CRFsuite - Tutorial on Chunking Task

Table of Contents

Task description

Text chunking divides a text into syntactically correlated parts of words. For example, the sentence "He reckons the current account deficit will narrow to only # 1.8 billion in September." can be divided as follows:

[NP He] [VP reckons] [NP the current account deficit] [VP will narrow] [PP to] [NP only # 1.8 billion] [PP in] [NP September] .

[frez]

In this example, NP stands for a noun phrase, VP for a verb phrase, and PP for a prepositional phrase. This task is formalized as a sequential labeling task in which a sequence of tokens in a text is assigned with a sequence of labels. In order to represent a chunk (a span of tokens) with labels, we often use the IOB2 notation. Using the IOB2 notation, a chunk NP is represented by a begin of a chunk (B-NP) and an inside of a chunk (I-NP). Tokens that do not belong to a chunk are represented by O labels.

```
B-NP He
B-VP reckons
B-NP the
I-NP current
I-NP account
I-NP deficit
B-VP will
I-VP narrow
B-PP to
B-NP only
I-NP #
I-NP 1.8
I-NP billion
B-PP in
B-NP September
O    .
```

The goal of this tutorial is to build a model that predicts chunk labels for a given sentence (sequence of tokens) by using CRFsuite.

Training and testing data

This tutorial uses the training and testing data distributed by the CoNLL 2000 shared task. Necessary scripts for this tutorial are included under `example` directory in the CRFsuite distribution. Firstly, move the current directory to the example directory and download the training and testing data from their website:

```
$ cd example
$ wget http://www.cnts.ua.ac.be/conll2000/chunking/train.txt.gz
$ wget http://www.cnts.ua.ac.be/conll2000/chunking/test.txt.gz
$ less train.txt.gz
... (snip) ...

London JJ B-NP
shares NNS I-NP
closed VBD B-VP
moderately RB B-ADVP
lower JJR I-ADVP
in IN B-PP
thin JJ B-NP
trading NN I-NP
. . O

At IN B-PP
Tokyo NNP B-NP
, , O
the DT B-NP
Nikkei NNP I-NP
index NN I-NP
of IN B-PP
225 CD B-NP
selected VBN I-NP
issues NNS I-NP
was VBD B-VP
up IN B-ADVP
112.16 CD B-NP
points NNS I-NP
to TO B-PP
35486.38 CD B-NP
. . O

... (snip) ...
```

The data consists of a set of sentences (sequences) each of which contains a series of words (e.g., 'London', 'shares'), part-of-speech tags (e.g., 'JJ', 'NNS'), and chunk labels (e.g., 'B-NP', 'I-NP') separated by space characters. In this tutorial, we would like to construct a CRF model that assigns a sequence of chunk labels, given a sequence of words and part-of-speech codes. Please refer to CoNLL 2000 shared task website for more information about the data set.

Feature (attribute) generation

['ekstrækt]

The next step is to preprocess the training and testing data to extract attributes that express the characteristics of words (items) in the data. CRFsuite internally generates features from attributes in a data set. In general, this is the most important process for machine-learning approaches because a feature design greatly affects the labeling accuracy. In this tutorial, we extract 19 kinds of attributes from a word at position t (in offsets from the begining of a sequence):

- `w[t-2]`, `w[t-1]`, `w[t]`, `w[t+1]`, `w[t+2]`,
- `w[t-1]|w[t]`, `w[t]|w[t+1]`,
- `pos[t-2]`, `pos[t-1]`, `pos[t]`, `pos[t+1]`, `pos[t+2]`,
- `pos[t-2]|pos[t-1]`, `pos[t-1]|pos[t]`, `pos[t]|pos[t+1]`, `pos[t+1]|pos[t+2]`,
- `pos[t-2]|pos[t-1]|pos[t]`, `pos[t-1]|pos[t]|pos[t+1]`, `pos[t]|pos[t+1]|pos[t+2]`

In this list, `w[t]` and `pos[t]` present the word and part-of-speech respectively at position t in a sequence. These features express the characteristic of the word at position t by using information from surrounding words, e.g., `w[t-1]` and `pos[t+1]`. For example, the token 'the' in the following example,

```
        He PRP B-NP
```

```
        reckons VBZ B-VP
  t --> the DT B-NP
        current JJ I-NP
        account NN I-NP
```

obtains these attributes (position t is omitted for simplicity),

- w[-2]=He, w[-1]=reckons, w[0]=the, w[1]=current, w[2]=account
- w[-1]|w[0]=reckons|the, w[0]|w[1]=the|current
- pos[-2]=PRP, pos[-1]=VBZ, pos[0]=DT, pos[1]=JJ, pos[2]=NN
- pos[-2]|pos[-1]=PRP|VBZ, pos[-1]|pos[0]=VBZ|DT, pos[0]|pos[1]=DT|JJ, pos[1]|pos[2]=JJ|NN
- pos[-2]|pos[-1]|pos[0]=PRP|VBZ|DT, pos[-1]|pos[0]|pos[1]=VBZ|DT|JJ, pos[0]|pos[1]|pos[2]=DT|JJ|NN

In this example, the attribute "w[0]=the" presents the event where the current token is "the", and the attribute "pos[0]|pos[1]|pos[2]=DT|JJ|NN" presents the event where the parts-of-speech at the current, next, and two words ahead are DT, JJ, NN, respectively. CRFsuite will learn associations between these attributes (e.g, "pos[0]|pos[1]|pos[2]=DT|JJ|NN") and labels (e.g., "B-NP") to predict a label sequence for a given text. Please note that an attribute need not follow the convention "name=value", e.g., "w[0]=the". *CRFsuite accepts any string as an attribute name as long as the string does not contain a colon character (that is used to separate an attribute name and its weight).* The convention "name=value" is merely for the convenience to interpret attribute names.

CRFsuite requires a data set in which an item line begins with its label, followed by its attributes separated by TAB ('\t') characters (see documentation for more information). It is not difficult to implement the conversion from the training/testing data to CRFsuite data. As an implementation of the conversion, the CRFsuite distribution includes a Python script **chunking.py** that generates attributes from the CoNLL 2000 data. The procedure below converts `train.txt.gz` and `test.txt.gz` into `train.crfsuite.txt` and `test.crfsuite.txt` that are compatible with the CRFsuite data format.

```
$ zcat train.txt.gz | ./chunking.py > train.crfsuite.txt
$ zcat test.txt.gz | ./chunking.py > test.crfsuite.txt
$ less train.crfsuite.txt
... (snip) ...

B-NP    w[0]=He w[1]=reckons    w[2]=the        w[0]|w[1]=He|reckons    pos[0]=P
RP      pos[1]=VBZ      pos[2]=DT       pos[0]|pos[1]=PRP|VBZ   pos[1]|pos[2]=VB
Z|DT    pos[0]|pos[1]|pos[2]=PRP|VBZ|DT __BOS__
B-VP    w[-1]=He        w[0]=reckons    w[1]=the        w[2]=current    w[-1]|w[
0]=He|reckons   w[0]|w[1]=reckons|the   pos[-1]=PRP     pos[0]=VBZ      pos[1]=D
T       pos[2]=JJ       pos[-1]|pos[0]=PRP|VBZ  pos[0]|pos[1]=VBZ|DT    pos[1]|p
os[2]=DT|JJ     pos[-1]|pos[0]|pos[1]=PRP|VBZ|DT        pos[0]|pos[1]|pos[2]=VBZ
|DT|JJ
B-NP    w[-2]=He        w[-1]=reckons   w[0]=the        w[1]=current    w[2]=acc
ount    w[-1]|w[0]=reckons|the  w[0]|w[1]=the|current   pos[-2]=PRP     pos[-1]=
VBZ     pos[0]=DT       pos[1]=JJ       pos[2]=NN       pos[-2]|pos[-1]=PRP|VBZ
pos[-1]|pos[0]=VBZ|DT   pos[0]|pos[1]=DT|JJ     pos[1]|pos[2]=JJ|NN     pos[-2]|
pos[-1]|pos[0]=PRP|VBZ|DT       pos[-1]|pos[0]|pos[1]=VBZ|DT|JJ pos[0]|pos[1]|po
s[2]=DT|JJ|NN
I-NP    w[-2]=reckons   w[-1]=the       w[0]=current    w[1]=account    w[2]=def
icit    w[-1]|w[0]=the|current  w[0]|w[1]=current|account       pos[-2]=VBZ
     pos[-1]=DT      pos[0]=JJ       pos[1]=NN       pos[2]=NN       pos[-2]|pos
[-1]=VBZ|DT  pos[-1]|pos[0]=DT|JJ    pos[0]|pos[1]=JJ|NN     pos[1]|pos[2]=NN|NN
     pos[-2]|pos[-1]|pos[0]=VBZ|DT|JJ        pos[-1]|pos[0]|pos[1]=DT|JJ|NN  pos
[0]|pos[1]|pos[2]=JJ|NN|NN
I-NP    w[-2]=the       w[-1]=current   w[0]=account    w[1]=deficit    w[2]=wil
l       w[-1]|w[0]=current|account      w[0]|w[1]=account|deficit       pos[-2]=
DT      pos[-1]=JJ      pos[0]=NN       pos[1]=NN       pos[2]=MD       pos[-2]|
pos[-1]=DT|JJ   pos[-1]|pos[0]=JJ|NN    pos[0]|pos[1]=NN|NN     pos[1]|pos[2]=NN
|MD     pos[-2]|pos[-1]|pos[0]=DT|JJ|NN pos[-1]|pos[0]|pos[1]=JJ|NN|NN  pos[0]|p
os[1]|pos[2]=NN|NN|MD

... (snip) ...
```

Training

Now we are ready to use CRFsuite for training. Simply type the following command to train a CRF model from `train.crfsuite.txt`. CRFsuite will read the training data, generate necessary state (attribute-label) and transition (label bigram) features based on the data, maximize the log-likelihood of the conditional probability distribution, and store the model into `CoNLL2000.model`.

```
$ crfsuite learn -m CoNLL2000.model train.crfsuite.txt
CRFSuite 0.12  Copyright (c) 2007-2011 Naoaki Okazaki

Start time of the training: 2011-06-25T14:52:13Z

Reading the data set(s)
[1] train.crfsuite.txt
0....1....2....3....4....5....6....7....8....9....10
Number of instances: 8937
Seconds required: 5.890

Statistics the data set(s)
Number of data sets (groups): 1
Number of instances: 8936
Number of items: 211727
Number of attributes: 335674
Number of labels: 22

Feature generation
type: CRF1d
feature.minfreq: 0.000000
feature.possible_states: 0
feature.possible_transitions: 0
0....1....2....3....4....5....6....7....8....9....10
Number of features: 452755
Seconds required: 2.140

L-BFGS optimization
c1: 0.000000
c2: 1.000000
num_memories: 6
max_iterations: 2147483647
epsilon: 0.000010
stop: 10
delta: 0.000010
linesearch: MoreThuente
linesearch.max_iterations: 20

***** Iteration #1 *****
Log-likelihood: -275528.648286
Feature norm: 5.000000
Error norm: 44363.015822
Active features: 452755
Line search trials: 2
Line search step: 0.000050
Seconds required for this iteration: 4.860

***** Iteration #2 *****
Log-likelihood: -164450.778877
Feature norm: 9.067189
Error norm: 26619.939310
Active features: 452755
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 1.630

... (snip) ...

***** Iteration #165 *****
Log-likelihood: -13139.375165
Feature norm: 81.074163
Error norm: 2.638386
Active features: 452755
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 1.660

L-BFGS terminated with the stopping criteria
Total seconds required for training: 293.610

Storing the model
Number of active features: 452755 (452755)
Number of active attributes: 335674 (335674)
Number of active labels: 22 (22)
Writing labels
```

```
Writing attributes
Writing feature references for transitions
Writing feature references for attributes
Seconds required: 0.730

End time of the training: 2011-06-25T14:57:15Z
```

You can also train a CRF model, watching its performance (accuracy, precision, recall, f1 score) evaluated on the test data. It must be exciting to see your model improved as the training process advances! The following command-line performs a holdout evaluation on the data set #2 (`test.crfsuite.txt`) with the option (-e2).

```
$ crfsuite learn -e2 train.crfsuite.txt test.crfsuite.txt
CRFSuite 0.12  Copyright (c) 2007-2011 Naoaki Okazaki

Start time of the training: 2011-06-25T16:07:40Z

Reading the data set(s)
[1] train.crfsuite.txt
0....1....2....3....4....5....6....7....8....9....10
Number of instances: 8937
Seconds required: 5.870
[2] test.crfsuite.txt
0....1....2....3....4....5....6....7....8....9....10
Number of instances: 2013
Seconds required: 1.370

Statistics the data set(s)
Number of data sets (groups): 2
Number of instances: 10948
Number of items: 259104
Number of attributes: 387579
Number of labels: 23

Holdout group: 2

Feature generation
type: CRF1d
feature.minfreq: 0.000000
feature.possible_states: 0
feature.possible_transitions: 0
0....1....2....3....4....5....6....7....8....9....10
Number of features: 452755
Seconds required: 2.150

L-BFGS optimization
c1: 0.000000
c2: 1.000000
num_memories: 6
max_iterations: 2147483647
epsilon: 0.000010
stop: 10
delta: 0.000010
linesearch: MoreThuente
linesearch.max_iterations: 20

***** Iteration #1 *****
Log-likelihood: -279935.059188
Feature norm: 5.000000
Error norm: 45136.783202
Active features: 452755
Line search trials: 2
Line search step: 0.000050
Seconds required for this iteration: 5.230
Performance by label (#match, #model, #ref) (precision, recall, F1):
    B-NP: (8312, 10265, 12422) (0.8097, 0.6691, 0.7328)
    B-PP: (3986, 6030, 4811) (0.6610, 0.8285, 0.7354)
    I-NP: (14116, 27744, 14376) (0.5088, 0.9819, 0.6703)
    B-VP: (0, 0, 4658) (0.0000, 0.0000, 0.0000)
    I-VP: (0, 0, 2646) (0.0000, 0.0000, 0.0000)
    B-SBAR: (0, 0, 535) (0.0000, 0.0000, 0.0000)
    O: (3298, 3338, 6180) (0.9880, 0.5337, 0.6930)
    B-ADJP: (0, 0, 438) (0.0000, 0.0000, 0.0000)
    B-ADVP: (0, 0, 866) (0.0000, 0.0000, 0.0000)
    I-ADVP: (0, 0, 89) (0.0000, 0.0000, 0.0000)
    I-ADJP: (0, 0, 167) (0.0000, 0.0000, 0.0000)
    I-SBAR: (0, 0, 4) (0.0000, 0.0000, 0.0000)
    I-PP: (0, 0, 48) (0.0000, 0.0000, 0.0000)
    B-PRT: (0, 0, 106) (0.0000, 0.0000, 0.0000)
```

```
    B-LST: (0, 0, 5) (0.0000, 0.0000, 0.0000)
    B-INTJ: (0, 0, 2) (0.0000, 0.0000, 0.0000)
    I-INTJ: (0, 0, 0) (******, ******, ******)
    B-CONJP: (0, 0, 9) (0.0000, 0.0000, 0.0000)
    I-CONJP: (0, 0, 13) (0.0000, 0.0000, 0.0000)
    I-PRT: (0, 0, 0) (******, ******, ******)
    B-UCP: (0, 0, 0) (******, ******, ******)
    I-UCP: (0, 0, 0) (******, ******, ******)
    I-LST: (0, 0, 2) (0.0000, 0.0000, 0.0000)
Macro-average precision, recall, F1: (0.129025, 0.131010, 0.123104)
Item accuracy: 29712 / 47377 (0.6271)
Instance accuracy: 23 / 2012 (0.0114)

... (snip) ...

***** Iteration #162 *****
Log-likelihood: -13143.933308
Feature norm: 81.103204
Error norm: 2.207139
Active features: 452755
Line search trials: 1
Line search step: 1.000000
Seconds required for this iteration: 1.980
Performance by label (#match, #model, #ref) (precision, recall, F1):
    B-NP: (12013, 12374, 12422) (0.9708, 0.9671, 0.9689)
    B-PP: (4713, 4878, 4811) (0.9662, 0.9796, 0.9729)
    I-NP: (13998, 14497, 14376) (0.9656, 0.9737, 0.9696)
    B-VP: (4470, 4668, 4658) (0.9576, 0.9596, 0.9586)
    I-VP: (2551, 2700, 2646) (0.9448, 0.9641, 0.9544)
    B-SBAR: (449, 499, 535) (0.8998, 0.8393, 0.8685)
    O: (5945, 6122, 6180) (0.9711, 0.9620, 0.9665)
    B-ADJP: (322, 403, 438) (0.7990, 0.7352, 0.7658)
    B-ADVP: (711, 836, 866) (0.8505, 0.8210, 0.8355)
    I-ADVP: (54, 82, 89) (0.6585, 0.6067, 0.6316)
    I-ADJP: (110, 137, 167) (0.8029, 0.6587, 0.7237)
    I-SBAR: (2, 15, 4) (0.1333, 0.5000, 0.2105)
    I-PP: (34, 42, 48) (0.8095, 0.7083, 0.7556)
    B-PRT: (80, 102, 106) (0.7843, 0.7547, 0.7692)
    B-LST: (0, 0, 5) (0.0000, 0.0000, 0.0000)
    B-INTJ: (1, 1, 2) (1.0000, 0.5000, 0.6667)
    I-INTJ: (0, 0, 0) (******, ******, ******)
    B-CONJP: (5, 8, 9) (0.6250, 0.5556, 0.5882)
    I-CONJP: (10, 13, 13) (0.7692, 0.7692, 0.7692)
    I-PRT: (0, 0, 0) (******, ******, ******)
    B-UCP: (0, 0, 0) (******, ******, ******)
    I-UCP: (0, 0, 0) (******, ******, ******)
    I-LST: (0, 0, 2) (0.0000, 0.0000, 0.0000)
Macro-average precision, recall, F1: (0.604705, 0.576296, 0.581536)
Item accuracy: 45468 / 47377 (0.9597)
Instance accuracy: 1176 / 2012 (0.5845)

L-BFGS terminated with the stopping criteria
Total seconds required for training: 339.800

End time of the training: 2011-06-25T16:13:29Z
```

This log message reports that the CRF model obtained from the training data achieved 95.97% item accuracy.

Tagging

You can apply the CRF model and tag chunk labels to the test data. Even though the test data distributed by the CoNLL 2000 shared task has chunk labels annotated (for evaluation purposes), CRFsuite ignores the existing labels and outputs label sequences (one label per line; delimitered by empty lines) predicted by the model.

```
$ cat test.crfsuite.txt
B-NP    w[0]=Rockwell   w[1]=International      w[2]=Corp.      w[0]|w[1]=Rockwe
ll|International        pos[0]=NNP      pos[1]=NNP      pos[2]=NNP      pos[0]|p
os[1]=NNP|NNP   pos[1]|pos[2]=NNP|NNP   pos[0]|pos[1]|pos[2]=NNP|NNP|NNP
          __BOS__
I-NP    w[-1]=Rockwell  w[0]=International      w[1]=Corp.      w[2]='s w[-1]|w[
0]=Rockwell|International       w[0]|w[1]=International|Corp.   pos[-1]=NNP
     pos[0]=NNP      pos[1]=NNP      pos[2]=POS      pos[-1]|pos[0]=NNP|NNP  pos
[0]|pos[1]=NNP|NNP   pos[1]|pos[2]=NNP|POS   pos[-1]|pos[0]|pos[1]=NNP|NNP|NNP
```

```
     pos[0]|pos[1]|pos[2]=NNP|NNP|POS
... (snip) ...

$ crfsuite tag -m CoNLL2000.model test.crfsuite.txt
B-NP
I-NP
I-NP
B-NP
I-NP
I-NP
B-VP
B-NP
B-VP
B-NP
I-NP
I-NP
B-VP
B-NP
I-NP
B-PP
B-NP
I-NP
B-VP
I-VP
B-NP
I-NP
B-PP
B-NP
B-NP
I-NP
I-NP
O

... (snip) ...
```

CRFsuite can output both reference labels (in `test.crfsuite.txt`) and predicted labels separated by TAB characters. In this example, a left label in each line presents the reference label written in the input data (`test.crfsuite.txt`), ,and a right label presents the predicted label. This functionality may be useful for evaluating tagging results.

```
$ crfsuite tag -r -m CoNLL2000.model test.crfsuite.txt
B-NP    B-NP
I-NP    I-NP
I-NP    I-NP
B-NP    B-NP
I-NP    I-NP
I-NP    I-NP
B-VP    B-VP
B-NP    B-NP
B-VP    B-VP
B-NP    B-NP
I-NP    I-NP
I-NP    I-NP
B-VP    B-VP
B-NP    B-NP
I-NP    I-NP
B-PP    B-PP
B-NP    B-NP
I-NP    I-NP
B-VP    B-VP
I-VP    I-VP
B-NP    B-NP
I-NP    I-NP
B-PP    B-PP
B-NP    B-NP
B-NP    B-NP
I-NP    I-NP
I-NP    I-NP
O       O

... (snip) ...
```

CRFsuite can also evaluate the CRF model with labeled test data with "-qt" options.

```
$ crfsuite tag -qt -m CoNLL2000.model test.crfsuite.txt
Performance by label (#match, #model, #ref) (precision, recall, F1):
    B-NP: (12000, 12358, 12407) (0.9710, 0.9672, 0.9691)
    B-PP: (4707, 4872, 4805) (0.9661, 0.9796, 0.9728)
```

```
    I-NP: (13984, 14484, 14359) (0.9655, 0.9739, 0.9697)
    B-VP: (4466, 4662, 4653) (0.9580, 0.9598, 0.9589)
    I-VP: (2549, 2698, 2643) (0.9448, 0.9644, 0.9545)
    B-SBAR: (448, 498, 534) (0.8996, 0.8390, 0.8682)
    O: (5939, 6113, 6174) (0.9715, 0.9619, 0.9667)
    B-ADJP: (322, 403, 438) (0.7990, 0.7352, 0.7658)
    B-ADVP: (711, 835, 866) (0.8515, 0.8210, 0.8360)
    I-ADVP: (54, 82, 89) (0.6585, 0.6067, 0.6316)
    I-ADJP: (110, 137, 167) (0.8029, 0.6587, 0.7237)
    I-SBAR: (2, 15, 4) (0.1333, 0.5000, 0.2105)
    I-PP: (34, 42, 48) (0.8095, 0.7083, 0.7556)
    B-PRT: (80, 102, 106) (0.7843, 0.7547, 0.7692)
    B-LST: (0, 0, 4) (0.0000, 0.0000, 0.0000)
    B-INTJ: (1, 1, 2) (1.0000, 0.5000, 0.6667)
    I-INTJ: (0, 0, 0) (******, ******, ******)
    B-CONJP: (5, 7, 9) (0.7143, 0.5556, 0.6250)
    I-CONJP: (10, 12, 13) (0.8333, 0.7692, 0.8000)
    I-PRT: (0, 0, 0) (******, ******, ******)
    B-UCP: (0, 0, 0) (******, ******, ******)
    I-UCP: (0, 0, 0) (******, ******, ******)
Macro-average precision, recall, F1: (0.639239, 0.602512, 0.611086)
Item accuracy: 45422 / 47321 (0.9599)
Instance accuracy: 1176 / 2011 (0.5848)
Elapsed time: 0.940000 [sec] (2140.4 [instance/sec])
```

Dumping the model file

When we improve the accuracy of a CRF model by tweaking the feature set, it may be useful to see the feature weights assigned by a trainer. You cannot simply read the model file since CRFsuite stores models in a binary format for the efficiency reason. Therefore, you need to use the dump command to read a model in plain text format.

```
$ crfsuite dump CoNLL2000.model
FILEHEADER = {
  magic: lCRF
  size: 28242501
  type: FOMC
  version: 100
  num_features: 0
  num_labels: 23
  num_attrs: 338547
  off_features: 0x30
  off_labels: 0x8B4EE4
  off_attrs: 0x8B5A0C
  off_labelrefs: 0x169C145
  off_attrrefs: 0x169C515
}

LABELS = {
      0: B-NP
      1: B-PP
      2: I-NP
      3: B-VP
      4: I-VP
      5: B-SBAR
      6: O
      7: B-ADJP
      8: B-ADVP
      9: I-ADVP
     10: I-ADJP
     11: I-SBAR
     12: I-PP
     13: B-PRT
     14: B-LST
     15: B-INTJ
     16: I-INTJ
     17: B-CONJP
     18: I-CONJP
     19: I-PRT
     20: B-UCP
     21: I-UCP
     22: I-LST
}

ATTRIBUTES = {
      0: U00=
      1: U01=
      2: U02=Confidence
```

```
      3: U03=in
      4: U04=the
      5: U05=/Confidence
      6: U06=Confidence/in
      7: U10=
... (snip) ...
}

TRANSITIONS = {
  (1) B-NP --> B-NP: 2.327985
  (1) B-NP --> B-PP: 4.391125
  (1) B-NP --> I-NP: 30.372649
  (1) B-NP --> B-VP: 7.725525
  (1) B-NP --> B-SBAR: 1.821388
  (1) B-NP --> O: 3.805715
  (1) B-NP --> B-ADJP: 4.801651
  (1) B-NP --> B-ADVP: 3.842473
... (snip) ...
}

TRANSITIONS_FROM_BOS = {
  (2) BOS --> B-NP: 17.875605
  (2) BOS --> B-PP: -0.318745
  (2) BOS --> I-NP: -4.387101
  (2) BOS --> B-VP: -0.383031
  (2) BOS --> I-VP: -1.163315
  (2) BOS --> B-SBAR: 1.368176
  (2) BOS --> O: 2.783132
... (snip) ...
}

TRANSITIONS_TO_EOS = {
  (3) B-NP --> EOS: 16.156051
  (3) B-PP --> EOS: -1.045312
  (3) I-NP --> EOS: -2.762051
  (3) B-VP --> EOS: -0.767247
  (3) I-VP --> EOS: -1.113502
  (3) B-SBAR --> EOS: -2.407145
  (3) O --> EOS: 4.131429
... (snip) ...
}

STATE_FEATURES = {
  (0) U00= --> B-NP: -2.622045
  (0) U00= --> B-PP: -1.562976
  (0) U00= --> I-NP: -2.555526
  (0) U00= --> B-VP: -1.329829
  (0) U00= --> I-VP: -1.152970
  (0) U00= --> B-SBAR: -2.590170
  (0) U00= --> O: -1.584688
  (0) U00= --> B-ADJP: -1.526879
... (snip) ...
}
```

Notes on writing attribute extractors

This tutorial used `chunking.py` bundled in the CRFsuite distribution to extract attributes from a data set. In practice, one may need to implement an attribute extractor suitable for a target task. One can write an attribute extractor in any manner in any programming language. However, if your data has a fixed length of fields (like the CoNLL 2000 data set) and if you are familiar with Python, it may be a good idea to modify the code of `chunking.py`. This section explains the structure of the script `chunking.py` for those who may modify it.

Here is the implementation of `chunking.py`:

```
#!/usr/bin/env python

"""
An attribute extractor for chunking.
Copyright 2010,2011 Naoaki Okazaki.
"""

# Separator of field values.
separator = ' '
```

```
# Field names of the input data.
fields = 'w pos y'

# Attribute templates.
templates = (
    (('w', -2), ),
    (('w', -1), ),
    (('w',  0), ),
    (('w',  1), ),
    (('w',  2), ),
    (('w', -1), ('w',  0)),
    (('w',  0), ('w',  1)),
    (('pos', -2), ),
    (('pos', -1), ),
    (('pos',  0), ),
    (('pos',  1), ),
    (('pos',  2), ),
    (('pos', -2), ('pos', -1)),
    (('pos', -1), ('pos',  0)),
    (('pos',  0), ('pos',  1)),
    (('pos',  1), ('pos',  2)),
    (('pos', -2), ('pos', -1), ('pos',  0)),
    (('pos', -1), ('pos',  0), ('pos',  1)),
    (('pos',  0), ('pos',  1), ('pos',  2)),
    )


import crfutils

def feature_extractor(X):
    # Apply feature templates to obtain features (in fact, attributes)
    crfutils.apply_templates(X, templates)
    if X:
        # Append BOS and EOS features manually
        X[0]['F'].append('__BOS__')     # BOS feature
        X[-1]['F'].append('__EOS__')    # EOS feature

if __name__ == '__main__':
    crfutils.main(feature_extractor, fields=fields, sep=separator)
```

The implementation is very simple because the common staffs (attribute generation from templates, data I/O, etc) are implemented in other modules, `crfutils.py` and `template.py`. The script `chunking.py` defines three important variables:

separator
: separator character(s) of an input data; this can be overwritten by a command-line argument ("-s" option); this example assumes a space character as a separator of fields.

fields
: field name(s) (ordered from left to right) of an input data, separated by a space character; this can be overwritten by a command-line argument ("-f" option); this example assumes that each line of an input data consists of fields named "w", "pos", and "y".

templates
: attribute (feature) templates written as a Python tuple/list object

It may be sufficient to modify these variables for your input data. Each element in the templates is a tuple/list of `(name, offset)` pairs, in which `name` presents a field name, and `offset` presents an offset to the current position.For example, the tuple,

```
    (('w', -2), ),
```

extract the value of 'w' field at two tokens ahead of the current position. Note that the comma after `('w', -2)` is necessary to define a tuple with one element `('w', -2)`. The tuple,

```
    (('w', -1), ('w',  0)),
```

defines an attribute that concatenates the value of 'w' field at the previous token and the value of 'w' field at the current position (i.e., the bigram starting at the previous token).

The function `feature_extractor` receives a sequence of items (`x` in this example) read from the input data, and generates necessary attributes. The argument `x` presents a list of items; each item is represented by a mapping (dictionary) object from field names to their values. In the CoNLL chunking task, `x[0]['w']` presents the word of the first item in the sequence, `x[0]['pos']` presents the part-of-speech tag of the last item in the sequence.

The mapping object of each item in `x` has a special key 'F' whose value (a list object) stores attributes generated for the item. Each element of an attribute list must be either a string or a tuple of `(name, value)` (an attribute with a weight). In the script `chunking.py`, feature templates are applied by using `crfutils.apply_templates` (which may fill attribute lists). The example also generates special attributes, "__BOS__" and "__EOS__" at the beginning and end of a sequence.

Feature extractors for other tasks

In addition to `chunking.py`, the `example` directory contains:

- `pos.py` for part-of-speech tagging. The structure of the code is similar to that of `chunking.py`; only the field descriptor and attribute templates are different.
- `ner.py` for named entity recognition. The script is more complicated because it extract various characteristics (e.g., character shape, prefixes and suffixes of tokens) from the input data.
- `template.py` for using feature templates compatible with CRF++. Features conditioned with attributes and label bigrams are not supported.

Wed, 03 Apr 2013 21:48:47 +0900