Dongqing Zhu

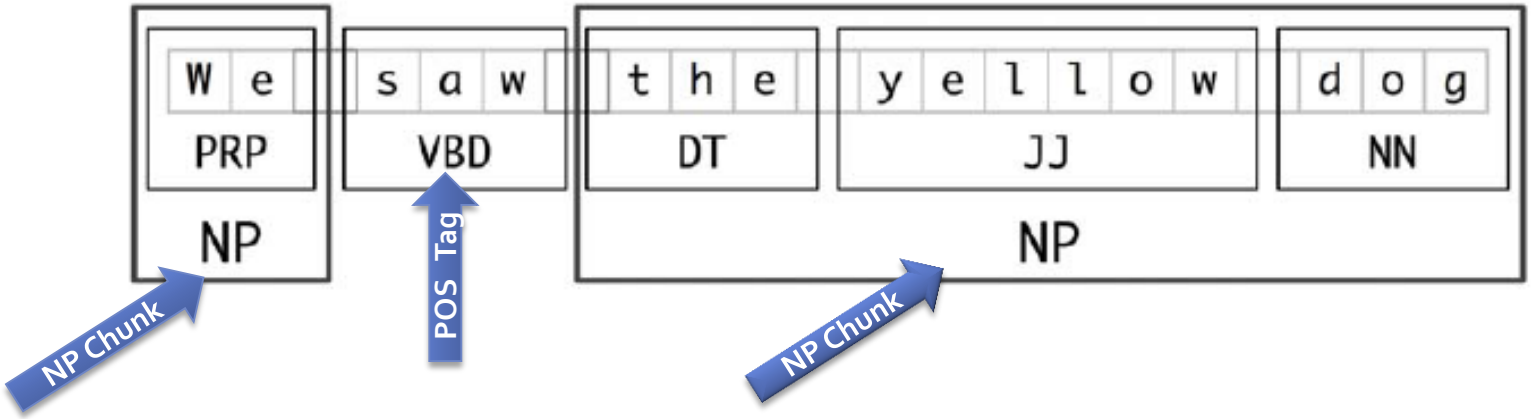
Text Chunking using NLTK

Outline

- What is chunking
- Why chunking
- How to do chunking
- An example: chunking a Wikipedia page
- Some suggestion
- Useful links

What is chunking?

- recovering phrases constructed by the part-of-speech tags
  - finding base noun phrases (focus of this tutorial)
  - finding verb groups, etc.



Why chunking?

- Information Extraction
  - keywords extraction
  - entity recognition
  - relation extraction
  - other applicable areas

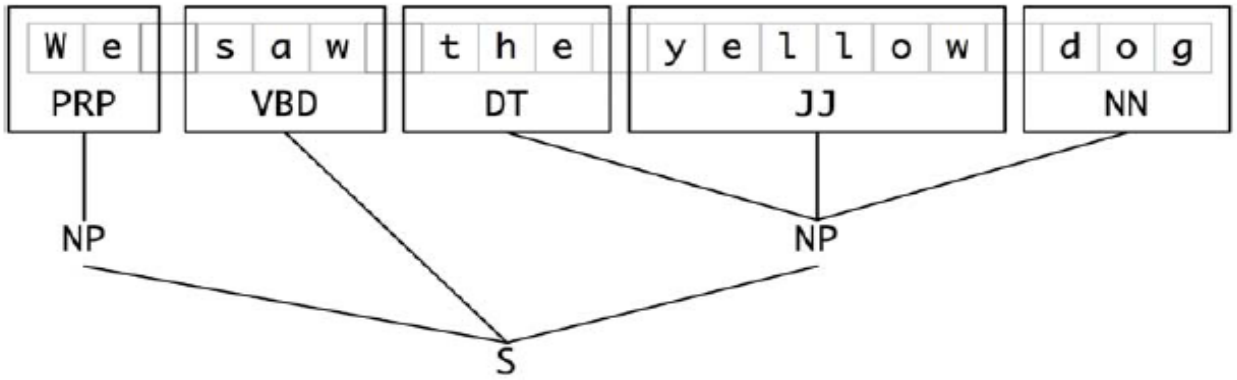
Noun phase chunking

- What is an NP chunk
  - base NP
  - an individual noun phrase that contains no other NP-chunks

- Examples:
  - 1. We saw the yellow dog. (2 NP chunks, underlined text)
  - 2. The market for system-management software for Digital's hardware is fragmented enough that a giant such as Computer Associates should do well there. (5 NP chunks)

Representation of chunk structures

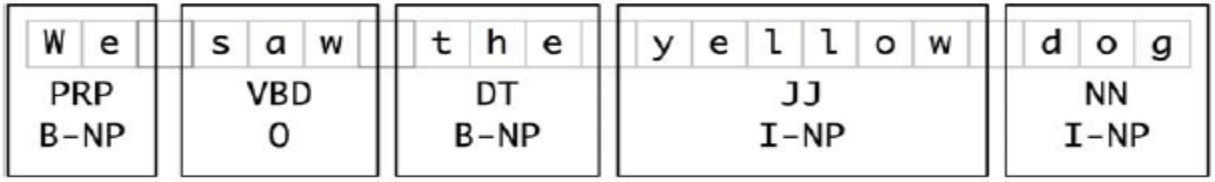
- Trees



TREE File format:

```
(S
   (NP the/DT
little/JJ yellow/JJ
dog/NN)
   barked/VBD
   at/IN
   (NP the/DT
cat/NN))
```

- Tags
  - IOB tags (I-inside, O-outside, B-begin)
  - label O for tokens outside a chunk



IOB File format:

he RPR B-NP
Accepted VBD B-VP
The DT B-NP
Position NN I-NP
…

Chunking with NLTK?

- Two approaches will be covered in this tutorial
  - approach one: chunking with regular expression
  - approach two: train a chunk parser

Chunking with regular expression

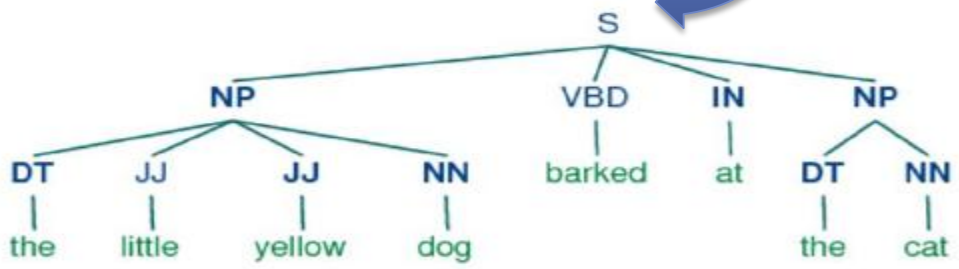
- Use POS tagging as the basis for extracting higher-level structure, i.e, phrases
- Key step: define tag patterns for deriving chunks
  - a tag pattern is a sequence of part-of-speech tags delimited using angle brackets
  - example: <DT>?<JJ>*<NN> defines a common NP pattern, i.e., an optional determiner (DT) followed by any number of adjectives (JJ) and then a noun (NN)

Chunking with regular expression

- Define tag patterns to find NP chunks

```
# Python code (remember to install and import nltk)
sentence = [("the", "DT"), ("little", "JJ"), ("yellow", "JJ"), ("dog", "NN"), ("barked",
"VBD"), ("at", "IN"), ("the", "DT"), ("cat", "NN")]   # a simple sentence with POS tags

pattern = "NP: {<DT>?<JJ>*<NN>}"  # define a tag pattern of an NP chunk

NPChunker = nltk.RegexpParser(pattern) # create a chunk parser

result = NPChunker . parse(sentence) # parse the example sentence
print result                         # or draw graphically using result.draw()
```

```
(S
  (NP the/DT little/JJ yellow/JJ dog/NN)
  barked/VBD
  at/IN
  (NP the/DT cat/NN))
```



Defining more tag patterns

- More tags patterns/rules for NP chunks
  - determiner/possessive, adjectives and noun: {<DT|PP\$>?<JJ>*<NN>}
  - sequences of proper nouns: {<NNP>+}
  - consecutive nouns: {<NN>+}

```
# define several tag patterns, used in the way as previous slide
patterns = """
            NP:     {<DT|PP\$>?<JJ>*<NN>}
                    {<NNP>+}
                    {<NN>+}
"""\

NPChunker = nltk.RegexpParser(patterns) # create a chunk parser
```

Defining more tag patterns

- Obtain tag patterns from corpus

Data from CoNLL 2000 Corpus (see next slide for how to load this corpus)
(NP UAL/NNP Corp./NNP stock/NN) # e.g. define {<NNP>+<NN>} to capture this pattern
(NP more/JJR borrowers/NNS)
(NP the/DT fact/NN)
(NP expected/VBN mortgage/NN servicing/NN fees/NNS)
(NP a/DT $/$ 7.6/CD million/CD reduction/NN)
(NP other/JJ matters/NNS)
(NP general/JJ and/CC administrative/JJ expenses/NNS)
(NP a/DT special/JJ charge/NN)
(NP the/DT increased/VBN reserve/NN)

Precision-recall tradeoff
Note that by adding more rules/tag patterns, you may achieve high recall but the precision will usually go down.

Training a chunk parser in NLTK

- Use CoNLL 2000 Corpus for training
  - CoNLL 2000 corpus contains 270k words of WSJ text
  - divided into training and testing portions
  - POS tags, chunk tags available in IOB format

```
# get training and testing data
from nltk.corpus import conll2000
test_sents = conll2000.chunked_sents('test.txt', chunk_types=['NP'])
train_sents = conll2000.chunked_sents('train.txt', chunk_types=['NP'])

# training the chunker, ChunkParser is a class defined in the next slide
NPChunker = ChunkParser(train_sents)
```

Training a chunk parser in NLTK

- Define ChunkerParser Class
  - to learn tag patterns for NP chunks

```
class ChunkParser(nltk.ChunkParserI):
   def __init__(self, train_sents):
      train_data = [[(t,c) for w,t,c in nltk.chunk.tree2conlltags(sent)]
                    for sent in train_sents]
      self.tagger = nltk.TrigramTagger(train_data)

   def parse(self, sentence):
      pos_tags = [pos for (word,pos) in sentence]
      tagged_pos_tags = self.tagger.tag(pos_tags)
      chunktags = [chunktag for (pos, chunktag) in tagged_pos_tags]
      conlltags = [(word, pos, chunktag) for ((word,pos),chunktag)
                   in zip(sentence, chunktags)]
      return nltk.chunk.conlltags2tree(conlltags)
```

Testing a chunk parser

- Evaluate the trained chunk parser

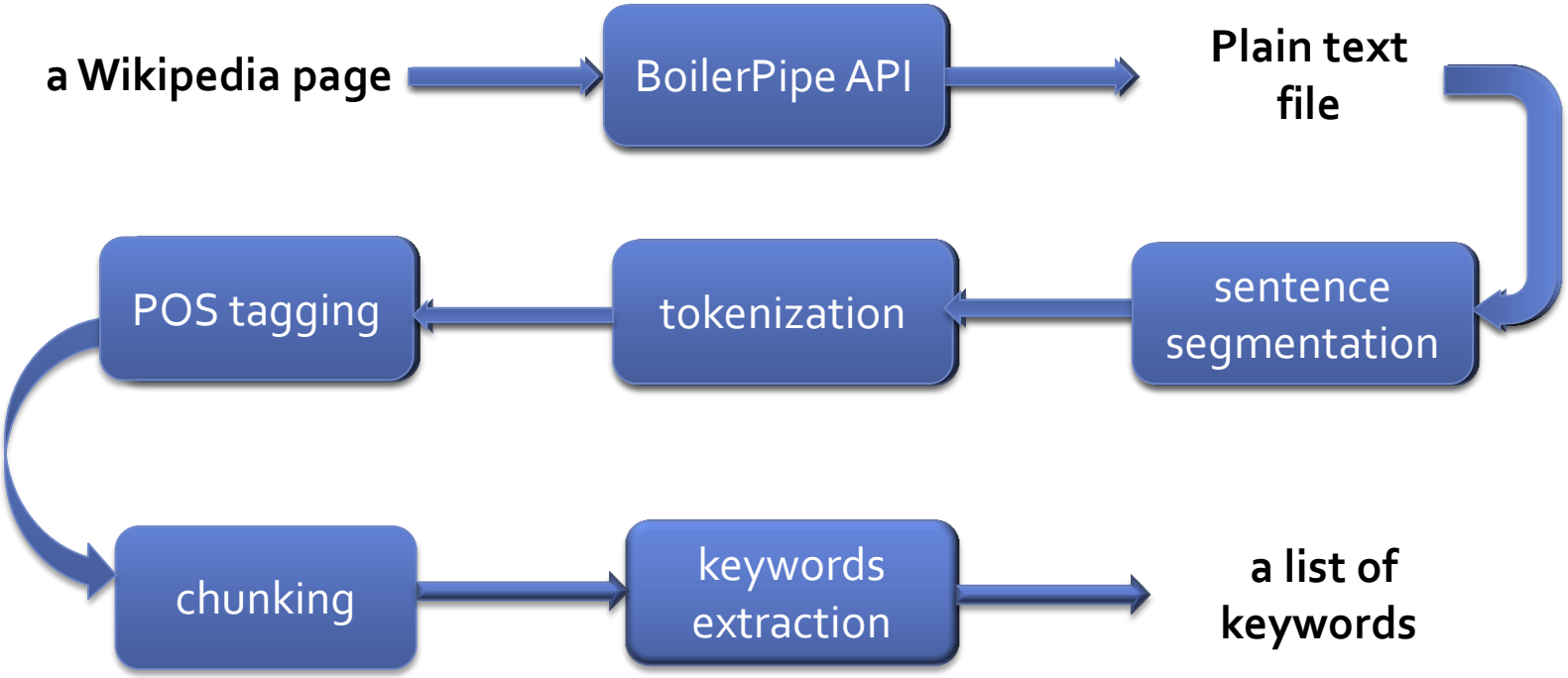
```
>>> print NPChunker.evaluate(test_sents)

#IOB Accuracy:  93.3%
ChunkParse score:

    Precision:     82.5%
    Recall:        86.8%
    F-Measure:     84.6%

# the chunker got decent results and is ready to use
# Note: IOB Accuracy corresponds to the IOB file format described in slide
'Representation of chunk structures'
```

Comparison of the two approaches

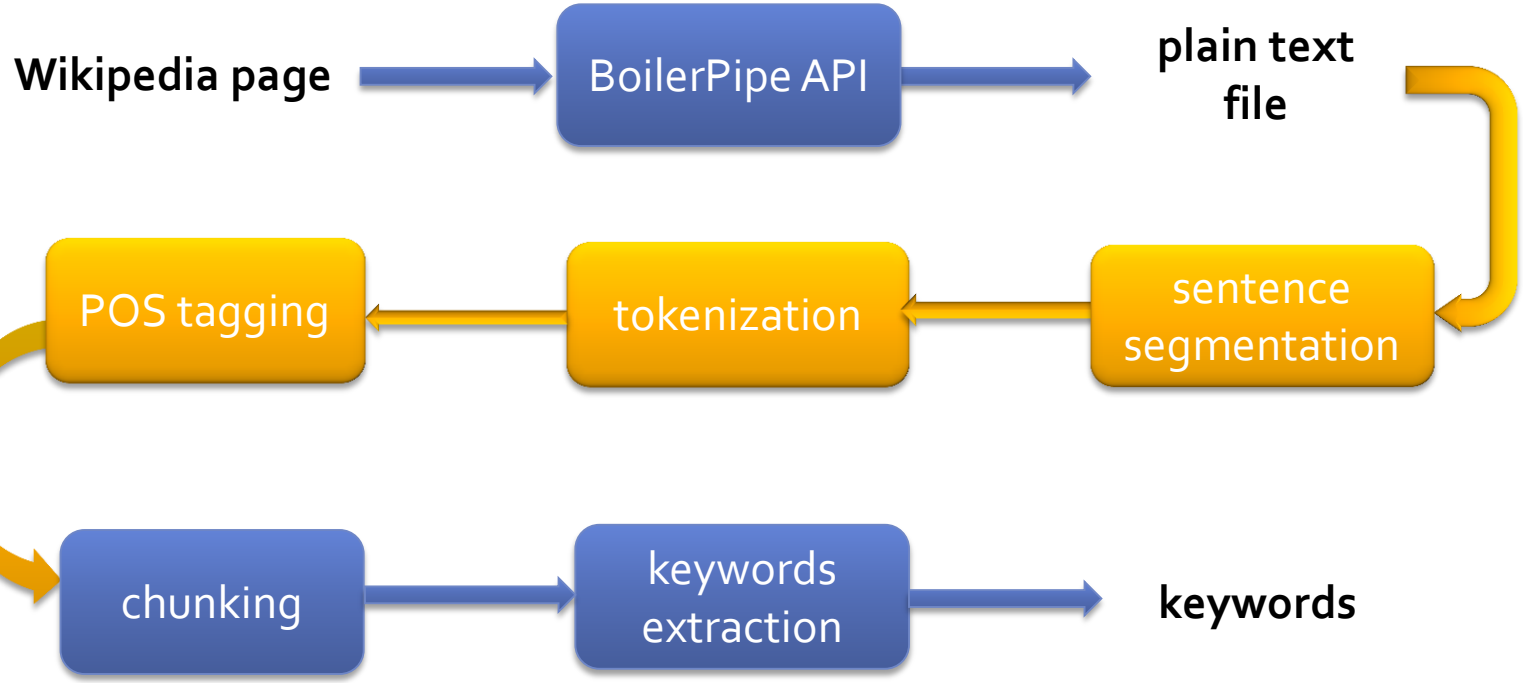
- Approach One
  - Pros: more control over what kind of tag patterns you want to match
  - Cons: difficult to come up with a set of rules to capture all base NP chunks and still keep a high precision
- Approach Two
  - Pros: high P/R for extracting all NP chunks
  - Cons: possibly need more post-processing to filter unwanted words

A text chunking example

- Chunking a Wikipedia page

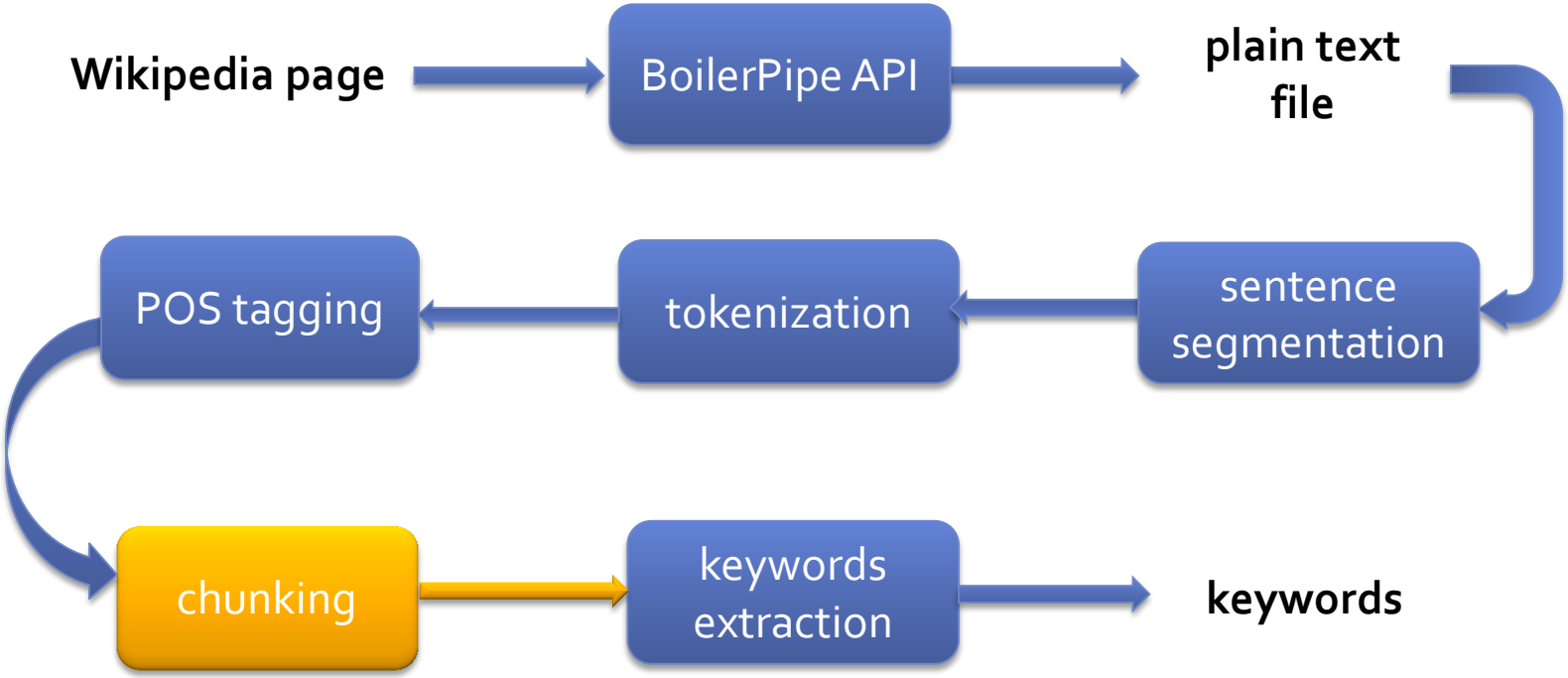


A text chunking example



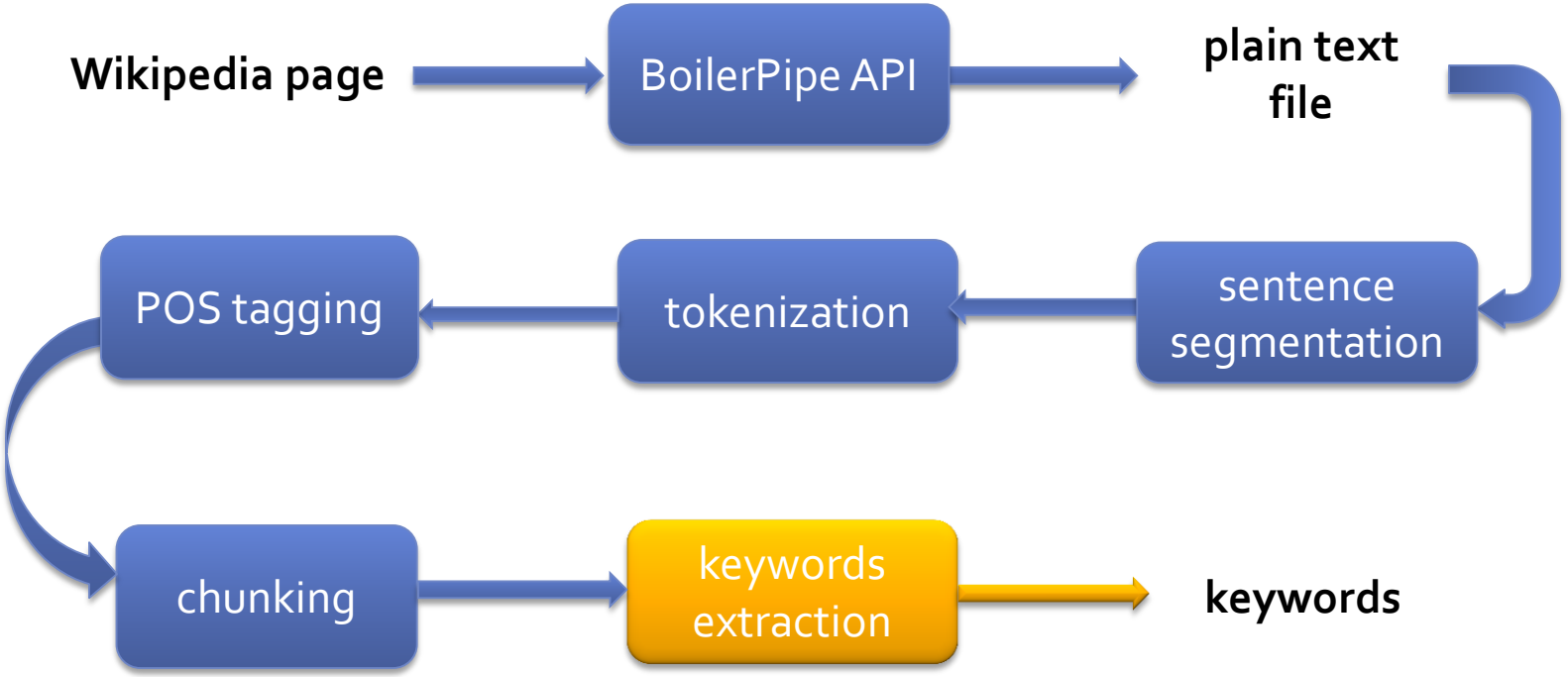

```
# Python code for segmentation, POS tagging and tokenization
import nltk
rawtext  =  open(plain_text_file).read()
sentences = nltk.sent_tokenize(rawtext) # NLTK default sentence segmenter
sentences = [nltk.word_tokenize(sent) for sent in sentences] # NLTK word tokenizer
sentences = [nltk.pos_tag(sent) for sent in sentences] # NLTK POS tagger
```

A text chunking example




```
for sent in sentences:
  # TO DO LIST (already covered in a few slides ahead):
  # 1. create a chunk parser by defining patterns of NP chunks or using the trained one
  # 2. parse every sentence
  # 3. store NP chunks
```

A text chunking example




```
# extract the keywords based on frequency
```

A little help

```
# a tree traversal function for extracting NP chunks in the parsed tree

def traverse(t):
   try:
      t.node
   except AttributeError:
      return
   else:
      if t.node == 'NP':  print t  # or do something else
      else:
         for child in t:
            traverse(child)
```

Results I got…

Tag patterns used for this example: 1. {<NN>+} 2. {<JJ>*<NN>} 3. {<NNP>+}

Top unigram NP chunks
(freq, term) pairs:

(118, 'iphone')
(55, 'apple')
(19, 'screen')
(18, 'software')
(16, 'update')
(16, 'phone')
(13, 'application')
(12, 'user')
(12, 'itunes')
(11, 'june')
(10, 'trademark')
…

Top bigram NP chunks

(6, 'app store')
(4, 'ipod touch')
(3, 'virtual keyboard')
(3, 'united kingdom')
(3, 'steve jobs')
(3, 'ocean telecom')
(3, 'mac os x')

…

Top NP chunks containing >2 terms
(3, 'mac os x')
(1, 'real-time geographic location')
(1, 'apple-approved cryptographic signature')
(1, 'free push-email service')
(1, 'computerized global information')
(1, 'apple ceo steve jobs')
(1, 'direct internal camera-to-e-mail picture')
…

Guess what the title of this Wikipedia page is?

Yes, it's about iPhone

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact Wikipedia

Toolbox

Print/export

Languages
العربية
Беларуская (тарашкевіца)
Brezhoneg
Български
Català
Česky
Cymraeg
Dansk
Deutsch
Eesti
Ελληνικά
Español
Esperanto
Euskara
فارسی
Français
Gaeilge
Galego
한국어
Հայերեն
हिन्दी

Article Discussion

iPhone

From Wikipedia, the free encyclopedia

This article is about the line of smartphones designed by Apple Inc. For other uses, see iPhone (disambiguation).

This article **may be too long to read and navigate comfortably.** Please consider splitt

The **iPhone** (pronounced /ˈaɪfoʊn/ *EYE-fohn*) is a line of Internet and multimedia-enabled smartphones designed and marketed by Apple Inc.

An iPhone can function as a video camera, a camera phone with text messaging and visual voicemail, a portable media player, and an Inter than a physical one. Third-party as well as Apple application software is available from the App Store, which launched in mid-2008 and now shows, films, and celebrities.

There are four generations of iPhone models, and they were accompanied by four major releases of iOS (formerly iPhone OS). The original capabilities and A-GPS location. The iPhone 3GS added a compass, faster processor, and higher resolution camera, including video. The iP 2011, when a CDMA version of the iPhone 4 launched for Verizon.

Contents [hide]

1 History and availability
2 Hardware
- 2.1 Screen and input
- 2.2 Audio and output
- 2.3 Battery
- 2.4 Camera
- 2.5 Storage and SIM
- 2.6 Liquid contact indicators
- 2.7 Included items

3 Model comparison
4 Software
- 4.1 Interface
- 4.2 Phone
- 4.3 Multimedia
- 4.4 Internet connectivity
- 4.5 Text input
- 4.6 E-mail and text messages
- 4.7 Third-party applications
- 4.8 Accessibility

5 Intellectual property
6 Restrictions
- 6.1 Activation
- 6.2 Unapproved third-party software and jailbreaking
- 6.3 SIM unlocking
  - 6.3.1 United States
  - 6.3.2 United Kingdom

Some suggestion for project 2

- If using approach one, define a few good tag patterns to only extract things you're interested
  - e.g. do not include determiners
  - define tag patterns for n-gram (n<4)
- If using approach two, do some post-processing
  - drop long NP phrases
- Try to form a tag cloud by just taking frequent bigram and trigram NP chunks, and use PMI or TF/IDF information to prune a bit, and then add some unigrams (remember you are only allowed to have no more than 15 tags)

Useful Links

- Chapters from book *Natural Language Processing with Python*
  - http://nltk.googlecode.com/svn/trunk/doc/book/ch05.html
  - http://nltk.googlecode.com/svn/trunk/doc/book/ch07.html
- LingPipe does chunking in a very similar way
  - http://alias-i.com/lingpipe/demos/tutorial/posTags/read-me.html