## Contents

1 Standing on the shoulders of giants 2

2 License 3

3 Contribute 4

4 Getting started 5
   4.1 What is NLP? ................................................................. 5
   4.2 What do I need to get started? ........................................ 5

5 User guide 7
   5.1 Installation ................................................................. 7
   5.2 Morphological analysis and POS tagging .......................... 10
   5.3 Data ............................................................................... 13
   5.4 Examples ....................................................................... 14
   5.5 Running tests ............................................................... 28
   5.6 References ..................................................................... 28

6 API 32
   6.1 konlpy Package ............................................................ 32

7 Indices and tables 41

Python Module Index 42
KoNLPy Documentation, Release 0.4.4

KoNLPy (pronounced “ko en el PIE”) is a Python package for natural language processing (NLP) of the Korean language. For installation directions, see here (page 7).

For users new to NLP, go to Getting started (page 5). For step-by-step instructions, follow the User guide (page 7). For specific descriptions of each module, go see the API (page 32) documents.

```python
>>> from konlpy.tag import Kkma
>>> from konlpy.utils import pprint

>>> kkma = Kkma()

>>> pprint(kkma.sentences(u'네, 안녕하세요. 반갑습니다.'))
[네, 안녕하세요..., 반갑습니다.]

>>> pprint(kkma.nouns(u'질문이나 건의사항은 깃합 이슈 트래커에 남겨주세요.'))
[ 질문, 건의, 건의사항, 사항, 깃합, 이슈, 트래커]

>>> pprint(kkma.pos(u'오류보고는 실행환경, 예외메세지와함께 설명을 최대한상세히!^^'))
[(오류, NNG),
(보고, NNG),
(는, JX),
(실행, NNG),
(환경, NNG),
(, SP),
(예외, NNG),
(메세지, NNG),
(와, JKM),
(함께, MAG),
(설명, NNG),
(을, JKO),
(최대한, NNG),
(상세히, MAG),
(!, SF),
(^^, EMO)]
```
Korean, the 13th most widely spoken language in the world (http://www.koreatimes.co.kr/www/news/nation/2014/05/116_157214.html), is a beautiful, yet complex language. Myriad Korean morpheme analyzer tools (page 28) were built by numerous researchers, to computationally extract meaningful features from the labyrinthine text.

KoNLPy is not just to create another, but to unify and build upon their shoulders, and see one step further. It is built particularly in the Python (programming) language (http://python.org), not only because of the language’s simplicity and elegance, but also the powerful string processing modules and applicability to various tasks - including crawling, Web programming, and data analysis.

The three main philosophies of this project are:

• Keep it simple.
• Make it easy. For humans.
• “Democracy on the web works.” (page 4)

Please report (https://github.com/konlp/konlpy/issues) when you think any have gone stale.
KoNLPy is Open Source Software, and is released under the license below:

- **GPL v3 or above** (http://gnu.org/licenses/gpl.html)

You are welcome to use the code under the terms of the license, however please acknowledge its use with a citation.


Here is a BibTeX entry:

```latex
@inproceedings{park2014konlpy,
  title={KoNLPy: Korean natural language processing in Python},
  author={Park, Eunjeong L. and Cho, Sungzoon},
  booktitle={Proceedings of the 26th Annual Conference on Human & Cognitive Language Technology},
  address={Chuncheon, Korea},
  month={October},
  year={2014}
}
```
KoNLPy isn’t perfect, but it will continuously evolve and you are invited to participate!

Found a bug? Have a good idea for improving KoNLPy? Visit the KoNLPy GitHub page (https://github.com/konlpy/konlpy) and suggest an idea (https://github.com/konlpy/konlpy/issues) or make a pull request (https://github.com/konlpy/konlpy/pulls).

You are also welcome to join our gitter (https://gitter.im/konlpy/konlpy) and the mailing list (https://groups.google.com/forum/#!forum/konlpy). Gitter is more focused on development discussions while the mailing list is a better place to ask questions, but nobody stops you from going the other way around.

Please note that asking questions through these channels is also a great contribution, because it gives the community feedback as well as ideas. Don’t hesitate to ask.
CHAPTER 4

Getting started

What is NLP?

NLP (Natural Language Processing) is a set of techniques for analyzing and extracting, and understanding meaningful information text.

We have various NLP applications in our lives. For example:

- Text summarization (ex: Summly (http://www.summly.com/index.html))
- Question answering (ex: Wolfram Alpha (http://www.wolframalpha.com/input/?i=what+is+the+meaning+of+life&lk=4&num=1))
- Machine translation (ex: Google Translate (http://translate.google.com))

And obviously information retrieval systems such as Web search engines. For a better understanding of NLP techniques, consider referring the so-called “bibles”:


KoNLPy will help you to actually carry out some fundamental NLP tasks with Korean text. In case you’re interested in handling English text, check out NLTK (http://nltk.org).

What do I need to get started?

You have some prerequisites to use KoNLPy.

1. Deep interest in natural languages and some familiarity with the Korean language
2. Understanding of basic Python programming\(^1\)

---

\(^1\) If you’re new to Python, this tutorial should get you through in minutes: http://learnxinyminutes.com/docs/python/. If you’re up to putting in some more time, try The Hitchhiker’s Guide (http://docs.python-guide.org/en/latest/) or Learn Python the hard way (http://learnyounetheway.org/book/).
3. A “good” text editor and terminal (or Python IDE)\(^2\)
5. pip (https://pypi.python.org/pypi/pip), the Python package manager

Got ‘em all? Then let’s go.

\(^2\) Many use Sublime Text 2 (http://www.sublimetext.com/) for Python programming. Some others use Vim and Terminal. But other than these, there are numerous great text editors (http://tutorialzine.com/2012/07/battle-of-the-tools-which-is-the-best-code-editor/) and Python IDEs (http://pedrokruger.net/choosing-best-python-ide/) out there, so take your pick!
User guide

Installation

**Note:** For troubleshooting information, see these pages: Linux (https://github.com/konlpy/konlpy/issues?q=label%3Alinux). Mac OS (https://github.com/konlpy/konlpy/issues?q=label%3A"mac+os"). Windows (https://github.com/konlpy/konlpy/issues?q=label%3Awindows). Please record a “New Issue” (https://github.com/konlpy/konlpy/issues/new) if you have an error that is not listed. You can also see testing logs here (https://docs.google.com/spreadsheets/d/1Ii_L9NF9gLbsJOQg5f-zfQTTyhhthmJWNC2kUDIsU/edit#gid=0).

### Ubuntu

1. Install dependencies

   $ sudo apt-get install g++ openjdk-7-jdk python-dev python3-dev # Install Java 1.7 or up
   $ pip install JPype1 # Python 2.x
   $ pip3 install JPype1-py3 # Python 3.x

2. Install KoNLPy

   $ pip install konlpy # Python 2.x
   $ pip3 install konlpy # Python 3.x

3. Install MeCab (optional)

   $ sudo apt-get install curl
   $ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/mecab.sh)

### CentOS

1. Install dependencies
1. Install KoNLPy

$ pip install konlpy   # Python 2.x
$ pip3 install konlpy  # Python 3.x

2. Install MeCab (optional)

$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/mecab.sh)

Mac OS

1. Install dependencies

$ pip install JPype1   # Python 2.x
$ pip3 install JPype1-py3  # Python 3.x

1. Install KoNLPy

$ pip install konlpy   # Python 2.x
$ pip3 install konlpy  # Python 3.x

2. Install MeCab (optional)

$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/master/scripts/mecab.sh)

Windows

1. Do you have Java 1.7+ installed? If not, download and install JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html).


3. Does your Python installation bit version match your Windows OS? If you're using a 64 bit Windows, you need a 64 bit

• How to check your Windows bit version

5.1. Installation
• How to check your Python bit version


    > pip install --upgrade pip
    > pip install JPyte1-0.5.7-cp27-none-win_amd64.whl

5. From the command prompt, install KoNLPy.

    > pip install konlpy

¹ win-amd64 for 64-bit Windows, win32 for 32-bit Windows.
**Warning:**

- KoNLPy’s `Mecab()` class is not supported on Windows machines.

---

**Morphological analysis and POS tagging**

*Morphological analysis* is the identification of the structure of morphemes and other linguistic units, such as root words, affixes, or parts of speech.

*POS (part-of-speech) tagging* is the process of marking up morphemes in a phrase, based on their definitions and contexts. For example:

가방에 들어가신다 -> 가방/NNG + 에/JKM + 들어가/VV + 시/EPH + 데/EFN

---

**POS tagging with KoNLPy**

In KoNLPy, there are several different options you can choose for POS tagging. All have the same input-output structure; the input is a phrase, and the output is a list of tagged morphemes.

For detailed usage instructions see the *tag Package* (page 32).

See also:

Korean POS tags comparison chart (https://docs.google.com/spreadsheets/d/1OGAjiUvalBuX-oZvZ_-9tE/YD2gQe7hTGsUpiB5XI8/edit#gid=0)

Compare POS tags between several Korean analytic projects. (In Korean)

---

**Comparison between POS tagging classes**

Now, we do time and performance analysis for executing the `pos` method for each of the classes in the *tag Package* (page 32). The experiments were carried out on a Intel i7 CPU with 4 cores, Python 2.7, and KoNLPy 0.4.1.

**Time analysis**

1. *Loading time:* Class loading time, including dictionary loads.

   - `Kkma` (page 33): 5.6988 secs
   - `Komoran` (page 34): 5.4866 secs
   - `Hannanum` (page 32): 0.6591 secs
   - `Twitter` (page 35): 1.4870 secs
   - `Mecab` (page 34): 0.0007 secs

2. *Execution time:* Time for executing the `pos` method for each class, with 100K characters.

   - `Kkma` (page 33): 35.7163 secs
   - `Komoran` (page 34): 25.6008 secs
   - `Hannanum` (page 32): 8.8251 secs
   - `Twitter` (page 35): 2.4714 secs

---

1 Please note that these are comparisons among KoNLPy classes, and not the original distributions.
If we test among a various number of characters, all classes’ execution times increase in an exponential manner.

### Performance analysis

The performance evaluation is replaced with result comparisons for several sample sentences.

1. “아버지가방에 들어가신다”

   We can check the spacing algorithm through this example. Desirably, an analyzer would parse this sentence to 아버지가 방에 들어가신다 (My father enters the room), rather than 아버지 + 가방에 + 들어가신다 (My father goes in the bag). *Hannanum* (page 32) and *Komoran* (page 34) are careful in spacing uncertain terms, and defaults the whole phrase to nouns. *Kkma* (page 33) is more confident, but gets undesirable results. For this result, *Mecab* (page 34) shows the best results.

<table>
<thead>
<tr>
<th>Hannanum</th>
<th>Kkma</th>
<th>Komoran</th>
<th>Mecab</th>
<th>Twitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>아버지가방에 들어가/N</td>
<td>아버지/NNG</td>
<td>아버지가방에 들어가신다/NNP</td>
<td>아버지/NNG</td>
<td>아버지/Noun</td>
</tr>
<tr>
<td>이/J</td>
<td>가방/NNG</td>
<td>가/JKS</td>
<td>가방/Noun</td>
<td></td>
</tr>
<tr>
<td>시/EPH</td>
<td>에/JKB</td>
<td>들어가/VV</td>
<td>들어가신/Verb</td>
<td></td>
</tr>
<tr>
<td>시/EPH</td>
<td>들어가/VV</td>
<td>다/Eomi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>늘/EFN</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. “나는 밤을 먹는다” vs “하늘을 나는 자동차”

   If we focus on “나는” in both sentences, we can see whether an analyzer considers the context of words. “나는” in the first sentence should be 나/N + 는/J, and in the second sentence 나 (-ㄹ 다)/V + 는/E. *Kkma* (page 33) properly understands the latter “나는” as a verb, wheras the rest observe it as nouns.
3. “아이폰 기다리다 지쳐 애플공홈에서 언락폰질러버렸다 6+ 128기가실버ㅋ”

How do each of the analyzers deal with slang, or terms that are not included in the dictionary?

Note: If you would like to run the experiments yourself, run this code (https://github.com/konlpy/konlpy/blob/master/docs/morph.py) from your local machine.
Data

Corpora

The following corpora are currently available:

1. **kolaw**: Korean law corpus.
   - constitution.txt

2. **kobill**: Korean National Assembly bill corpus. The file ID corresponds to the bill number.
   - 1809890.txt - 1809899.txt

For more detailed usage of the corpora, see the *corpus Package* (page 36).

```python
>>> from konlpy.corpus import kolaw
>>> c = kolaw.open('constitution.txt').read()
>>> print c[:10]
대한민국 헌법
유구한 역사와

>>> from konlpy.corpus import kobill
>>> d = kobill.open('1809890.txt').read()
>>> print d[:15]
지방공무원 법 일부개정 법률안
```

Dictionaries

Dictionaries are used for *Morphological analysis and POS tagging* (page 10), and are built with *Corpora* (page 30).

**Hannanum system dictionary**

A dictionary created with the KAIST corpus. (4.7MB)

Located at ./konlpy/java/data/kE/dic_system.txt. Part of this file is shown below:

```
... 
나라경제  ncn 
나라기획  nqq 
나라기획처장  ncn 
나라균  ncn 
나라통  ncn 
나라도독  ncn 
나라따르  pvg 
나라링프로덕션  ncn 
나라말  ncn 
나라망신  ncn 
나라박물관  ncn 
나라발견  ncpa 
나라별  ncn 
나라부동산  nqq 
나라사항  ncn 
나라실림  ncpa 
나라시  nqq 
나라시마  ncn 
... 
```

You can add your own terms, modify ./konlpy/java/data/kE/dic_user.txt.
**Kkma system dictionary**

A dictionary created with the Sejong corpus. (32MB)

It is included within the Kkma .jar file, so in order to see dictionary files, check out the KKMA’s mirror (https://github.com/e9t/kkma/tree/master/dic). Part of kcc.dic is shown below:

아니/IC
후우/IC
그래서/MAC
그리나/MAC
그러나까/MAC
그러면/MAC
그리므로/MAC
그런데/MAC
그리고/MAC
따라서/MAC
하지만/MAC
...

**Mecab system dictionary**

A CSV formatted dictionary created with the Sejong corpus. (346MB)

The compiled version is located at /usr/local/lib/mecab/dic/mecab-ko-dic (or the path you assigned during installation), and you can see the original files in the source code (https://bitbucket.org/eunjeon/mecab-ko-dic/src/ce04f82ab0083fb24e4e542e69d9e88a672c3325/seed/?at=master). Part of CoinedWord.csv is shown below:

가오플, 0, 0, NNG, *, T, 가오플, *, *, *, *
강득퍼, 0, 0, NNG, *, F, 강득퍼, *, *, *, *
강퇴, 0, 0, NNG, *, F, 강퇴, *, *, *, *
개드럼, 0, 0, NNG, *, T, 개드럼, *, *, *, *
견소, 0, 0, NNG, *, T, 견소, *, *, *
고필, 0, 0, NNG, *, T, 고필, *, *, *
광찰, 0, 0, NNG, *, T, 광찰, *, *, *
광탈, 0, 0, NNG, *, T, 광탈, *, *, *
광채, 0, 0, NNG, *, T, 광채, *, *
국유, 0, 0, NNG, *, T, 국유, *, *
귀요미, 0, 0, NNG, *, F, 귀요미, *, *, *
...

To add your own terms, see here (https://bitbucket.org/eunjeon/mecab-ko-dic/src/ce04f82ab0083fb24e4e542e69d9e88a672c3325/final/user-dic/?at=master).

**Note:** You can add new words either to the system dictionaries or user dictionaries. However, there is a slight difference in the two choices:

- **Adding to the system dictionary:** When dictionary updates are not frequent, when you do not want to drop the analysis speed.
- **Adding to the user dictionary:** When dictionary updates are frequent, when you do not have root access.

**Examples**

Below are a set of example tasks using KoNLPy.

*Exploring a document* (page 15)
Exploring a corpus (page 26)

Finding collocations (page 19)

Chunking (page 20)

Generating random text (page 21)

Drawing a word cloud (page 23)

Multithreading with KoNLPy (page 25)

Contents

Exploring a document

Exploring a document can consist of various components:

- Counts (characters, words, etc.)
- Checking Zipf’s laws: \( f r = k \)
5.4. Examples
5.4. Examples
from collections import Counter
from konlp.core import kolaw
from konlp.tag import Hannanum
from konlp.utils import concordance, pprint
from matplotlib import pyplot

def draw_zipf(count_list, filename, color='blue', marker='o'):
    sorted_list = sorted(count_list, reverse=True)
    pyplot.plot(sorted_list, color=color, marker=marker)
    pyplot.xscale('log')
    pyplot.yscale('log')
    pyplot.savefig(filename)

doc = kolaw.open('constitution.txt').read()
pos = Hannanum().pos(doc)
cnt = Counter(pos)

print('nchars :', len(doc))
print('ntokens :', len(doc.split()))
print('nmorphs :', len(set(pos)))

Top 20 frequent morphemes:
([(의, J), 398],
 (., S), 340),
 (하, X), 297),
 (예, J), 283),
 (త다, E), 242),
 (L, E), 226),
 (에, J), 218),
 (을, J), 211),
 (은, J), 184),
 (어, E), 177),
 (을, J), 148),
 (로, E), 135),
 (., S), 131),
 (하, P), 124),
 (을, J), 117),
 (법률, N), 115),
 (., S), 100),
 (는, E), 97),
 (있, P), 96),
 (ידי, X), 95)]

Locations of "대한민국" in the document:
Finding collocations

We can find collocations with the help of NLTK (http://nltk.org).

In order to find trigram collocations, replace BigramAssocMeasures with TrigramAssocMeasures, and BigramCollocationFinder with TrigramCollocationFinder.

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from konlp.tag import Kkma
from konlp.corpus import kolaw
from konlp.utils import pprint
from nltk import collocations

measures = collocations.BigramAssocMeasures()
doc = kolaw.open('constitution.txt').read()

print('Collocations among tagged words:)
tagged_words = Kkma().pos(doc)
```
finder = collocations.BigramCollocationFinder.from_words(tagged_words)
pprint(finder.nbest(measures.pmi, 10)) # top 5 n-grams with highest PMI

print('
Collocations among words:
')
words = [w for w, t in tagged_words]
ignored_words = ['안녕']
finder = collocations.BigramCollocationFinder.from_words(words)
finder.apply_word_filter(lambda w: len(w) < 2 or w in ignored_words)
finder.apply_freq_filter(3) # only bigrams that appear 3+ times
pprint(finder.nbest(measures.pmi, 10))

print('
Collocations among tags:
')
tags = [t for w, t in tagged_words]
finder = collocations.BigramCollocationFinder.from_words(tags)
pprint(finder.nbest(measures.pmi, 5))

• Console:

Collocations among tagged words:
[('가부', 'NNG'), ('동수', 'NNG'), ('강제', 'NNG'), ('노역', 'NNG'), ('경자', 'NNG'), ('유전', 'NNG'), ('고', 'ECS'), ('채취', 'NNG'), ('공무', 'NNG'), ('담당', 'NNNG'), ('공중', 'NNG'), ('도덕', 'NNG'), ('과반', 'NNG'), ('수가', 'NNG'), ('교전', 'NNG'), ('상태', 'NNNG'), ('그러', 'VV'), ('나', 'ECE'), ('기본적', 'NNNG'), ('인권', 'NNNG')]

Collocations among words:
[('현행', '범인'), ('형의', '선고'), ('내부', '규율'), ('정치적', '공립성'), ('누구', '들지'), ('사회', '연도'), ('지제', '없이'), ('평화적', '통일'), ('형사', '피고인'), ('지방', '자치')]

Collocations among tags:
[('XR', 'XSA'), ('JKC', 'VCN'), ('VCN', 'ECD'), ('ECD', 'VX'), ('ECD', 'VXV')]

Chunking

After tagging a sentence with part of speech (page 10), we can segment it into several higher level multitoken sequences, or “chunks”.


```python
#! /usr/bin/python2.7
# -*- coding: utf-8 -*-
```

5.4. Examples
According to the chunk grammar defined above, we have three rules to extract phrases from our sentence. First, we have a rule to extract noun phrases (NP), where our chunker finds a serial of nouns, followed with an optional Suffix. (Note that these rules can be modified for your purpose, and that they should differ for each morphological analyzer.) Then we have two more rules, each defining verb phrases (VP) and adjective phrases (AP).

The result is a tree, which we can print on the console, or display graphically as follows.

- Console:

```
# Print whole tree
(S
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
   의/Josa
  (NP 초등학교/Noun 취학/Noun 전/Noun 자녀/Noun)
  를/Josa
  (NP 양육/Noun)
  (VP 하기/Verb 위해서/Verb)
   는/Eomi)
# Print noun phrases only
만 6 세 이하
  (NP 만/Noun 6/Number 세/Noun 이하/Noun)
  초등학교/Noun 취학/Noun 전/Noun 자녀/Noun
  양육/Noun
```

- chunking.png

### Generating random text

Say you want to generate random text in Korean. How would you do it?
The easiest way would probably be to have your cat walk across your keyboard which would result in something like this:\footnote{This story would actually feature a monkey instead of a cat. Namely by the \href{http://en.wikipedia.org/wiki/Infinite_monkey_theorem}{Infinite monkey theorem}.}

\begin{verbatim}
|키_PAYLOAD | 음아 | 쩐 | コ | 문화 | 투터 | 하 | \n| ㅕ | 키 |ㅔ |츄 |ㅑ |응 |아 |ㅝ |ㄴ |ㅇ |쿄 |ㅛ |ㅍ |ㄹ |ㅎ |튜 |허 |
\end{verbatim}

However a sequence of random letters like this does not make any sense. Normally, Korean text is formed with a sequence of words, formed by a sequence of syllables, which are each formed with two to three types of the keyboard, each uniquely called choseong, jungseong, jongseong\footnote{Please refer to the \href{http://www.unicode.org/charts/}{Hangul Jamo in Unicode character code charts}.}. (though in casual chatting, just one type of the keyboard is common as well, ex: "ㅋㅋㅋ") So now we educate our cat to type syllables:

\begin{verbatim}
로 | helicoptor | 불 | 도 | 약 | 니 | 앙 |
\end{verbatim}

Then we notice that in Korean, the syllable ‘이’ is more frequently used than ‘앙’ and definitely much more than ‘뻘’ or ‘곱’. If our cat knew that, he would have typed something like this:

\begin{verbatim}
다 | 이 |는 |가 | 고 |
\end{verbatim}

But then, this still doesn’t make any sense because the syllables don’t form words. Rather than generating each syllable independently, we can generate a syllable base on its precedent so that after ‘다’ follows ‘타’, and after ‘그’ we get ‘라’ and ‘고’. In mathematical terms, this process is better known as a Markov chain\footnote{Please refer to the \href{http://en.wikipedia.org/wiki/Markov_chain}{Markov chain}.}

\begin{verbatim}
국의의 | 하 | 되 |고 |
\end{verbatim}

Our “sentence” above was generated with “bigrams”, or “2-grams”. If we wish to make more sense out of it, we could try “3-grams” (better known as trigrams) or “4-grams”. Or, we could extend the same idea to longer sequences of letters, such as morphemes. Let’s try this with actual code.

\begin{verbatim}
#! /usr/bin/python3
# -*- coding: utf-8 -*-
import bisect
import itertools
import random
import nltk
from konlpy.corpus import kolaw
from konlpy.tag import Mecab # MeCab tends to reserve the original form of morphemes

def generate_sentence(cfdist, word, num=15):
    sentence = []

    # Generate words until we meet a period
    while word!='.):
        sentence.append(word)
        word = choices[bisect.bisect(cumdist, x)]

    return ' '.join(sentence)

sentence = generate_sentence(cfdist, '안녕하세요')
print(sentence)
\end{verbatim}

Warning: The code below works with Python3, and not with Python2! You can run the code by typing python3 generate.py on your terminal.
def calc_cfd(doc):
    # Calculate conditional frequency distribution of bigrams
    words = [w for w, t in Mecab().pos(doc)]
    bigrams = nltk.bigrams(words)
    return nltk.ConditionalFreqDist(bigrams)

if __name__ == '__main__':
    nsents = 5 # Number of sentences
    initstr = u'국가' # Try replacing with u'국가', u'대통령', etc

    doc = kolaw.open('constitution.txt').read()
    cfd = calc_cfd(doc)
    for i in range(nsents):
        print('%d. %s' % (i, generate_sentence(cfd, initstr)))

    • Console:

    0. 국민은 법률로 인한 배상은 특별한 영장은 청구할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특별자 가 청구 할 수 있 어서 최고 특 볼
1. 국민 투표의 범죄에 의하여 발생한 자가 아닌 경우는 1988 년으로 대통령 이 의결한다
2. 국민 경제 자문 기구를 터파하기 위하여 열린 기회로 정한다
3. 국민은 이 정하는 헌법 시행 당시의 심사할 수 있다
4. 국민의 기본 질문을 진다

Well, that’s a lot better than the random string typed by our cat! The sentences look a bit ugly because there are whitespaces between all morphemes, whereas in actual Korean text, they would be stuck together. Also note that this text generation model was built from a single document. If you were to build a model with a much larger corpus, you wouldn’t even have to do morpheme analysis because you would have enough data for any potential initstr. Other than that, there are much more ways to improve this model! Feel free to experiment.

For more on generating text, you can refer to Jon Bentley’s Programming Pearls (Section 15.3) (http://www.cs.bell-labs.com/cm/cs/pearls/sec153.html).

Furthermore, if you use language models (http://en.wikipedia.org/wiki/Language_model), you can evaluate your random texts and figure out whether they actually make sense in a statistical point of view.

Drawing a word cloud

Below shows a code example that crawls a National Assembly bill from the web, extract nouns and draws a word cloud - from head to tail in Python.

You can change the bill number (i.e., bill_num), and see how the word clouds differ per bill. (ex: ‘1904882’, ‘1904883’, ‘ZZ19098’, etc)

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from collections import Counter
import urllib
import random
import webbrowser
from konlpy.tag import Hannanum
from lxml import html
import pytagcloud # requires Korean font support
```
```python
import sys

if sys.version_info[0] >= 3:
    urlopen = urllib.request.urlopen
else:
    urlopen = urllib.urlopen

r = lambda: random.randint(0, 255)
color = lambda: (r(), r(), r())

def get_bill_text(billnum):
    url = 'http://pokr.kr/bill/%s/text' % billnum
    response = urlopen(url).read().decode('utf-8')
    page = html.fromstring(response)
    text = page.xpath(".//div[@id='bill-sections']/pre/text()")[0]
    return text

def get_tags(text, ntags=50, multiplier=10):
    h = Hannanum()
    nouns = h.nouns(text)
    count = Counter(nouns)
    return [{'color': color(), 'tag': n, 'size': c*multiplier}
            for n, c in count.most_common(ntags)]

def draw_cloud(tags, filename, fontname='Noto Sans CJK', size=(800, 600)):
    pytagcloud.create_tag_image(tags, filename, fontname=fontname, size=size)
    webbrowser.open(filename)

bill_num = '1904882'
text = get_bill_text(bill_num)
tags = get_tags(text)
print(tags)
draw_cloud(tags, 'wordcloud.png')
```

**Note:** The PyTagCloud (https://pypi.python.org/pypi/pytagcloud) installed in PyPI may not be sufficient for drawing wordclouds in Korean. You may add eligible fonts - that support the Korean language - manually, or install the Korean supported version here (https://github.com/e9t/PyTagCloud).

---

5.4. Examples
Multithreading with KoNLPy

Sometimes it gets boring to wait for tagging jobs to end. How about using some concurrency tricks? Python supports multithreading and multiprocessing out-of-the-box, and you can use them with KoNLPy as well. Here’s an example using multithreading.

```python
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from konlp.tag import Kkma
from konlp.corpus import kolaw
from threading import Thread
import jpype

def do_concurrent_tagging(start, end, lines, result):
    jpype.attachThreadToJVM()
    l = [k.pos(lines[i]) for i in range(start, end)]
    result.append(l)
return

if __name__ == "__main__":
    import time

    k = Kkma()
    lines = kolaw.open('constitution.txt').read().splitlines()
nlines = len(lines)
print(nlines)

    print('Batch tagging: ')
s = time.clock()
result = []
l = [k.pos(line) for line in lines]
result.append(l)
t = time.clock()
print(t - s)

    print('Concurrent tagging: ')
```

5.4. Examples
```python
result = []
t1 = Thread(target=do_concurrent_tagging, args=(0, int(nlines/2), lines, result))
t2 = Thread(target=do_concurrent_tagging, args=(int(nlines/2), nlines, lines, result))
t1.start(); t2.start()
t1.join(); t2.join()
m = sum(result, []); # Merge results
print(time.clock() - t)
```

• Console:

<table>
<thead>
<tr>
<th>Number of lines in document:</th>
<th>356</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch tagging:</td>
<td>37.758173</td>
</tr>
<tr>
<td>Concurrent tagging:</td>
<td>8.037602</td>
</tr>
</tbody>
</table>

Check out how much faster it gets!

**Note:**

- Some useful references on concurrency with Python:

### Exploring a corpus

A corpus is a set of documents.

Below is a way of exploring unique tokens of a corpus, namely the Heap’s Law (http://en.wikipedia.org/wiki/Heaps%27_law).

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-

from konlpy.corpus import kobill
from konlpy.tag import Twitter; t = Twitter()
from matplotlib import pyplot as plt

pos = lambda x: [u'/' .join(p) for p in t.pos(x)]
docs = [kobill.open(i).read() for i in kobill.fileids()]

# get global unique token counts
global_unique = []
global_unique_cnt = []
for doc in docs:
    tokens = pos(doc)
    unique = set(tokens)
    global_unique += list(unique)
    global_unique_cnt.append(len(unique))
print(len(unique), len(global_unique))

# draw heap
```

5.4. Examples
But why is our image not log-function shaped, as generally known? That is because the corpus we used is very small, and contains only 10 documents. To observe the Heap’s law’s log-function formatted curve, try experimenting with a larger corpus. Below is an image drawn from 1,000 Korean news articles. Of course, the curve will become smoother with a much larger corpus.
Running tests

KoNLPy has tests to evaluate its quality. To perform a test, use the code below.

```bash
$ pip install pytest
$ cd konlpy
$ python -m pytest test/*  # for Python 2.x
$ python3 -m pytest test/*  # for Python 3.x
```

**Note:** To see testing logs on KoNLPy, see here (https://docs.google.com/spreadsheets/d/1Ii_L9NF9gSLbsJOGqsfrTtyhhtmJWNC2kgUDIsU/edit?usp=sharing).

**Note:** To see known bugs/issues, see here (https://github.com/konlpy/konlpy/labels/bug).

References

**Note:** Please modify this document (https://github.com/konlpy/konlpy/blob/master/docs/references.rst) if anything is erroneous or not included. Last updated at Sep 25, 2017.

Korean morpheme analyzer tools

When you’re analyzing Korean text, the most basic task you need to perform is morphological analysis. There are several libraries in various programming languages to achieve this:
C/C++

- MeCab-ko (https://bitbucket.org/eunjeon/mecab-ko/) (2013) - By Yong-woon Lee and Youngho Yoo GPL LGPL BSD

  - slides (http://www.slideserve.com/mills/u-tagger-2013)


  - Created at 1995, released at 2002.1

Java/Scala

  - Apache v2


  - code (http://sourceforge.net/projects/lucenekorean)

- HanNanum (http://semanticweb.kaist.ac.kr/home/index.php/HanNanum) (1999) - By Key-Sun Choi* et al. (KAIST) GPL v3
  - code (http://kldp.net/projects/hannanum/src), docs (http://semanticweb.kaist.ac.kr/research/hannanum/j/javadoc/)

---

1 https://wiki.kldp.org/wiki.php/KTS
Python

- **KoNLPy** ([http://konlp.org](http://konlp.org)) (2014) GPL v3+
  - By Lucy Park (Seoul National University)
  - Wrapper for Hannanum, KKMA, KOMORAN, twitter-korean-text, MeCab-ko
  - Tools for Hangul/Korean manipulation

- **UMorpheme** ([https://pypi.python.org/pypi/UMorpheme](https://pypi.python.org/pypi/UMorpheme)) (2014) MIT
  - By Kyunghoon Kim (UNIST)
  - Wrapper for MeCab-ko for online usage

R

- **KoNLP** ([https://github.com/haven-jeon/KoNLP](https://github.com/haven-jeon/KoNLP)) (2011) GPL v3
  - By Heewon Jeon
  - Wrapper for Hannaum

Others

- **K-LIWC** ([http://k-liwc.ajou.ac.kr/](http://k-liwc.ajou.ac.kr/)) (아주대)
- **KRISTAL-IRMS** ([http://www.kristalinfo.com/](http://www.kristalinfo.com/)) (KISTI)
  - Development history ([http://spasis.egloos.com/9507](http://spasis.egloos.com/9507))
- **Korean XTAG** ([http://www.cis.upenn.edu/~xtag/koreantag/](http://www.cis.upenn.edu/~xtag/koreantag/)) (UPenn)
- **HAM** ([http://nlp.kookmin.ac.kr/HAM/kor/ham-intr.html](http://nlp.kookmin.ac.kr/HAM/kor/ham-intr.html)) (국민대)
- **POSTAG/K** ([http://nlp.postech.ac.kr/~project/DownLoad/k_api.html](http://nlp.postech.ac.kr/~project/DownLoad/k_api.html)) (POSTECH)

Corpora

- **Korea University Korean Corpus, 1995.**
  - 10M tokens of Korean of 1970-90s
  - 120,000 test documents (237MB)
  - 50 TREC-type questions for QA (48KB)
  - 40,075 test documents for text categorization (88MB)
  - 42M tokens of Korean since the 1960s
- **BoRA 언어자원은행** ([http://semanticweb.kaist.ac.kr/org/bora/](http://semanticweb.kaist.ac.kr/org/bora/)), KAIST
Other NLP tools

- **Hangulize** (http://www.hangulize.org/) - By Heungsub Lee Python
  - Hangul transcription tool to 38+ languages
- **Hanja** (https://github.com/suminb/hanja) - By Sumin Byeon Python
  - Hanja to hangul transcriptor
- **Jamo** (http://github.com/JDong820/python-jamo) - By Joshua Dong Python
  - Hangul syllable decomposition and synthesis
- **KoreanParser** (http://semanticweb.kaist.ac.kr/home/index.php/KoreanParser) - By DongHyun Choi, Jungyeul Park, Key-Sun Choi (KAIST) Java
  - Language parser
- **Korean** (http://pythonhosted.org/korean) - By Heungsub Lee Python
  - Package for attaching particles (josa) in sentences
- **Speller** (http://speller.cs.pusan.ac.kr/) (부산대)
konlpy Package

Subpackages

tag Package

Note: Initial runs of each class method may require some time to load dictionaries (< 1 min). Second runs should be faster.

Hannanum Class

class konlpy.tag._hannanum.Hannanum (jvmpath=None)
JHannanum is a morphological analyzer and POS tagger written in Java, and developed by the Semantic Web Research Center (SWRC) (http://semanticweb.kaist.ac.kr/) at KAIST since 1999.

```python
>>> from konlpy.tag import Hannanum
>>> hannanum = Hannanum()
>>> print(hannanum.analyze(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다. '))
[['('롯데마트', 'ncn'), ('의', 'jcm')], [('롯데마트의', 'nncn')], [('롯데마트', 'nq'), ('의', 'jcm')], [('롯데마트의', 'nq')]], [('혈액', 'ncn')], [('기분', 'nncn')], [('기분', 'ncn')]
```

```python
>>> print(hannanum.morphs(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다. '))
['로', '테', '마', '트', '의', '의', '와', '요', '양', '념', '치', '킨', '이', '되', '고', '있', '다', '(', '.)', '(', '.)']
```
Parameters 
jvmpath – The path of the JVM passed to init_jvm() (page 38).

analyze (phrase)
Phrase analyzer.
This analyzer returns various morphological candidates for each token. It consists of two parts: 1) Dictionary search (chart), 2) Unclassified term segmentation.

morphs (phrase)
Parse phrase to morphemes.
nouns (phrase)
Noun extractor.
pos (phrase, ntags=9, flatten=True)
POS tagger.
This tagger is HMM based, and calculates the probability of tags.
Parameters
• ntags – The number of tags. It can be either 9 or 22.
• flatten – If False, preserves eojoles.

Kkma Class
class konlpy.tag._kkma.Kkma (jvmpath=None)
Kkma is a morphological analyzer and natural language processing system written in Java, developed by the Intelligent Data Systems (IDS) Laboratory at SNU (http://snu.ac.kr).

Warning: There are reports that Kkma() is weak for long strings with no spaces between words. See issue #73 (https://github.com/konlpy/konlpy/issues/73) for details.

Parameters 
jvmpath – The path of the JVM passed to init_jvm() (page 38).
morphs (phrase)
    Parse phrase to morphemes.
	nouns (phrase)
    Noun extractor.

pos (phrase, flatten=True)
    POS tagger.

Parameters
    flatten – If False, preserves eojeols.

definitions (phrase)
    Sentence detection.

Komoran Class

class konlpy.tag._komoran.Komoran (jvmpath=None, dicpath=None)

    KOMORAN is a relatively new open source Korean morphological analyzer written in Java, developed by
    Shineware (http://shineware.co.kr), since 2013.

    >>> from konlpy.tag import Komoran
    >>> komoran = Komoran()
    >>> print(komoran.morphs(u'우왕 코모란도 오픈소스가 되었어요'))
    ['우왕', '꼬', '모란', '도', '오픈소스', '가', '되', '있', '아요']
    >>> print(komoran.nouns(u'오픈소스에 관심 많은 맛진 개발자님들!'))
    ['오픈소스', '관심', '개발자']
    >>> print(komoran.pos(u'원칙이나 기체 설계와 엔진·레더·항법장비 등'))
    [('원칙', 'NNG'), ('이나', 'JC'), ('기체', 'NNP'), ('설계', 'NNP'), ('엔진', 'NNG'), ('', 'SP'), ('렌더', 'NNP'), ('', 'SP'), ('항법', 'NNP'), ('', 'SP'), ('장비', 'NNP'), ('등', 'NNB')]

Parameters
    • jvmpath – The path of the JVM passed to init_jvm() (page 38).
    • dicpath – The path of dictionary files. The KOMORAN system dictionary is loaded
default.

morphs (phrase)
    Parse phrase to morphemes.
	nouns (phrase)
    Noun extractor.

pos (phrase, flatten=True)
    POS tagger.

Parameters
    flatten – If False, preserves eojeols.

Mecab Class

Warning: Mecab() is not supported on Windows 7.

class konlpy.tag._mecab.Mecab (dicpath=/usr/local/lib/mecab/dic/mecab-ko-dic)
    Wrapper for MeCab-ko morphological analyzer.
MeCab (https://code.google.com/p/mecab/), originally a Japanese morphological analyzer and POS tagger developed by the Graduate School of Informatics in Kyoto University, was modified to MeCab-ko by the Eunjeon Project (http://eunjeon.blogspot.kr) to adapt to the Korean language.

In order to use MeCab-ko within KoNLPy, follow the directions in optional-installations.

```python
>>> # MeCab installation needed
>>> from konlpy.tag import Mecab
>>> mecab = Mecab()
>>> print(mecab.morphs(u'영등포구청역에 있는 맛집 중 알려주세요. '))
['영등포구', '청역', '에', '있는', '맛집', '중', '알려', '주세요', '음']
>>> print(mecab.nouns(u'우리나라에는 우물 치료를 잘하는 정형외과가 없는가!'))
['우리', '나라', '우물', '치료', '정형외과']
>>> print(mecab.pos(u'자연주의 쇼핑몰은 어떤 곳인가?'))
[(', 'Eomi'), (', ', 'KoreanParticle')]
```

Parameters `dicpath` - The path of the MeCab-ko dictionary.

- `morphs (phrase)`
  Parse phrase to morphemes.

- `nouns (phrase)`
  Noun extractor.

- `pos (phrase, flatten=True)`
  POS tagger.

  Parameters `flatten` - If False, preserves eojeols.

**Twitter Class**

class konlpy.tag._twitter.Twitter(jvmpath=None)


Twitter Korean Text is an open source Korean tokenizer written in Scala, developed by Will Hohyon Ryu.

```python
>>> from konlpy.tag import Twitter
>>> twitter = Twitter()
>>> print(twitter.morphs(u'단독임차보다 복수임차의 경우'))
['단독', '임차', '보다', '복수', '임차', '의', '경우', '가']
>>> print(twitter.nouns(u'유일하게 항공기 체계 종합개발 경험을 갖고 있는 KAI는'))
['유일하', '항공기', '체계', '종합', '개발', '경험']
>>> print(twitter.pos(u'널리로운 분석과 신뢰감 있는 진행으로'))
['분석', '분석과 신뢰감', '신뢰감', '분석과 신뢰감 있는 진행', '신뢰감 있는 진행', '진행', '신뢰감']
>>> print(twitter.pos(u'이것도 되나요ㅋㅋ'))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나요', 'Noun'), ('ㅋㅋ', 'KoreanParticle')]
>>> print(twitter.pos(u'이것도 되나요ㅋㅋ', norm=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되', 'Verb'), ('나요', 'SV'), ('ㅋㅋ', 'KoreanParticle')]
>>> print(twitter.pos(u'이것도 되나요ㅋㅋ', norm=True, stem=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되다', 'Verb'), ('ㅋㅋ', 'KoreanParticle')]
```

Parameters `jvmpath` - The path of the JVM passed to `init_jvm()` (page 38).

- `morphs (phrase, norm=False, stem=False)`
  Parse phrase to morphemes.
nouns (phrase)
Noun extractor.

phrases (phrase)
Phrase extractor.

pos (phrase, norm=False, stem=False)
POS tagger. In contrast to other classes in this subpackage, this POS tagger doesn’t have a flatten option, but has norm and stem options. Check the parameter list below.

Parameters

• **norm** – If True, normalize tokens.
• **stem** – If True, stem tokens.

See also:
Korean POS tags comparison chart (https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSX8/edit#gid=0)
Compare POS tags between several Korean analytic projects. (In Korean)

corpus Package

class konlpy.corpus.CorpusLoader (name=None)
Loader for corpora. For a complete list of corpora available in KoNLPy, refer to *Corpora* (page 30).

```python
>>> from konlpy.corpus import kolaw
>>> fids = kolaw.fileids()
>>> fobj = kolaw.open(fids[0])
>>> print fobj.read(140)
대한민국헌법유구한역사와전통에빛나는우리대한민국은3·1운동으로건립된대한민국임시정부의법통과불의에항한4·19민주이념을계승하고,조국의민주개혁과평화통일의사명에입각하여정의·인도와동포에별도민족의단결을공고히하고,모든사회적폐습과불의를타파하며,자율과조화를바바
```

abspath (filename=None)
Absolute path of corpus file. If filename is None, returns absolute path of corpus.

Parameters filename – Name of a particular file in the corpus.

fileids ()
List of file IDs in the corpus.

open (filename)
Method to open a file in the corpus. Returns a file object.

Parameters filename – Name of a particular file in the corpus.

data Module

ekonlpy.data.find(resource_url)
Find the path of a given resource URL by searching through directories in konlpy.data.path. If the given resource is not found, raise a LookupError, whose message gives a pointer to the installation instructions for konlpy.download().

Parameters resource_url (str (https://docs.python.org/2/library/functions.html#str)) – The URL of the resource to search for. URLs are posix-style relative path names, such as corpora/kolaw. In particular, directory names should always be separated by the forward slash character (i.e., `/`), which will be automatically converted to a platform-appropriate path separator by KoNLPy.
KonLPy Documentation, Release 0.4.4

```python
donlp.data.load(resource_url, format='auto')
```

Load a given resource from the KonLPy data package. If no format is specified, load() will attempt to determine a format based on the resource name’s file extension. If that fails, load() will raise a ValueError exception.

**Parameters**

- `resource_url` *(str)*: A URL specifying where the resource should be loaded from.
- `format` – Format type of resource.

```python
donlp.data.path = ['/home/docs/konlp_data', '/usr/share/konlp_data', '/usr/local/share/konlp_data', '/usr/lib/konlp_data', '/usr/local/lib/konlp_data', '/home/docs/checkouts/readthedocs.org/user_builds/konlp/checkouts/v0.4.4/konlp/data']
```

A list of directories where the KonLPy data package might reside. These directories will be checked in order when looking for a resource. Note that this allows users to substitute their own versions of resources.

```python
class donlp.data.FileSystemPathPointer(path)
```

A path pointer that identifies a file by an absolute path.

```python
def size()
open(encoding='utf-8')
```

```python
class donlp.data.PathPointer
```


```python
def size()
open(encoding='utf-8')
```

**downloader Module**

```python
class donlp.downloader.Downloader(download_dir=None)
```

A class used to access the KonLPy data server, which can be used to download packages.

```python
INDEX_URL = 'http://konlp.github.io/konlp-data/index.json'
INSTALLED = 'installed'
NOT_INSTALLED = 'not installed'
PACKAGE_URL = 'http://konlp.github.io/konlp-data/packages/%s.%s'
SCRIPT_URL = 'http://konlp.github.io/konlp-data/packages/%s.sh'
STALE = 'corrupt or out of date'
```

```python
download(id=None, download_dir=None)
```

The KonLPy data downloader. With this module you can download corpora, models and other data packages that can be used with KonLPy.

Individual packages can be downloaded by passing a single argument, the package identifier for the package that should be downloaded:

```python
>>> download('corpus/kobill')
[konlp_data] Downloading package 'kobill'...
[konlp_data] Unzipping corpora/kobill.zip.
```

To download all packages, simply call download with the argument `all`:
KoNLPy Documentation, Release 0.4.4

status (info_or_id=None, download_dir=None)

konlpy.downloader.default_download_dir()

Returns the directory to which packages will be downloaded by default. This value can be overridden using the constructor, or on a case-by-case basis using the download_dir argument when calling download().

On Windows, the default download directory is PYTHONHOME/lib/konlpy, where PYTHONHOME is the directory containing Python e.g., C:\Python27.

On all other platforms, the default directory is the first of the following which exists or which can be created with write permission: /usr/share/konlpy_data, /usr/local/share/konlpy_data, /usr/lib/konlpy_data, /usr/local/lib/konlpy_data, ~/konlpy_data.

jvm Module

konlpy.jvm.init_jvm(jvmpath=None)

Initializes the Java virtual machine (JVM).

Parameters

• jvmpath – The path of the JVM. If left empty, inferred by jpype.getDefaultJVMPath().

utils Module

class konlpy.utils.UnicodePrinter (indent=1, width=80, depth=None, stream=None)


format (object, context, maxlevels, level)

Overrided method to enable Unicode pretty print.

konlpy.utils.char2hex(c)

Converts a unicode character to hex.

>>> char2hex(u'음')
'\xc74c'

konlpy.utils.concordance (phrase, text, show=False)

Find concordances of a phrase in a text.

The farmost left numbers are indices, that indicate the location of the phrase in the text (by means of tokens).

The following string, is part of the text surrounding the phrase for the given index.

Parameters

• phrase – Phrase to search in the document.
• text – Target document.
• show – If True, shows locations of the phrase on the console.

>>> from konlpy.corpus import kolaw
>>> from konlpy.tag import Mecab
>>> from konlpy import utils
>>> idx = utils.concordance(u'대한민국', constitution, show=True)
0 대한민국헌법 유구한 역사와
9 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
98 충강 제1조 ① 대한민국은 민주공화국이다. ②대한민국의
koNLPy Documentation, Release 0.4.4

---

[341x798]KoNLPy Documentation, Release 0.4.4
[97x755]100 ① 대한민국은 민주공화국이다. ②대한민국의 주권은 국민에게
110 나온다. 제2조 ① 대한민국의 국민이 되는
126 의무를 전다. 제3조 대한민국의 영토는 한반도와
133 부속도서로 한다. 제4조 대한민국은 통일을 지향하며,
147 추진한다. 제5조 ① 대한민국은 국제평화의 유지에
787 군무원이 아닌 국민은 대한민국의 영역안에서는 중대한
1836 파견 또는 외국군대의 대한민국 영역안에서의 주류에
3620 경제 제159조 ① 대한민국의 경제질서는 개인과

---

**koNLPy.utils.csvread** *(f, encoding='utf-8')*

Reads a csv file.

```python
>>> from konlpy.utils import csvread
>>> with open('some.csv', 'r') as f:
...     print csvread(f)
[[u'이 / NR', u'차 / NNB'], [u'나가 / VV', u'네 / EFN']]
```

**koNLPy.utils.csvwrite** *(data, f)*

Writes a csv file.

```python
>>> from konlpy.utils import csvwrite
>>> d = [[u'이 / NR', u'차 / NNB'], [u'나가 / VV', u'네 / EFN']]
>>> with open('some.csv', 'w') as f:
...     csvwrite(d, f)
```

**koNLPy.utils.hex2char** *(h)*

Converts a hex character to unicode.

```python
>>> print hex2char('c74c')
음
>>> print hex2char('0xc74c')
음
```

**koNLPy.utils.load_text** *(filename, encoding='utf-8')*

Text file loader. To read a file, use `read_txt()` instead.

**koNLPy.utils.partition** *(list, indices)*

Partitions a list to several parts using indices.

**Parameters**

- **list** – The target list.
- **indices** – Indices to partition the target list.

**koNLPy.utils.pprint** *(obj)*

Unicode pretty printer.

```python
>>> import pprint, konlpy
>>> print pprint.pprint({u'Print', u'유니코드', u'easily'})
{'Print', '유니코드', 'easily'}
```

**koNLPy.utils.read_json** *(filename, encoding='utf-8')*

JSON file reader.

**koNLPy.utils.read_txt** *(filename, encoding='utf-8')*

Text file reader.

---

6.1. konlpy Package
konlp.utils.select(phrase)
    Replaces some ambiguous punctuation marks to simpler ones.
CHAPTER 7

Indices and tables

- genindex
- modindex
- search
- changelog
Python Module Index

k
konlp.corpus, 36
konlp.data, 36
konlp.downloader, 37
konlp.jvm, 38
konlp.tag._hannanum, 32
konlp.tag._kkma, 33
konlp.tag._komoran, 34
konlp.tag._mecab, 34
konlp.tag._twitter, 35
konlp.utils, 38
Index

A
abspath() (konlpy.corpus.CorpusLoader method), 36
analyze() (konlpy.tag._hannanum.Hannanum method), 33

C
c2hex() (in module konlpy.utils), 38
concordance() (in module konlpy.utils), 38
CorpusLoader (class in konlpy.corpus), 36
csvread() (in module konlpy.utils), 39
csvwrite() (in module konlpy.utils), 39

D
default_download_dir() (in module konlpy.downloader), 38
download() (konlpy.downloader.Downloader method), 37
Downloader (class in konlpy.downloader), 37

F
file_size() (konlpy.data.FileSystemPathPointer method), 37
file_size() (konlpy.data.PathPointer method), 37
fileids() (konlpy.corpus.CorpusLoader method), 36
FileSystemPathPointer (class in konlpy.data), 37
find() (in module konlpy.data), 36
format() (konlpy.utils.UnicodePrinter method), 38

H
Hannanum (class in konlpy.tag._hannanum), 32
hex2char() (in module konlpy.utils), 39

I
INDEX_URL (konlpy.downloader.Downloader attribute), 37
init_jvm() (in module konlpy.jvm), 38
INSTALLED (konlpy.downloader.Downloader attribute), 37

K
Kkma (class in konlpy.tag._kkma), 33
Komoran (class in konlpy.tag._komoran), 34
konlpy.corpus (module), 36
konlpy.data (module), 36
konlpy.downloader (module), 37
konlpy.jvm (module), 38
konlpy.tag._hannanum (module), 32
konlpy.tag._kkma (module), 33
konlpy.tag._komoran (module), 34
konlpy.tag._mecab (module), 34
konlpy.tag._twitter (module), 35
konlpy.utils (module), 38

L
load() (in module konlpy.data), 36
load_txt() (in module konlpy.utils), 39

M
Mecab (class in konlpy.tag._mecab), 34
morphs() (konlpy.tag._hannanum.Hannanum method), 33
morphs() (konlpy.tag._kkma.Kkma method), 33
morphs() (konlpy.tag._komoran.Komoran method), 34
morphs() (konlpy.tag._mecab.Mecab method), 35
morphs() (konlpy.tag._twitter.Twitter method), 35

N
NOT_INSTALLED (konlpy.downloader.Downloader attribute), 37
nouns() (konlpy.tag._hannanum.Hannanum method), 33
nouns() (konlpy.tag._kkma.Kkma method), 34
nouns() (konlpy.tag._komoran.Komoran method), 34
nouns() (konlpy.tag._mecab.Mecab method), 35
nouns() (konlpy.tag._twitter.Twitter method), 35

O
open() (konlpy.corpus.CorpusLoader method), 36
open() (konlpy.data.FileSystemPathPointer method), 37
open() (konlpy.data.PathPointer method), 37

P
PACKAGE_URL (konlpy.downloader.Downloader attribute), 37
partition() (in module konlpy.utils), 39
path (in module konlpy.data), 37
PathPointer (class in konlpy.data), 37
phrases() (konlpy.tag._twitter.Twitter method), 36
pos() (konlpy.tag._hannanum.Hannanum method), 33
pos() (konlpy.tag._kkma.Kkma method), 34
pos() (konlpy.tag._komoran.Komoran method), 34
pos() (konlpy.tag._mecab.Mecab method), 35
pos() (konlpy.tag._twitter.Twitter method), 36
pprint() (in module konlpy.utils), 39
R
read_json() (in module konlpy.utils), 39
read_txt() (in module konlpy.utils), 39
S
SCRIPT_URL (konlpy.downloader.Downloader attribute), 37
select() (in module konlpy.utils), 39
sentences() (konlpy.tag._kkma.Kkma method), 34
STALE (konlpy.downloader.Downloader attribute), 37
status() (konlpy.downloader.Downloader method), 38
T
Twitter (class in konlpy.tag._twitter), 35
U
UnicodePrinter (class in konlpy.utils), 38