tag(sentences)
```

How to train a POS Tagging Model or POS Tagger in NLTK

You have used the maxent treebank pos tagging model in NLTK by default, and NLTK provides not only the maxent pos tagger, but other pos taggers like crf, hmm, brill, tnt and interfaces with stanford pos tagger, hunpos pos tagger and senna postaggers:

-rwxr-xr-x@ 1 textminer staff 4.4K 7 22 2013 __init__.py
-rwxr-xr-x@ 1 textminer staff 2.9K 7 22 2013 api.py
-rwxr-xr-x@ 1 textminer staff 56K 7 22 2013 brill.py
-rwxr-xr-x@ 1 textminer staff 31K 7 22 2013 crf.py
-rwxr-xr-x@ 1 textminer staff 48K 7 22 2013 hmm.py
-rwxr-xr-x@ 1 textminer staff 5.1K 7 22 2013 hunpos.py
-rwxr-xr-x@ 1 textminer staff 11K 7 22 2013 senna.py
-rwxr-xr-x@ 1 textminer staff 26K 7 22 2013 sequential.py
-rwxr-xr-x@ 1 textminer staff 3.3K 7 22 2013 simplify.py
-rwxr-xr-x@ 1 textminer staff 6.4K 7 22 2013 stanford.py
-rwxr-xr-x@ 1 textminer staff 18K 7 22 2013 tnt.py
-rwxr-xr-x@ 1 textminer staff 2.3K 7 22 2013 util.py

Here we will show you how to train a TnT POS Tagger Model, you can find the details about TnT in tnt.py:

```
# Natural Language Toolkit: TnT Tagger
#
# Copyright (C) 2001-2013 NLTK Project
# Author: Sam Huston <sjh900@gmail.com>
#
# URL: <http://www.nltk.org/>
# For license information, see LICENSE.TXT

'''
Implementation of 'TnT - A Statisical Part of Speech Tagger'
by Thorsten Brants

http://acl.ldc.upenn.edu/A/A00/A00-1031.pdf
'''
from __future__ import print_function
from math import log

from operator import itemgetter

from nltk.probability import FreqDist, ConditionalFreqDist
from nltk.tag.api import TaggerI

class TnT(TaggerI):
    '''
    TnT - Statistical POS tagger

    IMPORTANT NOTES:

    * DOES NOT AUTOMATICALLY DEAL WITH UNSEEN WORDS

      - It is possible to provide an untrained POS tagger to
        create tags for unknown words, see __init__ function

    * SHOULD BE USED WITH SENTENCE-DELIMITED INPUT

      - Due to the nature of this tagger, it works best when
        trained over sentence delimited input.
      - However it still produces good results if the training
        data and testing data are separated on all punctuation eg: [,.?!]
      - Input for training is expected to be a list of sentences
        where each sentence is a list of (word, tag) tuples
      - Input for tag function is a single sentence
        Input for tagdata function is a list of sentences
        Output is of a similar form

    * Function provided to process text that is unsegmented

      - Please see basic_sent_chop()


    TnT uses a second order Markov model to produce tags for
    a sequence of input, specifically:

      argmax [Proj(P(t_i|t_i-1,t_i-2)P(w_i|t_i))] P(t_T+1 | t_T)

    IE: the maximum projection of a set of probabilities

    The set of possible tags for a given word is derived
    from the training data. It is the set of all tags
    that exact word has been assigned.

    To speed up and get more precision, we can use log addition
    to instead multiplication, specifically:

      argmax [Sigma(log(P(t_i|t_i-1,t_i-2))+log(P(w_i|t_i)))] +
             log(P(t_T+1|t_T))

    The probability of a tag for a given word is the linear
    interpolation of 3 markov models; a zero-order, first-order,
    and a second order model.

      P(t_i| t_i-1, t_i-2) = l1*P(t_i) + l2*P(t_i| t_i-1) +
                             l3*P(t_i| t_i-1, t_i-2)

    A beam search is used to limit the memory usage of the algorithm.
    The degree of the beam can be changed using N in the initialization.
    N represents the maximum number of possible solutions to maintain
```

```
while tagging.

It is possible to differentiate the tags which are assigned to
capitalized words. However this does not result in a significant
gain in the accuracy of the results.
'''
```

First you need the train dada and test data, we use the treebank data from nltk.corpus:

```
>>> from nltk.corpus import treebank
>>> len(treebank.tagged_sents())
3914
>>> train_data = treebank.tagged_sents()[:3000]
>>> test_data = treebank.tagged_sents()[3000:]
>>> train_data[0]
[(u'Pierre', u'NNP'), (u'Vinken', u'NNP'), (u',', u','), (u'61', u'CD'), (u'years', u'NNS'), (u'old', u'JJ'), (u',', u','), (u'will', u'MD'), (u'join', u'VB'), (u'the', u'DT'), (u'board', u'NN'), (u'as', u'IN'), (u'a', u'DT'), (u'nonexecutive', u'JJ'), (u'director', u'NN'), (u'Nov.', u'NNP'), (u'29', u'CD'), (u'.', u'.')]
>>> test_data[0]
[(u'At', u'IN'), (u'Tokyo', u'NNP'), (u',', u','), (u'the', u'DT'), (u'Nikkei', u'NNP'), (u'index', u'NN'), (u'of', u'IN'), (u'225', u'CD'), (u'selected', u'VBN'), (u'issues', u'NNS'), (u',', u','), (u'which', u'WDT'), (u'*T*-1', u'-NONE-'), (u'gained', u'VBD'), (u'132', u'CD'), (u'points', u'NNS'), (u'Tuesday', u'NNP'), (u',', u','), (u'added', u'VBD'), (u'14.99', u'CD'), (u'points', u'NNS'), (u'to', u'TO'), (u'35564.43', u'CD'), (u'.', u'.')]
>>>
```

We use the first 3000 treebank tagged sentences as the train_data, and last 914 tagged sentences as the test_data, now we train TnT POS Tagger by the train_data and evaluate it by the test_data:

```
>>> from nltk.tag import tnt
>>> tnt_pos_tagger = tnt.TnT()
>>> tnt_pos_tagger.train(train_data)
>>> tnt_pos_tagger.evaluate(test_data)
0.8755881718109216
```

You can save this pos tagger model as a pickle file:

```
>>> import pickle
>>> f = open('tnt_treebank_pos_tagger.pickle', 'w')
>>> pickle.dump(tnt_pos_tagger, f)
>>> f.close()
```

And you can use it any time you want:

```
>>> tnt_tagger.tag(nltk.word_tokenize("this is a tnt treebank tnt tagger"))
[('this', u'DT'), ('is', u'VBZ'), ('a', u'DT'), ('tnt', 'Unk'), ('treebank', 'Unk'), ('tnt', 'Unk'), ('tagger', 'Unk')]
>>>
```

That's it, now you can train your POS Tagging Model by yourself, just do it.

Posted by TextMiner

Related Posts:

1. **Getting Started with Pattern**
2. **Dive Into NLTK, Part V: Using Stanford Text Analysis Tools in Python**
3. **Dive Into NLTK, Part I: Getting Started with NLTK**
4. **Dive Into NLTK, Part VII: A Preliminary Study on Text Classification**

This entry was posted in NLTK, Text Analysis, Text Mining, Text Processing and tagged brill pos tagger, crf pos tagger, hmm pos tagger, maxent pos tagger, NLTK, nltk pos tagger, nltk pos tagging, Part-of-speech tagging, POS, POS Tagger, Pos Tagging, stanford pos tagger, tnt pos tagger, train a pos tagger on July 9, 2014 [http://textminingonline.com/dive-into-nltk-part-iii-part-of-speech-tagging-and-pos-tagger] .