

KoNLPy Documentation

Release 0.6.0

Lucy Park

Jan 02, 2021

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 . . . 15
- 5.5 Running tests . . . 29
- 5.6 References . . . 29

6 API **33**
- 6.1 konlpy Package . . . 33

7 Indices and tables **43**

Python Module Index **44**

Index **45**

(https://travis-ci.org/konlpy/konlpy) (https://readthedocs.org/projects/konlpy/?badge=latest) 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 33) documents.

```
>>> 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 깃헙 이슈 트래커에 남겨주세요.'))
[질문,
 건의,
 건의사항,
 사항,
 깃헙,
 이슈,
 트래커]
>>> pprint(kkma.pos(u'오류보고는 실행환경, 에러메세지와함께 설명을 최대한상세히!^^'))
[(오류, NNG),
 (보고, NNG),
 (는, JX),
 (실행, NNG),
 (환경, NNG),
 (,, SP),
 (에러, NNG),
 (메세지, NNG),
 (와, JKM),
 (함께, MAG),
 (설명, NNG),
 (을, JKO),
 (최대한, NNG),
 (상세히, MAG),
 (!, SF),
 (^^, EMO)]
```

CHAPTER 1

Standing on the shoulders of giants

Korean, the 13th most widely spoken language in the world (http://www.koreatimes.co.kr/www/news/nation/2014/05/116_157214.ht is a beautiful, yet complex language. Myriad *Korean morpheme analyzer tools* (page 29) 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/konlpy/konlpy/issues) when you think any have gone stale.

CHAPTER 2

License

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.

- Eunjeong L. Park, Sungzoon Cho. "KoNLPy: Korean natural language processing in Python (http://dmlab.snu.ac.kr/~lucypark/docs/2014-10-10-hclt.pdf)", Proceedings of the 26th Annual Conference on Human & Cognitive Language Technology, Chuncheon, Korea, Oct 2014.

Here is a BibTeX entry.:

```
@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}
}
```

CHAPTER 3

Contribute

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

4.1 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))
- Dialogue systems (ex: Apple Siri (https://www.apple.com/ios/siri/))
- 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 to the so-called "bibles":

- Jurafsky et al., Speech and Language Processing (https://www.goodreads.com/book/show/908048), 2nd Edition, 2008.
- Manning and Schutze, Foundations of Statistical Natural Language Processing (https://www.goodreads.com/book/show/776349), 1999.

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).

4.2 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]
3. A "good" text editor and terminal (or Python IDE)[2]

[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://learnpythonthehardway.org/book/).

[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://pedrokroger.net/choosing-best-python-ide/) out there, so take your pick!

4. A Python installation (https://wiki.python.org/moin/BeginnersGuide/Download)
5. pip (https://pypi.python.org/pypi/pip), the Python package manager

Got 'em all? Then let's go.

CHAPTER 5

User guide

5.1 Installation

Note: For troubleshooting information, see these pages: Linux (https://github.com/konlpy/konlpy/issues?q=label%3AOS%2FLinux) Mac OS (https://github.com/konlpy/konlpy/issues?q=label%3AOS%2FMacOS). Windows (https://github.com/konlpy/konlpy/issues?q=label%3AOS%2FWindows). Please record a "New Issue" (https://github.com/konlpy/konlpy/issues/new) if you have an error that is not listed.

Warning: Python 2 End of Life Announced as January 1st 2020.

This page covers Python 3 only. The last-known version for Python 2 is v0.5.2, install it via *pip install konlpy==v0.5.2*.

5.1.1 Ubuntu

Supported: Xenial(16.04.3 LTS), Bionic(18.04.3 LTS), Disco(19.04), Eoan(19.10)

1. Install dependencies

```
# Install Java 1.8 or up
$ sudo apt-get install g++ openjdk-8-jdk python3-dev python3-pip curl
```

2. Install KoNLPy

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install konlpy       # Python 3.x
```

3. Install MeCab (*optional*)

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

5.1.2 CentOS

Supported: CentOS 7, 8

1. Install dependencies

```
$ sudo yum install gcc-c++ java-1.8.0-openjdk-devel python3 python3-
↪devel python3-pip make diffutils
```

2. Install KoNLPy

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install konlpy     # Python 3.x
```

3. Install MeCab (*optional*)

```
$ sudo yum install curl git
$ bash <(curl -s https://raw.githubusercontent.com/konlpy/konlpy/
↪master/scripts/mecab.sh)
```

5.1.3 Mac OS

1. Install KoNLPy

```
$ python3 -m pip install --upgrade pip
$ python3 -m pip install konlpy        # Python 3.x
```

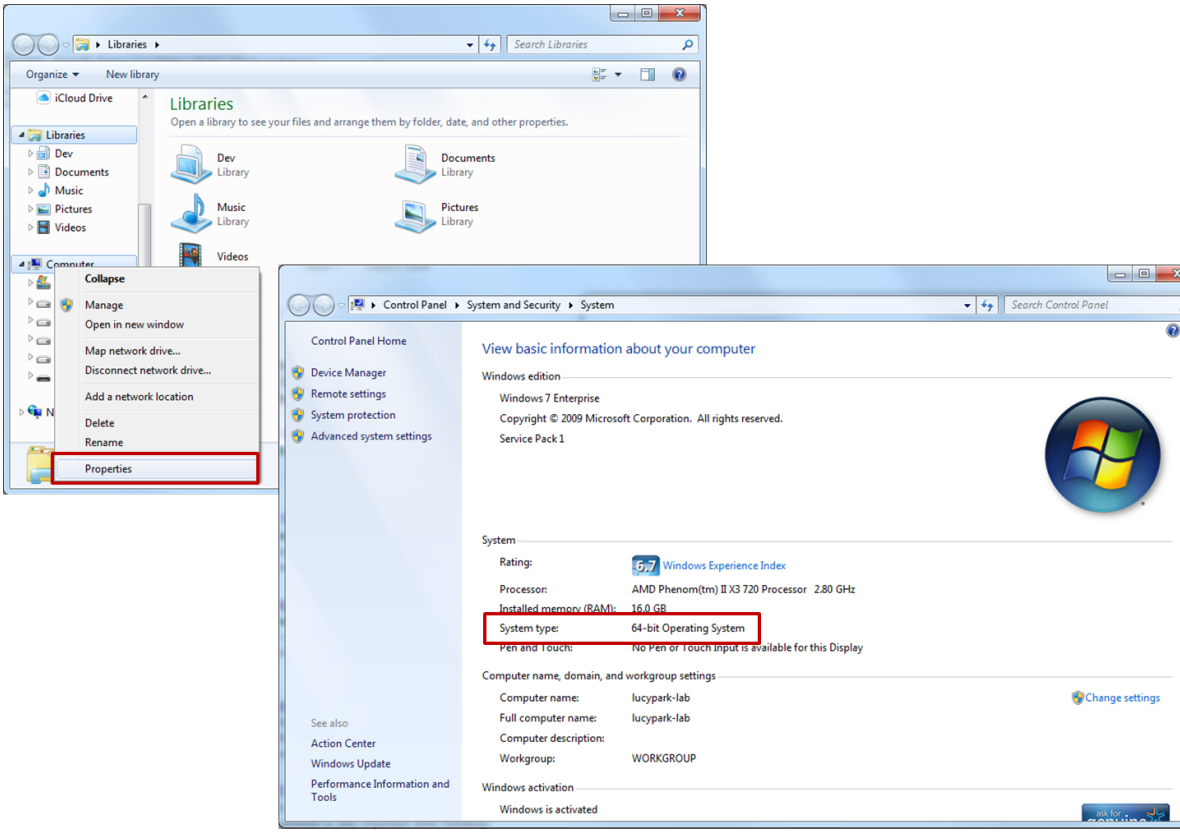
2. Install MeCab (*optional*)

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

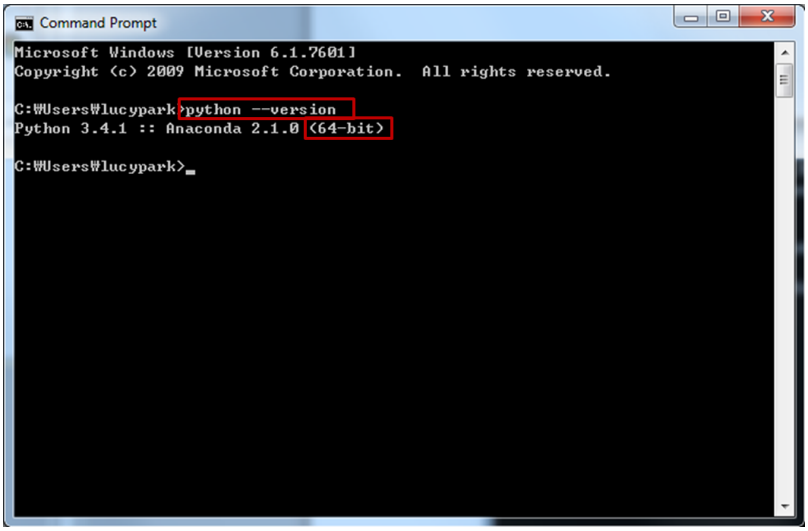
5.1.4 Windows

1. **Does your Python installation's "bit version" match your Windows OS? If you're using a 64 bit Windows you need a 6**

    - How to check your Windows bit version



- How to check your Python bit version



2. Do you have a Java of version 1.7 or above installed, that matches your OS bit version? If not, download and install a JDK (http://www.oracle.com/technetwork/java/javase/downloads/index.html). Note again, that the bit versions must match.
3. Set JAVA_HOME (http://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/index.html).
4. Download and install the JPype1 (>=0.5.7) (http://www.lfd.uci.edu/~gohlke/pythonlibs/#jpype) that matches your bit version: *win32* for 32 bit and *win-amd64* for 64 bit. You may have to upgrade your pip (https://pip.pypa.io/en/stable/installing.html#upgrade-pip) in order to install the downloaded *.whl* file.

```
> pip install --upgrade pip
> pip install JPype1-0.5.7-cp27-none-win_amd64.whl
```

5. From the command prompt, install KoNLPy.

```
> pip install konlpy
```

Warning:

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

5.1.5 Docker

If you are familiar with Docker, it is easy to install *konlpy* and *java-1.7-openjdk* on *python:3* image.

```
> FROM python:3

> ENV JAVA_HOME /usr/lib/jvm/java-1.7-openjdk/jre
> RUN apt-get update && apt-get install -y g++ default-jdk
> RUN pip install konlpy

> # Write left part as you want
```

5.2 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 + ㄴ다/EFN
```

5.2.1 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 33).

See also:

Korean POS tags comparison chart (https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0)

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

5.2.2 Comparison between POS tagging classes

Now, we do time and performation analysis for executing the `pos` method for each of the classes in the *tag Package* (page 33). The experiments were carried out on a Intel i7 CPU with 4 cores, Python 2.7, and KoNLPy 0.4.1.

Time analysis[1]

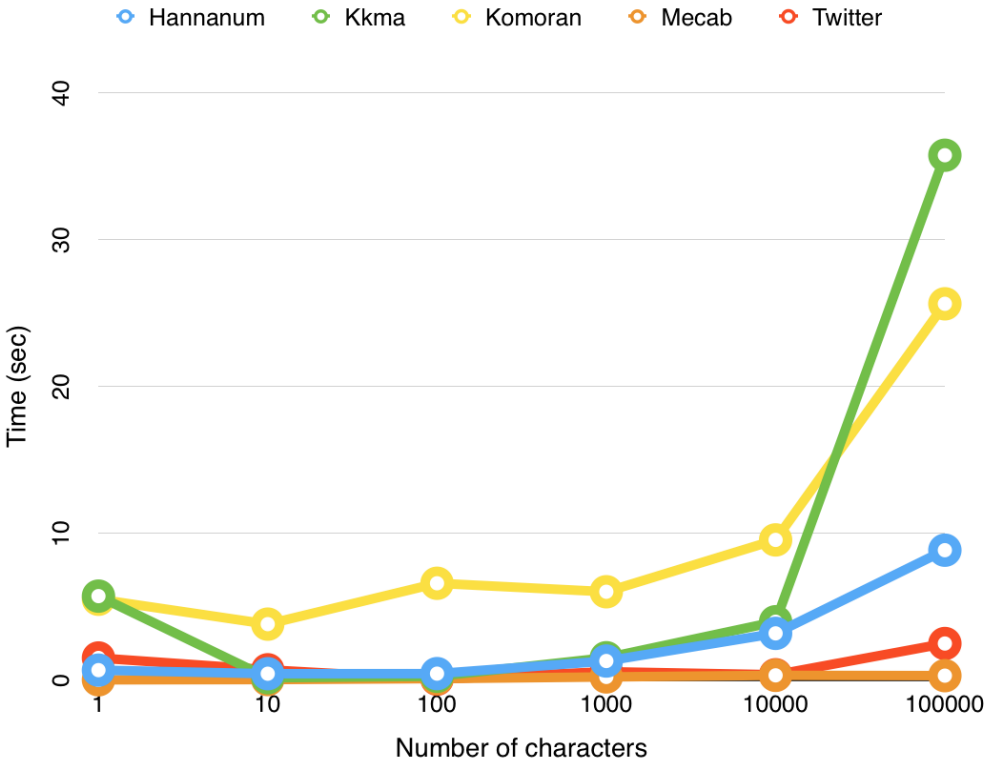
1. *Loading time*: Class loading time, including dictionary loads.
    - *Kkma* (page 34): 5.6988 *secs*

[1] Please note that these are comparisons among KoNLPy classes, and not the original distributions.

- *Komoran* (page 35): 5.4866 *secs*
- *Hannanum* (page 33): 0.6591 *secs*
- *Okt* (page 37) (previous Twitter): 1.4870 *secs*
- *Mecab* (page 36): 0.0007 *secs*

2. *Execution time*: Time for executing the pos method for each class, with 100K characters.
   - *Kkma* (page 34): 35.7163 *secs*
   - *Komoran* (page 35): 25.6008 *secs*
   - *Hannanum* (page 33): 8.8251 *secs*
   - *Okt* (page 37) (previous Twitter): 2.4714 *secs*
   - *Mecab* (page 36): 0.2838 *secs*

   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. *"아버지가방에들어가신다"*

   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 33) and *Komoran* (page 35) are careful in spacing uncertain terms, and defaults the whole phrase to nouns. *Kkma* (page 34) is more confident, but gets undesirable results. For this result, *Mecab* (page 36) shows the best results.

Hannanum	Kkma	Komoran	Mecab	Twitter
아버지가방에들어가 / N	아버지 / NNG	아버지가방에들어가신다 / NNP	아버지 / NNG	아버지 / Noun
이 / J	가방 / NNG		가 / JKS	가방 / Noun
시ㄴ다 / E	에 / JKM		방 / NNG	에 / Josa
	들어가 / VV		에 / JKB	들어가신 / Verb
	시 / EPH		들어가 / VV	다 / Eomi
	ㄴ다 / EFN		신다 / EP+EC	

2. *"나는 밥을 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(-ㄹ다)/V + 는/E. *Kkma* (page 34) properly understands the latter "나는" as a verb, wheras the rest observe it as nouns.

Hannanum	Kkma	Komoran	Mecab	Twitter
나 / N	나 / NP	나 / NP	나 / NP	나 / Noun
는 / J	는 / JX	는 / JX	는 / JX	는 / Josa
밥 / N	밥 / NNG	밥 / NNG	밥 / NNG	밥 / Noun
을 / J	을 / JKO	을 / JKO	을 / JKO	을 / Josa
먹 / P	먹 / VV	먹 / VV	먹 / VV	먹는 / Verb
는다 / E	는 / EPT	는다 / EC	는다 / EC	다 / Eomi
	다 / EFN			

Hannanum	Kkma	Komoran	Mecab	Twitter
하늘 / N	하늘 / NNG	하늘 / NNG	하늘 / NNG	하늘 / Noun
을 / J	을 / JKO	을 / JKO	을 / JKO	을 / Josa
나 / N	날 / VV	나 / NP	나 / NP	나 / Noun
는 / J	는 / ETD	는 / JX	는 / JX	는 / Josa
자동차 / N	자동차 / NNG	자동차 / NNG	자동차 / NNG	자동차 / Noun

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?

Hannanum	Kkma	Komoran	Mecab	Twitter
아이폰 / N	아이 / NNG	아이폰 / NNP	아이폰 / NNP	아이폰 / Noun
기다리 / P	폰 / NNG	기다리 / VV	기다리 / VV	기다리 / Verb
다 / E	기다리 / VV	다 / EC	다 / EC	다 / Eomi
지치 / P	다 / ECS	지치 / VV	지쳐 / VV+EC	지쳐 / Verb
어 / E	지치 / VV	어 / EC	애플 / NNP	애플 / Noun
애플공홈 / N	어 / ECS	애플 / NNP	공 / NNG	공홈 / Noun
에서 / J	애플 / NNP	공 / NNG	홈 / NNG	에서 / Josa
언락폰질러버렸다 / N	공 / NNG	홈 / NNG	에서 / JKB	언락폰 / Noun
6+ / N	홈 / NNG	에서 / JKB	언락 / NNG	질 / Verb
128기가실벜 / N	에서 / JKM	언 / NNG	폰 / NNG	러 / Eomi
	언락 / NNG	락 / NNG	질러버렸 / VV+EC+VX+EP	버렸 / Verb
	폰 / NNG	폰 / NNG	다 / EC	다 / Eomi
	질르 / VV	지르 / VV	6 / SN	6 / Number
	어 / ECS	어 / EC	+ / SY	+ / Punctuation
	버리 / VXV	버리 / VX	128 / SN	128 / Number
	었 / EPT	었 / EP	기 / NNG	기 / Noun
	다 / ECS	다 / EC	가 / JKS	가 / Josa
	6 / NR	6 / SN	실버 / NNP	실버 / Noun
	+ / SW	+ / SW	ㅋ / UNKNOWN	ㅋ / KoreanParticle
	128 / NR	128기가실벜 / NA		
	기가 / NNG			
	실버 / NNG			
	ㅋ / UN			

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.

5.3 Data

5.3.1 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 38).

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

유구한 역사와
```

(continues on next page)

(continued from previous page)

```
>>> from konlpy.corpus import kobill
>>> d = kobill.open('1809890.txt').read()
>>> print d[:15]
지방공무원법 일부개정법률안
```

5.3.2 Dictionaries

Dictionaries are used for *Morphological analysis and POS tagging* (page 10), and are built with *Corpora* (page 31).

`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
나라기획회장      ncn
나라꽃  ncn
나라님  ncn
나라도둑          ncn
나라따르          pvg
나라링링프로덕션          ncn
나라말  ncn
나라망신          ncn
나라박물관        ncn
나라발전          ncpa
나라별  ncn
나라부동산        nqq
나라사랑          ncn
나라살림          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
그러나/MAC
그러니까/MAC
그러면/MAC
그러므로/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.:

```
가오티,0,0,0,NNG,*,F,가오티,*,*,*,*,*
갑툭튀,0,0,0,NNG,*,F,갑툭튀,*,*,*,*,*
강퇴,0,0,0,NNG,*,F,강퇴,*,*,*,*,*
개드립,0,0,0,NNG,*,T,개드립,*,*,*,*,*
갠소,0,0,0,NNG,*,F,갠소,*,*,*,*,*
고퀄,0,0,0,NNG,*,T,고퀄,*,*,*,*,*
광삭,0,0,0,NNG,*,T,광삭,*,*,*,*,*
광탈,0,0,0,NNG,*,T,광탈,*,*,*,*,*
굉천,0,0,0,NNG,*,T,굉천,*,*,*,*,*
국을,0,0,0,NNG,*,T,국을,*,*,*,*,*
귀요미,0,0,0,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.

5.4 Examples

Below are a set of example tasks using KoNLPy.

Exploring a document (page 15)

Exploring a corpus (page 27)

Finding collocations (page 20)

Chunking (page 21)

Generating random text (page 23)

Drawing a word cloud (page 24)

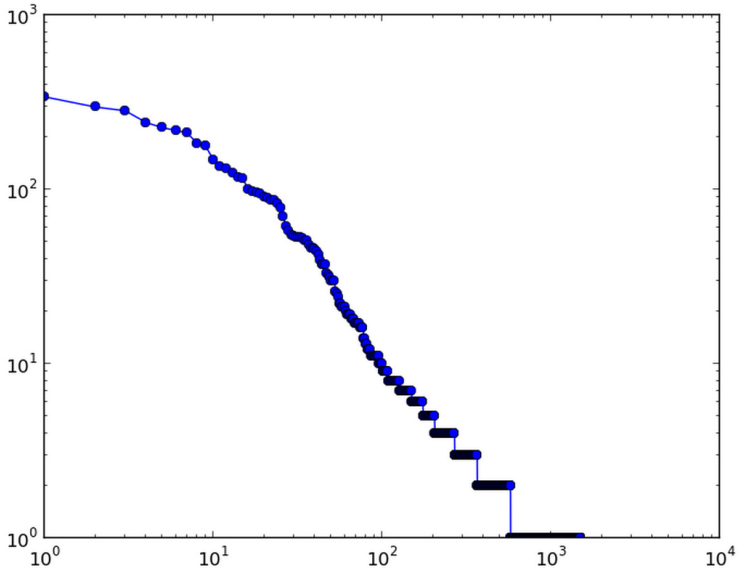
Multithreading with KoNLPy (page 26)

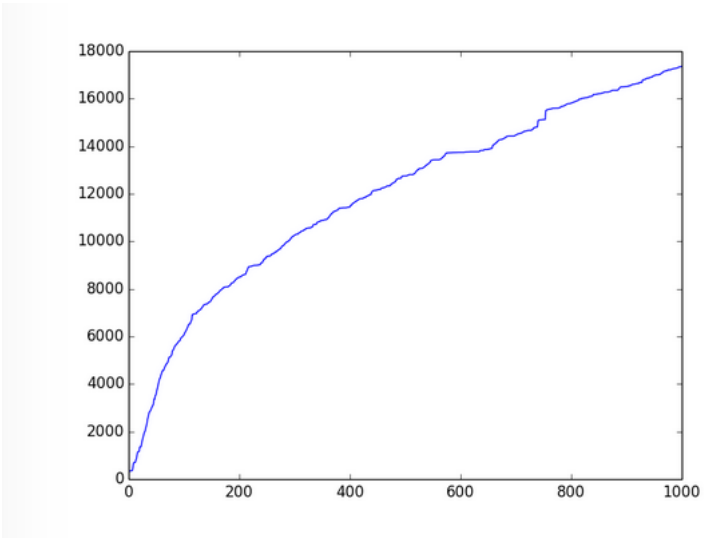
5.4.1 Contents

Exploring a document

Exploring a document can consist of various components:

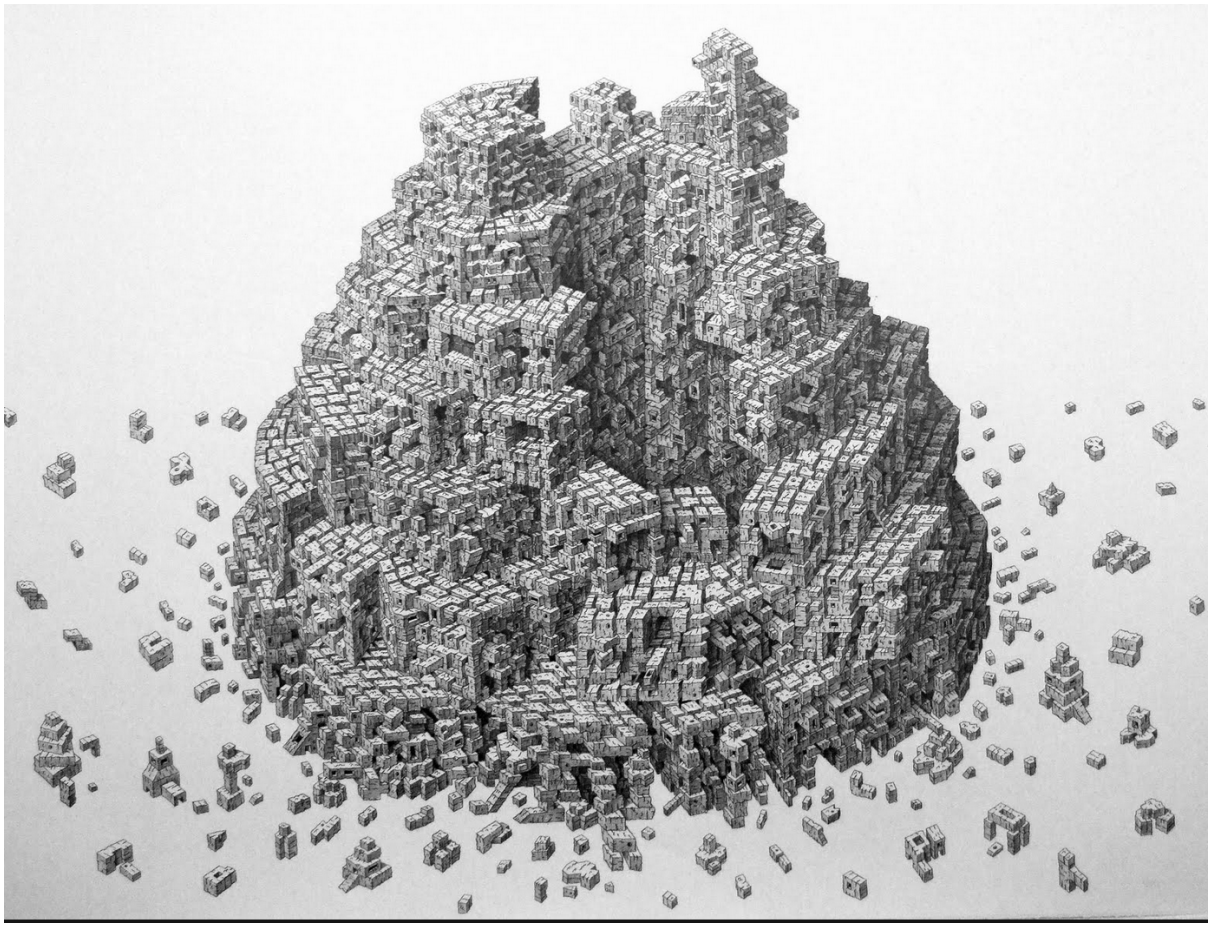
- Counts (characters, words, etc.)
- Checking Zipf's laws: $fr = k$


10^3
10^2
10^1
10^0
10^0
10^1
10^2
10^3
10^4

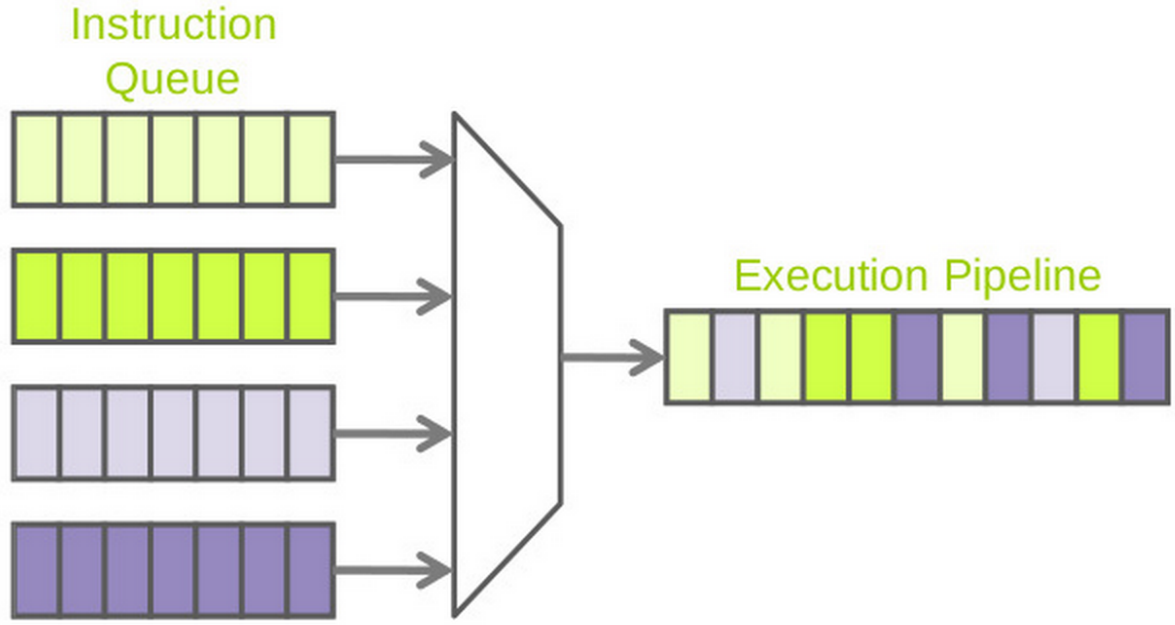

18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
200
400
600
800
1000




S
NP
의 J
NP
6세 N
이하 N
초등학교 N
취학 N




행위
동물
인터넷
판매 · 전시 · 전달 · 상영
학대
게재
영상물
신설
금지
정당
현행법
개정법률안
목적
제8조
누구
고통
다음
활동
적절
제외
질병
표시
의식
동물보호
일부
말
방치
신체
일체
위반
제8조제5항
제1항부터
과태료
홍보


Instruction
Queue
Execution Pipeline

- Concordances

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

from collections import Counter

from konlpy.corpus import kolaw
from konlpy.tag import Hannanum
from konlpy.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)))
print('\nTop 20 frequent morphemes:'); pprint(cnt.most_common(20))
print('\nLocations of "대한민국" in the document:')
concordance(u'대한민국', doc, show=True)

draw_zipf(cnt.values(), 'zipf.png')
```

- Console:

```
nchars  : 19240
ntokens : 4178
nmorphs : 1501

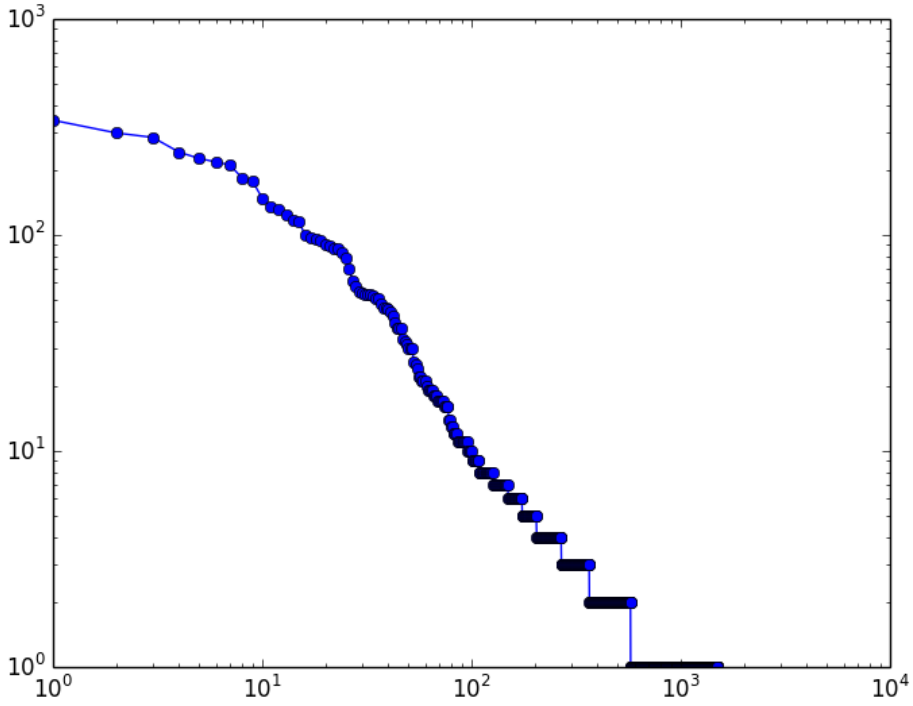
Top 20 frequent morphemes:
[((의, J), 398),
 ((., S), 340),
 ((하, X), 297),
 ((에, J), 283),
 ((ㄴ다, E), 242),
 ((ㄴ, E), 226),
 ((이, J), 218),
 ((을, J), 211),
 ((은, J), 184),
 ((어, E), 177),
 ((를, J), 148),
 ((ㄹ, E), 135),
 ((/, S), 131),
 ((하, P), 124),
 ((는, J), 117),
 ((법률, N), 115),
 ((,, S), 100),
 ((는, E), 97),
 ((있, P), 96),
 ((되, X), 95)]

Locations of "대한민국" in the document:
```

(continues on next page)

(continued from previous page)

```
0 대한민국헌법 유구한 역사와
9 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
98 총강 제1조 ① 대한민국은 민주공화국이다. ②대한민국의
100 ① 대한민국은 민주공화국이다. ②대한민국의 주권은 국민에게
110 나온다. 제2조 ① 대한민국의 국민이 되는
126 의무를 진다. 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 제119조 ① 대한민국의 경제질서는 개인과
```



- **zipf.png:**

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*.

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

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


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

print('\nCollocations among tagged words:')
```

(continues on next page)

(continued from previous page)

```
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('\nCollocations among words:')
words = [w for w, t in tagged_words]
ignored_words = [u'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('\nCollocations 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), (담임, NNG)),
 ((공중, NNG), (도덕, NNG)),
 ((과반, NNG), (수가, NNG)),
 ((교전, NNG), (상태, NNG)),
 ((그러, VV), (나, ECE)),
 ((기본적, NNG), (인권, NNG))]

Collocations among words:
[(현행, 범인),
 (형의, 선고),
 (내부, 규율),
 (정치적, 중립성),
 (누구, 든지),
 (회계, 연도),
 (지체, 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".

Here we demonstrate a way to easily chunk a sentence, and find noun, verb and adjective phrases in Korean text, using `nltk:nltk.chunk.regexp.RegexpParser`.

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

(continues on next page)

(continued from previous page)

```
import konlpy
import nltk

# POS tag a sentence
sentence = u'만 6세 이하의 초등학교 취학 전 자녀를 양육하기 위해서는'
words = konlpy.tag.Twitter().pos(sentence)

# Define a chunk grammar, or chunking rules, then chunk
grammar = """
NP: {<N.*>*<Suffix>?}   # Noun phrase
VP: {<V.*>*}            # Verb phrase
AP: {<A.*>*}            # Adjective phrase
"""
parser = nltk.RegexpParser(grammar)
chunks = parser.parse(words)
print("# Print whole tree")
print(chunks.pprint())

print("\n# Print noun phrases only")
for subtree in chunks.subtrees():
    if subtree.label()=='NP':
        print(' '.join((e[0] for e in list(subtree))))
        print(subtree.pprint())

# Display the chunk tree
chunks.draw()
```

According to the chunk grammer defined above, we have three rules to extracted 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).

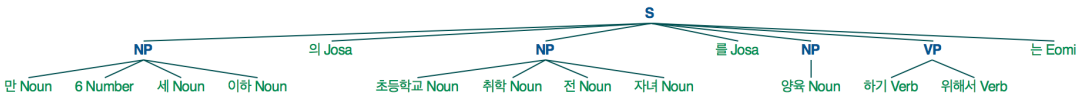
The result is a tree, which we can print on the console, or display graphically as follows.

- 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 취학 전 자녀
(NP 초등학교/Noun 취학/Noun 전/Noun 자녀/Noun)
양육
(NP 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[1]:

ㅓ키ㅖ츄ㅏㅋ응아 ㄲㄴㅇ 쿄ㅛㅍㄹㅎ튜허ㅓㅎㅠㅅㄴ마

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[2]. (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:

로켁불됴약 니앙굷 췡바이뀗노기

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

다이는가 고다하에지 요그이데습

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 '하' follows '다', and after '그' we get '리' and '고'. In mathematical terms, this process is better known as a Markov chain (http://en.wikipedia.org/wiki/Markov_chain):

국의의하되고인정부가는 요구한 대통령은 2조 사면 기밀과 헌법률로 위하의 위하며

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.

Warning: The code below works with Python3, and not with Python2! You can run the code by typing *python3 generate.py* on your terminal.

```
#! /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)

        # Generate the next word based on probability
```

(continues on next page)

[1] This story would actually feature a monkey instead of a cat. Namely by the Infinite monkey theorem (http://en.wikipedia.org/wiki/Infinite_monkey_theorem).

[2] Please refer to the Hangul Jamo in Unicode character code charts (http://www.unicode.org/charts/).

(continued from previous page)

```
        choices, weights = zip(*cfdist[word].items())
        cumdist = list(itertools.accumulate(weights))
        x = random.random() * cumdist[-1]
        word = choices[bisect.bisect(cumdist, x)]

    return ' '.join(sentence)


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. 국민 은 법률 로 인한 배상 은 특별 한 영장 을 청구 할 수 있 어서 최고 득표자 가 제출 한 유
일 한 때 에 의하 여 는 경우 를 선거 관리 할 수 없 을 포함 한 사항 은 청구 할 수 있 다
1. 국민 투표 의 범죄 에 의하 여 발언 하 지 아니한 회의 는 요건 은 1988 년 으로 대통령 이 의
결 한다
2. 국민 경제 자문 기구 를 타파 하 기 위하 여 긴급 한 형태 로 정한다
3. 국민 은 이 정하 는 헌법 시행 당시 의 심사 할 수 있 다
4. 국민 의 기본 질서 를 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 Bently'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)

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

from collections import Counter
import urllib
```

(continues on next page)

(continued from previous page)

```
import random
import webbrowser

from konlpy.tag import Hannanum
from lxml import html
import pytagcloud # requires Korean font support
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).



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.

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

from konlpy.tag import Kkma
from konlpy.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

    print('Number of lines in document:')
    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)
```

(continues on next page)

(continued from previous page)

```
    print('Concurrent tagging:')
    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:

```
Number of lines in document:
356
Batch tagging:
37.758173
Concurrent tagging:
8.037602
```

Check out how much faster it gets!

Note:

- **Some useful references on concurrency with Python:**
  - 장혜식, "파이썬은 멀티코어 쥐도 쓰잘데기가 없나요?"에 대한 파이썬 2.6의 대답 (http://highthroughput.org/wp/python-multiprocessing/), 2008.
  - 하용호, 파이썬으로 클라우드 하고 싶어요 (http://www.slideshare.net/devparan/h3-2011-c6-python-and-cloud), 2011.

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).

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

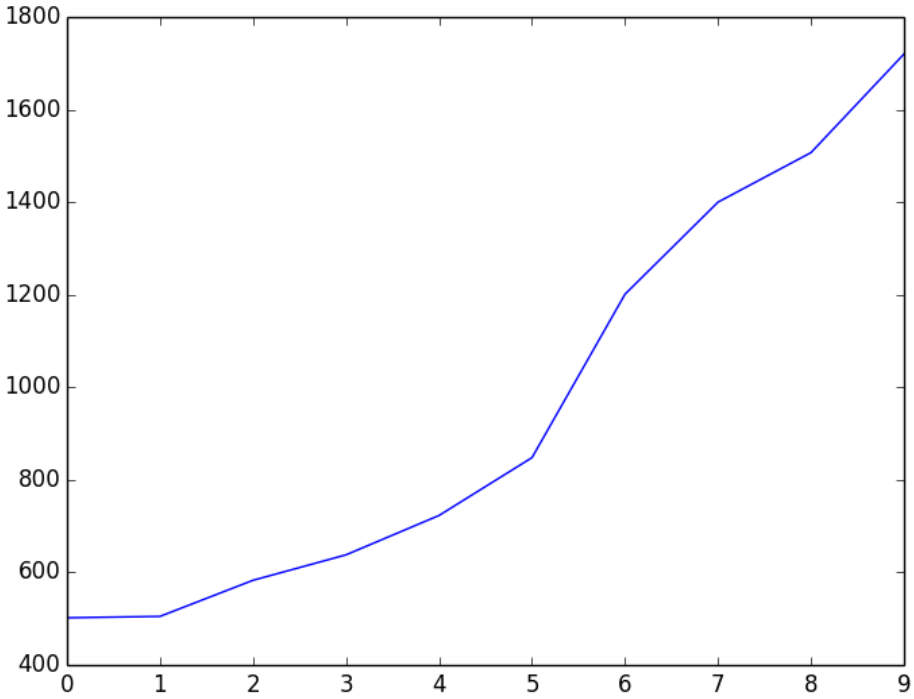
# get global unique token counts
global_unique = []
global_unique_cnt = []
for doc in docs:
    tokens = pos(doc)
    unique = set(tokens)
    global_unique += list(unique)
    global_unique = list(set(global_unique))
    global_unique_cnt.append(len(global_unique))
```

(continues on next page)

(continued from previous page)

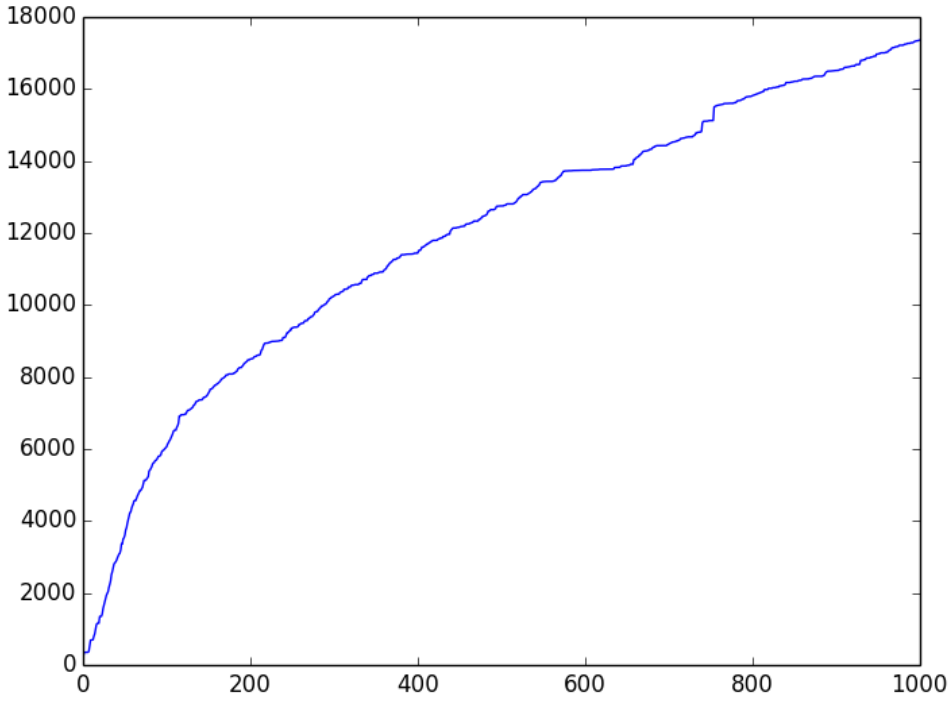
```
    print(len(unique), len(global_unique))

# draw heap
plt.plot(global_unique_cnt)
plt.savefig('heap.png')
```



- **heap.png:**

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.



- **heap-1000.png:**

5.5 Running tests

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

```
$ 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_L9NF9gSLbsJOGqsf-zfqTtyhhthmJWNC2kgUDIsU/edit?usp=sharing).

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

5.6 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 Jan 02, 2021.

5.6.1 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
- **UTagger (http://nlplab.ulsan.ac.kr/Demo/ProjectDemo.html) (2012) - By Joon-Choul Shin, Cheol-Young Ock* (Ulsan**
  - 신준철, 옥철영, 기분석 부분 어절 사전을 활용한 한국어 형태소 분석기 (A Korean Morphological Analyzer using a Pre-analyzed Partial Word-phrase Dictionary) (http://www.dbpia.co.kr/Journal/ArticleDetail/NODE01873335), 정보과학회논문지: 소프트웨어 및 응용, 제39권 제5호, 2012.
  - 신준철, 옥철영, 한국어 품사 및 동형이의어 태깅을 위한 단계별 전이모델 (A Stage Transition Model for Korean Part-of-Speech and Homograph Tagging) (http://www.dbpia.co.kr/Journal/ArticleDetail/NODE02033338), 정보과학회논문지: 소프트웨어 및 응용, 제39권 제11호, 2012.
  - slides (http://www.slideserve.com/mills/u-tagger-2013)
- **MACH (http://cs.sungshin.ac.kr/~shim/demo/mach.html) (2002) - By Kwangseob Shim (성신여대) custom**
  - Kwangseob Shim, Jaehyung Yang, MACH: A Supersonic Korean Morphological Analyzer (http://www.aclweb.org/anthology/C02-1092), ACL, 2002.
- **KTS (http://wiki.kldp.org/wiki.php/KTS) (1995) - By 이상호, 서정연, 오영환 (KAIST) GPL v2**
  - 이상호, KTS: Korean Tagging System Manual (Version 0.9) (https://wiki.kldp.org/wiki.php/KTS?action=download&value=ktsmanual.pdf)
  - 김재훈, 서정연, 자연언어 처리를 위한 한국어 품사 태그 (A Korean part-of-speech tag set for natural language processing) (https://wiki.kldp.org/wiki.php/KTS?action=download&value=tagset.pdf), 1993.
  - Created at 1995, released at 2002.[1]

Java/Scala

- twitter-korean-text (https://github.com/twitter/twitter-korean-text/) (2014) - By Will Hohyon Ryu (Twitter) Apache v2
- KOMORAN (http://shineware.tistory.com/tag/KOMORAN) (2013) - By 신준수 (shineware) Apache v2
- **KKMA (http://kkma.snu.ac.kr) (2010) - By Sang-goo Lee*, Dongjoo Lee (http://therocks.tistory.com), et al. (Seoul Nat**
  - 이동주, 연종흠, 황인범, 이상구, 꼬꼬마: 관계형 데이터베이스를 활용한 세종 말뭉치 활용 도구 (http://ids.snu.ac.kr/w/images/f/f8/CPL2010-therocks.pdf), 정보과학회논문지: 컴퓨팅의 실제 및 레터, Volume 16, No.11, 2010.
- **Arirang (http://cafe.naver.com/korlucene) (2009) - By SooMyung Lee 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) G**
  - 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://konlpy.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) (2014) MIT**
  - By Kyunghoon Kim (UNIST)
  - Wrapper for MeCab-ko for online usage

R

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

5.6.2 Corpora

- Korean Universal Dependency (UD) Treebank (https://github.com/UniversalDependencies/UD_Korean), 2013.
- **Korea University Korean Corpus, 1995.**
  - 10M tokens of Korean of 1970-90s
- **HANTEC 2.0 (http://www.kristalinfo.com/download/#hantec), KISTI & 충남대, 1998-2003.**
  - 120,000 test documents (237MB)
  - 50 TREC-type questions for QA (48KB)
- **HKIB-40075 (http://www.kristalinfo.com/TestCollections/readme_hkib.html), KISTI & 한국일보, 2002.**
  - 40,075 test documents for text categorization (88MB)
- KAIST Corpus (http://semanticweb.kaist.ac.kr/home/index.php/KAIST_Corpus), KAIST, 1997-2005.
- Sejong Corpus (http://www.sejong.or.kr/), National Institute of the Korean Language, 1998-2007.
- **Yonsei Corpus, 연세대, 1987.**
  - 42M tokens of Korean since the 1960s
- BoRA 언어자원은행 (http://semanticweb.kaist.ac.kr/org/bora/), KAIST

5.6.3 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, K**
  - Language parser
- **Korean (http://pythonhosted.org/korean) - By Heungsub Lee Python**
  - Package for attaching particles (josa) in sentences
- **go_hangul (https://github.com/suapapa/go_hangul) (2012) - By Homin Lee Go BSD**
  - Tools for Hangul manipulation [docs] (https://godoc.org/github.com/suapapa/go_hangul)
- Speller (http://speller.cs.pusan.ac.kr/) (부산대)

CHAPTER 6

API

6.1 konlpy Package

6.1.1 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*, *max_heap_size=1024*)

Bases: object (https://docs.python.org/3/library/functions.html#object)

Wrapper for JHannanum (http://semanticweb.kaist.ac.kr/home/index.php/HanNanum).

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.

```
>>> from konlpy.tag import Hannanum
>>> hannanum = Hannanum()
>>> print(hannanum.analyze(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다.'))
[[[('롯데마트', 'ncn'), ('의', 'jcm')], [('롯데마트의', 'ncn')], [('롯데마트', 'nqq
↪'), ('의', 'jcm')], [('롯데마트의', 'nqq')]], [[('흑마늘', 'ncn')], [('흑마늘',
↪'nqq')]], [[('양념', 'ncn')]], [[('치킨', 'ncn'), ('이', 'jcc')], [('치킨', 'ncn
↪'), ('이', 'jcs')], [('치킨', 'ncn'), ('이', 'ncn')]], [[('논란', 'ncpa'), ('이
↪', 'jcc')], [('논란', 'ncpa'), ('이', 'jcs')], [('논란', 'ncpa'), ('이', 'ncn
↪')]], [[('되', 'nbu'), ('고', 'jcj')], [('되', 'nbu'), ('이', 'jp'), ('고',
↪'ecc')], [('되', 'nbu'), ('이', 'jp'), ('고', 'ecs')], [('되', 'nbu'), ('이',
↪'jp'), ('고', 'ecx')], [('되', 'paa'), ('고', 'ecc')], [('되', 'paa'), ('고',
↪'ecs')], [('되', 'paa'), ('고', 'ecx')], [('되', 'pvg'), ('고', 'ecc')], [('되
↪', 'pvg'), ('고', 'ecs')], [('되', 'pvg'), ('고', 'ecx')], [('되', 'px'), ('고
↪', 'ecc')], [('되', 'px'), ('고', 'ecs')], [('되', 'px'), ('고', 'ecx')]], [[(
↪'있', 'paa'), ('다', 'ef')], [('있', 'px'), ('다', 'ef')]], [[('.', 'sf')], [(
↪'.', 'sy')]]]
```

(continues on next page)

(continued from previous page)

```
>>> print(hannanum.morphs(u'롯데마트의 흑마늘 양념 치킨이 논란이 되고 있다.'))
['롯데마트', '의', '흑마늘', '양념', '치킨', '이', '논란', '이', '되', '고', '있', '다
↪', '.']
>>> print(hannanum.nouns(u'다람쥐 헌 쳇바퀴에 타고파'))
['다람쥐', '쳇바퀴', '타고파']
>>> print(hannanum.pos(u'웃으면 더 행복합니다!'))
[('웃', 'P'), ('으면', 'E'), ('더', 'M'), ('행복', 'N'), ('하', 'X'), ('ㅂ니다', 'E
↪'), ('!', 'S')]
```

Parameters

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 40).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 40).

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*, *join=False*)
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 eojeols.
- **join** – If True, returns joined sets of morph and tag.

Kkma Class

class konlpy.tag._kkma.**Kkma** (*jvmpath=None*, *max_heap_size=1024*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

Wrapper for Kkma (http://kkma.snu.ac.kr).

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).

```
>>> from konlpy.tag import Kkma
>>> kkma = Kkma()
>>> print(kkma.morphs(u'공부를 하면할수록 모르는게 많다는 것을 알게 됩니다.'))
['공부', '를', '하', '면', '하', 'ㄹ수록', '모르', '는', '것', '이', '많', '다는', '것
↪', '을', '알', '게', '되', 'ㅂ니다', '.']
>>> print(kkma.nouns(u'대학에서 DB, 통계학, 이산수학 등을 배웠지만...'))
['대학', '통계학', '이산', '이산수학', '수학', '등']
>>> print(kkma.pos(u'다 까먹어버렸네요?ㅋㅋ'))
[('다', 'MAG'), ('까먹', 'VV'), ('어', 'ECD'), ('버리', 'VXV'), ('었', 'EPT'), (
↪'네요', 'EFN'), ('?', 'SF'), ('ㅋㅋ', 'EMO')]
>>> print(kkma.sentences(u'그래도 계속 공부합니다. 재밌으니까!'))
['그래도 계속 공부합니다.', '재밌으니까!']
```

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 40).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 40).

morphs(*phrase*)
Parse phrase to morphemes.

nouns(*phrase*)
Noun extractor.

pos(*phrase*, *flatten=True*, *join=False*)
POS tagger.

Parameters

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

sentences(*phrase*)
Sentence detection.

Komoran Class

class konlpy.tag._komoran.**Komoran**(*jvmpath=None*, *userdic=None*, *modelpath=None*, *max_heap_size=1024*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

Wrapper for KOMORAN (https://github.com/shin285/KOMORAN).

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

```
>>> cat /tmp/dic.txt  # Place a file in a location of your choice
코모란      NNP
오픈소스     NNG
바람과 함께 사라지다      NNP
>>> from konlpy.tag import Komoran
>>> komoran = Komoran(userdic='/tmp/dic.txt')
>>> print(komoran.morphs(u'우왕 코모란도 오픈소스가 되었어요'))
['우왕', '코모란', '도', '오픈소스', '가', '되', '었', '어요']
>>> print(komoran.nouns(u'오픈소스에 관심 많은 멋진 개발자님들!'))
['오픈소스', '관심', '개발자']
>>> print(komoran.pos(u'혹시 바람과 함께 사라지다 봤어?'))
[('혹시', 'MAG'), ('바람과 함께 사라지다', 'NNP'), ('보', 'VV'), ('았', 'EP'), ('어',
↪ 'EF'), ('?', 'SF')]
```

Parameters

- **jvmpath** – The path of the JVM passed to *init_jvm()* (page 40).
- **userdic** – The path to the user dictionary.

  This enables the user to enter custom tokens or phrases, that are mandatorily assigned to tagged as a particular POS. Each line of the dictionary file should consist of a token or phrase, followed by a POS tag, which are delimited with a <*tab*> character.

  An example of the file format is as follows:

```
바람과 함께 사라지다 NNG
바람과 함께 NNP
자연어 NNG
```

If a particular POS is not assigned for a token or phrase, it will be tagged as NNP.

- **modelpath** – The path to the Komoran HMM model.
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 40).

morphs (*phrase*)
Parse phrase to morphemes.

nouns (*phrase*)
Noun extractor.

pos (*phrase*, *flatten=True*, *join=False*)
POS tagger.

Parameters

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

Mecab Class

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

class konlpy.tag._mecab.**Mecab**(*dicpath='/usr/local/lib/mecab/dic/mecab-ko-dic'*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

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.

```
>>> # MeCab installation needed
>>> from konlpy.tag import Mecab
>>> mecab = Mecab()
>>> print(mecab.morphs(u'영등포구청역에 있는 맛집 좀 알려주세요.'))
['영등포구', '청역', '에', '있', '는', '맛집', '좀', '알려', '주', '세요', '.']
>>> print(mecab.nouns(u'우리나라에는 무릎 치료를 잘하는 정형외과가 없는가!'))
['우리', '나라', '무릎', '치료', '정형외과']
>>> print(mecab.pos(u'자연주의 쇼핑몰은 어떤 곳인가?'))
[('자연', 'NNG'), ('주', 'NNG'), ('의', 'JKG'), ('쇼핑몰', 'NNG'), ('은', 'JX'), (
↪'어떤', 'MM'), ('곳', 'NNG'), ('인가', 'VCP+EF'), ('?', 'SF')]
```

Parameters **dicpath** – The path of the MeCab-ko dictionary.

morphs (*phrase*)
Parse phrase to morphemes.

nouns (*phrase*)
Noun extractor.

pos (*phrase*, *flatten=True*, *join=False*)
POS tagger.

Parameters

- **flatten** – If False, preserves eojeols.
- **join** – If True, returns joined sets of morph and tag.

Okt Class

Warning: `Twitter()` has changed to `Okt()` since v0.5.0.

class konlpy.tag._okt.**Okt** (*jvmpath=None*, *max_heap_size=1024*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

Wrapper for Open Korean Text (https://github.com/open-korean-text/open-korean-text).

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

```
>>> from konlpy.tag import Okt
>>> okt = Okt()
>>> print(okt.morphs(u'단독입찰보다 복수입찰의 경우'))
['단독', '입찰', '보다', '복수', '입찰', '의', '경우']
>>> print(okt.nouns(u'유일하게 항공기 체계 종합개발 경험을 갖고 있는 KAI는'))
['항공기', '체계', '종합', '개발', '경험']
>>> print(okt.phrases(u'날카로운 분석과 신뢰감 있는 진행으로'))
['날카로운 분석', '날카로운 분석과 신뢰감', '날카로운 분석과 신뢰감 있는 진행', '분석', '신뢰
↪', '진행']
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ'))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나욬', 'Noun'), ('ㅋㅋ
↪', 'KoreanParticle')]
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ', norm=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나요', 'Verb'), ('ㅋㅋ
↪', 'KoreanParticle')]
>>> print(okt.pos(u'이것도 되나욬ㅋㅋ', 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 40).
- **max_heap_size** – Maximum memory usage limitation (Megabyte) *init_jvm()* (page 40).

morphs (*phrase*, *norm=False*, *stem=False*)
Parse phrase to morphemes.

normalize (*phrase*)

nouns (*phrase*)
Noun extractor.

phrases (*phrase*)
Phrase extractor.

pos (*phrase*, *norm=False*, *stem=False*, *join=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.
- **join** – If True, returns joined sets of morph and tag.

konlpy.tag._okt.**Twitter**(*jvmpath=None*)
The `Twitter()` backend has changed to `Okt()` since KoNLPy v0.5.0. See #141 (https://github.com/konlpy/konlpy/issues/141) for details.

See also:

Korean POS tags comparison chart (https://docs.google.com/spreadsheets/d/1OGAjUvalBuX-oZvZ_-9tEfYD2gQe7hTGsgUpiiBSXI8/edit#gid=0)

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

corpus Package

class konlpy.corpus.**CorpusLoader**(*name=None*)
Bases: `object` (https://docs.python.org/3/library/functions.html#object)

Loader for corpora. For a complete list of corpora available in KoNLPy, refer to *Corpora* (page 31).

```
>>> from konlpy.corpus import kolaw
>>> fids = kolaw.fileids()
>>> fobj = kolaw.open(fids[0])
>>> print fobj.read(140)
대한민국헌법

유구한 역사와 전통에 빛나는 우리 대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
항거한 4·19민주이념을 계승하고, 조국의 민주개혁과 평화적 통일의 사명에 입각하여 정의·인도와 동포
애로써 민족의 단결을 공고히 하고, 모든 사회적 폐습과 불의를 타파하며, 자율과 조화를 바 바
```

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.

6.1.2 data Module

konlpy.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/3/library/stdtypes.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.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* (https://docs.python.org/3/library/stdtypes.html#str)) – A URL specifying where the resource should be loaded from.
- **format** – Format type of resource.

konlpy.data.**listdir**()
list konlpy default data directory.

```
>>> import konlpy
>>> konlpy.listdir()
```

konlpy.data.**clear**()
clear the konlpy output data directory

```
>>> import konlpy
>>> konlpy.clear()
```

konlpy.data.**path = ['/home/docs/konlpy_data', '/usr/share/konlpy_data', '/usr/local/shar**
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.

class konlpy.data.**FileSystemPathPointer**(*path*)
Bases: *konlpy.data.PathPointer* (page 39), str (https://docs.python.org/3/library/stdtypes.html#str)

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

file_size()

open(*encoding='utf-8'*)

class konlpy.data.**PathPointer**
Bases: object (https://docs.python.org/3/library/functions.html#object)

An abstract base class for path pointers. One subclass exists: 1. FileSystemPathPointer: Identifies a file by an absolute path.

file_size()

open(*encoding='utf-8'*)

class konlpy.data.**CorpusReader**(*extension='.txt'*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

read()
read method reads all files included in items attr and save it into corpus dictionary.

class konlpy.data.**StringWriter**(*filename*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

write(*string*)

6.1.3 downloader Module

class konlpy.downloader.**Downloader**(*download_dir=None*)
Bases: object (https://docs.python.org/3/library/functions.html#object)

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

INDEX_URL = 'http://konlpy.github.io/konlpy-data/index.json'

INSTALLED = 'installed'

NOT_INSTALLED = 'not installed'

PACKAGE_URL = 'http://konlpy.github.io/konlpy-data/packages/%s.%s'

SCRIPT_URL = 'http://konlpy.github.io/konlpy-data/packages/%s.sh'

STALE = 'corrupt or out of date'

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:

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

To download all packages, simply call `download` with the argument 'all':

```
>>> download('all')
[konlpy_data] Downloading package 'kobill'...
[konlpy_data]   Unzipping corpora/kobill.zip.
...
```

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 overriden 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`.

6.1.4 `jvm` Module

konlpy.jvm.**init_jvm**(*jvmpath=None*, *max_heap_size=1024*)

Initializes the Java virtual machine (JVM).

Parameters

- **jvmpath** – The path of the JVM. If left empty, inferred by `jpype.getDefaultJVMPath()`.
- **max_heap_size** – Maximum memory usage limitation (Megabyte). Default is 1024 (1GB). If you set this value too small, you may got out of memory. We recommend that you set it 1024 ~ 2048 or more at least. However, if this value is too large, you may see inefficient memory usage.

6.1.5 `utils` Module

class konlpy.utils.**PropagatingThread**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs=None*, ***, *daemon=None*)

Bases: `threading.Thread` (https://docs.python.org/3/library/threading.html#threading.Thread)

PropagatingThread is just a fancy wrapper for Thread to manage exceptions.

Raises: self.exception: Exception defined in higher-level.

Returns: self.ret: Thread target object.

join()
Wait until the thread terminates.

This blocks the calling thread until the thread whose join() method is called terminates – either normally or through an unhandled exception or until the optional timeout occurs.

When the timeout argument is present and not None, it should be a floating point number specifying a timeout for the operation in seconds (or fractions thereof). As join() always returns None, you must call is_alive() after join() to decide whether a timeout happened – if the thread is still alive, the join() call timed out.

When the timeout argument is not present or None, the operation will block until the thread terminates.

A thread can be join()ed many times.

join() raises a RuntimeError if an attempt is made to join the current thread as that would cause a deadlock. It is also an error to join() a thread before it has been started and attempts to do so raises the same exception.

run()
Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

konlpy.utils.**char2hex**(*c*)
Converts a unicode character to hex.

```
>>> char2hex(u'음')
'0xc74c'
```

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
>>> constitution = kolaw.open('constitution.txt').read()
>>> idx = utils.concordance(u'대한민국', constitution, show=True)
0       대한민국헌법 유구한 역사와
9       대한국민은 3·1운동으로 건립된 대한민국임시정부의 법통과 불의에
98      총강 제1조 ① 대한민국은 민주공화국이다. ②대한민국의
100     ① 대한민국은 민주공화국이다. ②대한민국의 주권은 국민에게
110     나온다. 제2조 ① 대한민국의 국민이 되는
126     의무를 진다. 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 제119조 ① 대한민국의 경제질서는 개인과
>>> idx
[0, 9, 98, 100, 110, 126, 133, 147, 787, 1836, 3620]
```

konlpy.utils.**delete_links**(*string*)
Delete links from input string

Args: string (str): string to delete links

Returns: str: string without links

konlpy.utils.**delete_mentions**(*string*)
Delete at marks from input string

Args: string (str): string to delete at marks

Returns: str: string without at marks.

konlpy.utils.**hex2char**(*h*)
Converts a hex character to unicode.

```
>>> print hex2char('c74c')
ㅎ
>>> print hex2char('0xc74c')
ㅎ
```

konlpy.utils.**load_txt**(*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.**read_json**(*filename*, *encoding='utf-8'*)
JSON file reader.

konlpy.utils.**read_txt**(*filename*, *encoding='utf-8'*)
Text file reader.

konlpy.utils.**select**(*phrase*)
Replaces some ambiguous punctuation marks to simpler ones.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search
- changelog

Python Module Index

k

konlpy.corpus, 38
konlpy.data, 38
konlpy.downloader, 39
konlpy.jvm, 40
konlpy.tag._hannanum, 33
konlpy.tag._kkma, 34
konlpy.tag._komoran, 35
konlpy.tag._mecab, 36
konlpy.tag._okt, 37
konlpy.utils, 40

Index

A

abspath() (*konlpy.corpus.CorpusLoader method*), 38
analyze() (*konlpy.tag._hannanum.Hannanum method*), 34

C

char2hex() (*in module konlpy.utils*), 41
clear() (*in module konlpy.data*), 39
concordance() (*in module konlpy.utils*), 41
CorpusLoader (*class in konlpy.corpus*), 38
CorpusReader (*class in konlpy.data*), 39

D

default_download_dir() (*in module konlpy.downloader*), 40
delete_links() (*in module konlpy.utils*), 41
delete_mentions() (*in module konlpy.utils*), 42
download() (*konlpy.downloader.Downloader method*), 40
Downloader (*class in konlpy.downloader*), 39

F

file_size() (*konlpy.data.FileSystemPathPointer method*), 39
file_size() (*konlpy.data.PathPointer method*), 39
fileids() (*konlpy.corpus.CorpusLoader method*), 38
FileSystemPathPointer (*class in konlpy.data*), 39
find() (*in module konlpy.data*), 38

H

Hannanum (*class in konlpy.tag._hannanum*), 33
hex2char() (*in module konlpy.utils*), 42

I

INDEX_URL (*konlpy.downloader.Downloader attribute*), 39
init_jvm() (*in module konlpy.jvm*), 40
INSTALLED (*konlpy.downloader.Downloader attribute*), 39

J

join() (*konlpy.utils.PropagatingThread method*), 40

K

Kkma (*class in konlpy.tag._kkma*), 34
Komoran (*class in konlpy.tag._komoran*), 35
konlpy.corpus (*module*), 38
konlpy.data (*module*), 38
konlpy.downloader (*module*), 39
konlpy.jvm (*module*), 40
konlpy.tag._hannanum (*module*), 33
konlpy.tag._kkma (*module*), 34
konlpy.tag._komoran (*module*), 35
konlpy.tag._mecab (*module*), 36
konlpy.tag._okt (*module*), 37
konlpy.utils (*module*), 40

L

listdir() (*in module konlpy.data*), 39
load() (*in module konlpy.data*), 38
load_txt() (*in module konlpy.utils*), 42

M

Mecab (*class in konlpy.tag._mecab*), 36
morphs() (*konlpy.tag._hannanum.Hannanum method*), 34
morphs() (*konlpy.tag._kkma.Kkma method*), 35
morphs() (*konlpy.tag._komoran.Komoran method*), 36
morphs() (*konlpy.tag._mecab.Mecab method*), 36
morphs() (*konlpy.tag._okt.Okt method*), 37

N

normalize() (*konlpy.tag._okt.Okt method*), 37
NOT_INSTALLED (*konlpy.downloader.Downloader attribute*), 39
nouns() (*konlpy.tag._hannanum.Hannanum method*), 34
nouns() (*konlpy.tag._kkma.Kkma method*), 35
nouns() (*konlpy.tag._komoran.Komoran method*), 36
nouns() (*konlpy.tag._mecab.Mecab method*), 36
nouns() (*konlpy.tag._okt.Okt method*), 37

O

Okt (*class in konlpy.tag._okt*), 37
open() (*konlpy.corpus.CorpusLoader method*), 38
open() (*konlpy.data.FileSystemPathPointer method*), 39
open() (*konlpy.data.PathPointer method*), 39

P

PACKAGE_URL (*konlpy.downloader.Downloader attribute*), 39
partition() (*in module konlpy.utils*), 42
path (*in module konlpy.data*), 39
PathPointer (*class in konlpy.data*), 39
phrases() (*konlpy.tag._okt.Okt method*), 37
pos() (*konlpy.tag._hannanum.Hannanum method*), 34
pos() (*konlpy.tag._kkma.Kkma method*), 35
pos() (*konlpy.tag._komoran.Komoran method*), 36
pos() (*konlpy.tag._mecab.Mecab method*), 36
pos() (*konlpy.tag._okt.Okt method*), 37
PropagatingThread (*class in konlpy.utils*), 40

R

read() (*konlpy.data.CorpusReader method*), 39
read_json() (*in module konlpy.utils*), 42
read_txt() (*in module konlpy.utils*), 42
run() (*konlpy.utils.PropagatingThread method*), 41

S

SCRIPT_URL (*konlpy.downloader.Downloader attribute*), 39
select() (*in module konlpy.utils*), 42
sentences() (*konlpy.tag._kkma.Kkma method*), 35
STALE (*konlpy.downloader.Downloader attribute*), 40
status() (*konlpy.downloader.Downloader method*), 40
StringWriter (*class in konlpy.data*), 39

T

Twitter() (*in module konlpy.tag._okt*), 38

W

write() (*konlpy.data.StringWriter method*), 39