Pengenalan NLP(*Natural Language Processing*) dengan Python

Kholid Fuadi

December 1, 2013

Contents

1 Pendahuluan

Saya bukan sarjana ilmu komputer dan juga bukan sarjana linguistik, tapi saya sangat tertarik untuk mempelajari NLP, dengan dasar masih jarangnya pembahasan (atau bahkan penelitian) tentang tema ini di Indonesia. Selama ini artikel yang paling banyak saya temui adalah penggunaan NLP untuk analisis teks bahasa Inggris. Untuk itu kiranya penting untuk mulai [lebih] dikenalkan kepada publik, supaya tema ini semakin berkembang dan bermanfaat baik bagi lingkungan akademisi maupun awam seperti saya ini.

Sesuai judul, artikel ini berisi dasar-dasar penggunaan NLP untuk analisis teks, khususnya teks berbahasa Indonesia. Semoga bermanfaat!

Kholid Fuadi
Jogja, 27 November 2013.

1.1 NLP

Tujuan dari artikel ini adalah sebagai pengenalan terhadap *Natural Language Processing* (Pemrosesan Bahasa Alami / PBA), yang sering disebut dengan *NLP*, tidak hanya sebatas konsep, namun juga sisi praktis dengan menggunakan bahasa pemrograman Python agar dapat memberikan gambaran secara lebih jelas. Pembaca diharapkan memiliki dasar Python yang cukup atau jika belum dapat terlebih dulu mempelajari dasar-dasar Python yang banyak bertebaran di internet.

Istilah *Natural Language Processing* sendiri mengacu pada sejumlah teknik untuk *automated generation*, manipulasi dan analisis terhadap bahasa natural (bahasa manusia). Meski sebagian besar teknik NLP diturunkan dari tema *Linguistics* dan *Artificial Intelligence*, teknik ini juga dipengaruhi oleh *Machine Learning, Computational Statistics* dan *Cognitive Science*.

Sebelumnya, mari kita kenal dulu beberapa istilah yang sering digunakan dalam *NLP*, diantaranya:

- **Token:** Sebelum proses dijalankan, teks perlu dikelompokkan menjadi unit-unit *linguistic* seperti kata, tanda petik, angka atau *alphanumerics*. Unit-unit kecil ini dinamakan dengan *token*.
- **Sentence:** Sekumpulan token-token yang berurutan, dalam bahasa Indonesia disebut sebagai kalimat.
- **Tokenization:** Proses memecah kalimat (*sentence*) menjadi token-token.
- **Corpus:** *Corpus* didefinisikan sebagai kumpulan teks dalam jumlah yang sangat besar, bisa berbentuk mentah atau terkelompok, bisa fokus ke satu tema atau terdiri dari berbagai tema. Contoh: Brown Corpus, merupakan corpus pertama, terbesar dan sudah dikelompokkan berdasar genre. Webtext corpus, misal reviews, forums, dsbnya. Reuters, merupakan corpus

berita, Inaugural corpus, yang berisi pidato inaugural presiden Amerika, dan udhr corpus, corpus multilingual.

Apa kegunaan dari corpus?:

- to create linguistic resources (lexicon, annotated corpora, etc)
- to train statistical tools (taggers, parsers, etc.
- to evalute existing tools (taggers, parsers, etc.)

- **Part-of-speech (POS) Tag:** Sebuah kata dapat dikelompokkan ke dalam satu atau lebih kumpulan *lexical* atau kategori *part-of-speech* seperti *Nouns, Verbs , Adjectives, Articles. POS tag* adalah simbol yang mewakili kategori leksikal - NN (Noun), VB (Verb), JJ (Adjective), dan AT (Article). Salah satu POS tag yang paling tua dan sering dipakai adalah *Brown Corpus*. Kita akan membahas *corpus* ini di bagian lain.

- **Parse Tree:** *Tree* menggambarkan *syntactic structure* sebuah kalimat yang tersusun dari struktur grammar formal (resmi).

Setelah mengenal beberapa terminologi, selanjutnya kita akan melihat beberapa tugas-tugas yang berkaitan dengan NLP:

- **POS Tagging:** Salah satu tugas dari NLP adalah POS Tagging, yakni memberikan POS tags secara otomatis pada setiap kata dalam satu atau lebih kalimat. Sebagai contoh, ada kalimat sebagai berikut:

  The ball is red

  Jika kita lakukan proses POS Tagging, maka NLP akan memberikan keluaran sebagai berikut:

  The/AT ball/NN is/VB red/JJ

  Akurasi dari POS tagger dapat mencapai 96% (untuk bahasa Inggris). Proses *tagging* sebuah kalimat menggunakan *part-of-speech* sangat berguna untuk tugas-tugas NLP yang lebih rumit, seperti *parsing* dan *machine translation*, yang akan dibahas dibagian berikut ini.

- **Computational Morphology:** Bahasa natural (bahasa manusia) terdiri dari sejumlah kata dalam jumlah besar yang dibentuk dari sebuah *building blocks* yang disebut dengan *morphemes* atau *stems*, yakni bagian terkecil dari unit lingustik yang memiliki arti (makna). *Computational Morphology* memfokuskan pada penemuan (*discovery* dan analisis terhadap struktur internal dari kata menggunakan bantuan komputer.

- **Parsing:** Berkaitan dengan tugas ini, sebuah *parser* membentuk *parse tree* dari kalimat. Beberapa parser menggunakan asumsi aturan grammar, namun parser saat ini sudah semakin pintar melakukan tugas parsing menggunakan model statistik yang kompleks. Sebagian besar parser,

menggunakan aturan-aturan tertentu dan mensyaratkan kalimat harus sudah melalui proses *POS-tagging* sebelum dapat dilakukan parsing. *Statistical parsing* merupakan salah satu bidang penelitian yang aktif dalam tema NLP.

- **Machine Translation (MT):** Tujuan dari tugas ini adalah mengalihbahasakan teks ke dalam bahasa lain tanpa campur tangan manusia. Tugas ini merupakan salah satu tugas tersulit dari NLP dan agar dapat berjalan dengan sempurna, teknik ini terus-menerus dikembangkan. Hampir semua proses MT, didahului dengan proses POS tagging dan parsing.

1.2 Python

Python merupakan bahasa pemrograman yang relatif mudah dipelajari, dan memiliki banyak *library* standar yang berguna untuk membangun aplikasi secara cepat baik dalam skala kecil maupun besar. Banyak sekali literatur dan materi-materi yang disediakan secara gratis bagi Anda yang ingin mempelajari Python.

Berikut ini contoh pemrosesan teks sederhana menggunakan pustaka bawaan dari Python:

```
>>> nama = 'Kholid Fuadi'
>>> nama + ' sedang belajar Python NLTK'
'Kholid Fuadi sedang belajar Python NLTK'
>>> nama.find('Python') # tidak ketemu, maka hasil -1
-1
>>> nama.find('Fuadi') # Fuadi berada pada indeks nomor 7
7
>>> nama.upper() + ' and ' + nama.lower()
'KHOLID FUADI and kholid fuadi'
>>> nama.replace('i', 'y')
'Kholyd Fuady'
```

Selain string object, teks juga dapat dimasukkan kedalam `list`, kenapa `list`? Karena list lebih fleksibel terhadap tiap elemen yang ada didalamnya.

Perhatikan contoh berikut:

```
>>> kalimat1 = ['Monty', 'Python']
>>> kalimat2 = ['and', 'the', 'Holy', 'Grail']
>>> len(kalimat2)
4
>>> kalimat1[1]
'Python'
>>> kalimat2.append('1975')
>>> kalimat1 + kalimat2
['Monty', 'Python', 'and', 'the', 'Holy', 'Grail', '1975']
>>> sorted(kalimat1 + kalimat2)
['1975', 'Grail', 'Holy', 'Monty', 'Python', 'and', 'the']
>>> ' '.join(['Monty', 'Python'])
'Monty Python'
>>> 'Monty Python'.split()
['Monty', 'Python']
```

Namun tetap saja, pustaka bawaan ini tidak cukup *powerful* untuk melakukan keperluan analisis teks tingkat lanjut, untuk itu diperlukan pustaka luar yang bernama NLTK, yang akan dibahas berikut ini.

1.3 NLTK

Meski Python memiliki kemampuan untuk melakukan tugas-tugas NLP dasar, namun tidak cukup *powerful* untuk melakukan tugas-tugas standar NLP, maka dari itu muncullah modul `NLTK` (Natural Language Toolkit). Modul ini menyediakan berbagai fungsi dan *wrapper*, serta corpora standar baik itu mentah atau pun *pre-processed* yang digunakan dalam materi pengajaran NLP.

2 Menggunakan NLTK

Dokumentasi tentang NLTK secara lengkap dapat Anda baca di situs resmi NLTK. Dalam artikel ini nantinya akan dipelajari empat tugas NLP yang dikerjakan menggunakan pustaka NLTK.

2.1 NLTK Corpora

Seperti telah disebutkan di atas, pustaka NLTK menyediakan beberapa corpora teks yang sering digunakan dalam komunitas penelitian NLP. Berikut ini, tiga corpora yang akan kita gunakan sebagai materi latihan dalam artikel ini.

- **Brown Corpus:** The Brown Corpus of Standard American English dinilai sebagai corpus English pertama yang dapat dipakai dalam tugas pemrosesan *computational linguistic*. Corpus ini terdiri dari sekitar sejuta kata dalam bahasa Inggris Amerika yang dicetak pada tahun 1961. Agar corpus ini dapat mewakili bahasa Inggris seoptimal mungkin, corpus ini dibagi kedalam 15 genre tulisan, misalnya fiksi, berita dan teks agama. Berikutnya versi POS-tagged dari corpus ini juga dibuat menggunakan teknik-teknik manual.

- **Gutenberg Corpus:** The Gutenberg Corpus terdiri dari 14 teks pilihan dari *Project Gutenberg* - koleksi ebook gratis terbesar. Corpus ini terdiri dari 1,7 juta kata.

- **Treebank Corpus:** Kumpulan teks yang sudah di-*parsing* dari Penn treebank.

- **StopwordsCorpus:** Selain kata-kata umum, ada juga kelompok kata yang disebut dengan *stop words* yang memiliki posisi penting dalam grammar namun tidak bisa berdiri sendiri, seperti *prepositions, complementizers*, dan *determiners*. Pustaka NLTK menyediakan Stopword Corpus yang terdiri dari 2400 kata dalam 11 bahasa berbeda (termasuk bahasa Inggris).

2.2 Dasar-dasar Operasi NLTK

2.2.1 Melihat Daftar Corpus Bawaan

Untuk melihat daftar berkas pada corpus bawaan (misal corpus gutenberg), lakukan perintah berikut:

```
>>> import nltk
>>> nltk.corpus.gutenberg.items
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt',
'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
```

Jika Anda pengguna Linux (Ubuntu), Anda dapat melihat pada direktori `nltk_data` yang berada pada direktori `home` Anda, nanti di sana ada berkas corpus/gutenberg yang isinya berkas-berkas di atas. Dan jika Anda ingin melihat lebih jauh ke dalam berkas `txt` tersebut, maka isinya tidak lain adalah *plain text* biasa.[1]

2.2.2 Mencetak Kata yang terdapat pada Corpus

Jalankan perintah berikut:

```
>>> nltk.corpus.brown.words()
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
>>> for w in nltk.corpus.brown.words():
...     print w
...
```

2.3 Perbedaan Corpus Brown dengan Gutenberg

Corpus Gutenberg tokenized dari urutan kata-kata, tanpa struktur. Sementara itu corpus brown terdiri dari kalimat yang sudah ditandai. Untuk lebih jelasnya, silakan buka salah satu berkas pada corpus gutenberg dan brown, dan perhatikan perbedaannya.

Anda dapat mengekstrak daftar kata pada corpus brown dengan cara:

```
>>> nltk.corpus.brown.sents()
[['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', 'Friday', 'an',
'investigation', 'of', "Atlanta's", 'recent', 'primary', 'election',
'produced', '``', 'no', 'evidence', "''", 'that', 'any', 'irregularities',
'took', 'place', '.'], ['The', 'jury', 'further', 'said', 'in', 'term-end',
'presentments', 'that', 'the', 'City', 'Executive', 'Committee', ',',
'which', 'had', 'over-all', 'charge', 'of', 'the', 'election', ',', '``',
'deserves', 'the', 'praise', 'and', 'thanks', 'of', 'the', 'City', 'of',
'Atlanta', "''", 'for', 'the', 'manner', 'in', 'which', 'the', 'election',
'was', 'conducted', '.'], ...]
```

Anda dapat juga mengekstrak POS tag pada corpus brown dengan cara:

[1] Pengguna Linux, gunakan perintah head nama_berkas.txt

```
>>> nltk.corpus.brown.tagged_words()
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

Sedangkan pada corpus Penn treebank, Anda dapat mengakses teks yang sudah di-*parsing* dengan cara:

```
>>> for t in nltk.corpus.treebank.parsed_sents():
...     print t
...
```

Atau jika cuma ingin melihat kalimat pertama saja:

```
>>> for t in nltk.corpus.treebank.parsed_sents()[0]:
...     print t
...
(NP-SBJ
  (NP (NNP Pierre) (NNP Vinken))
  (, ,)
  (ADJP (NP (CD 61) (NNS years)) (JJ old))
  (, ,))
(VP
  (MD will)
  (VP
    (VB join)
    (NP (DT the) (NN board))
    (PP-CLR (IN as) (NP (DT a) (JJ nonexecutive) (NN director)))
    (NP-TMP (NNP Nov.) (CD 29))))
(. .)
```

2.3.1 Menghitung Jumlah Kata pada Corpus Gutenberg

Jalankan perintah berikut:

```
>>> for book in nltk.corpus.gutenberg.items:
...     print book + ':', len(nltk.corpus.gutenberg.words(book))
...
austen-emma.txt: 192427
austen-persuasion.txt: 98171
austen-sense.txt: 141576
bible-kjv.txt: 1010654
blake-poems.txt: 8354
bryant-stories.txt: 55563
burgess-busterbrown.txt: 18963
carroll-alice.txt: 34110
chesterton-ball.txt: 96996
chesterton-brown.txt: 86063
chesterton-thursday.txt: 69213
edgeworth-parents.txt: 210663
melville-moby_dick.txt: 260819
milton-paradise.txt: 96825
shakespeare-caesar.txt: 25833
shakespeare-hamlet.txt: 37360
shakespeare-macbeth.txt: 23140
whitman-leaves.txt: 154883
>>>
```

2.3.2 Menghitung Frekuensi Kemunculan Kata pada Corpora

Jalankan sintaks berikut:

```
>>> from collections import Counter
>>> cnt = Counter()
>>> for word in nltk.corpus.gutenberg.words('shakespeare-macbeth.txt'):
...     word = word.lower()
...     cnt[word] += 1
...
>>> cnt['scotland']
12
>>>
```

Atau bisa juga:

```
>>> for word in nltk.corpus.gutenberg.words('shakespeare-macbeth.txt'):
...     word = word.lower()
...     if word not in count:
...         count[word] = 0
...     count[word] += 1
...
>>> count['scotland']
12
>>> count['thane']
25
>>> count['blood']
24
>>> count['duncan']
10
>>>
```

2.3.3 Mengurutkan Frekuensi Kemunculan Kata

Dengan menggunakan `collections` di atas, kita dapat dengan mudah mencari misalnya 20 kata yang paling sering muncul pada corpus gutenberg.

```
>>> cnt.most_common()[:20]
[(',', 1962), ('.', 1235), ('the', 650), ("'", 637), ('and', 546),
(':', 477), ('to', 384), ('i', 348), ('of', 338), ('a', 241), ('?', 241),
('that', 238), ('d', 224), ('you', 206), ('my', 203), ('in', 201),
('is', 188), ('not', 165), ('it', 161), ('with', 153)]
```

Atau kita bisa juga menggunakan fungsi bawaan pustaka NLTK `FreqDist`:

```
>>> sec_a = nltk.corpus.gutenberg.words()
>>> fd = nltk.FreqDist(sec_a)
>>> from operator import itemgetter
>>> swc = sorted(fd.items(), key=itemgetter(1), reverse=True)
>>> [token for (token, freq) in swc[:20]]
[',', 'the', 'and', '.', 'of', ':', 'to', 'a', 'in', 'I', ';', 'that',
'he', 'his', "'", 'it', 'was', 'for', 'not', 'with']
>>>
```

Ada sedikit perbedaan hasil, karena pada cara kedua, kita tidak membedakan huruf besar dan huruf kecil. Dari kedua cara ini, menurut penulis cara pertama (pustaka `collections`) lebih cepat dan efisien.

2.3.4 Tokenizing

Tokenizing adalah proses memecah teks menjadi *words* atau *tokens*. Hal ini mutlak dilakukan agar teks dapat diproses lebih lanjut, misalnya untuk melakukan *parsing* struktur *grammar*.

Cara termudah melakukan *tokenizing* adalah dengan memecah teks berdasar penanda spasi. Perhatikan contoh berikut:

```
>>> s = "Nama saya Kholid Fuadi"
>>> words = s.split()
>>> words
['Nama', 'saya', 'Kholid', 'Fuadi']
```

Objek `words` selanjutnya dapat disebut sebagai kumpulan *token*. *Token* sendiri terdiri dari bermacam jenis, bisa berbentuk *words*, *lines*, *sentences*, atau bahkan *paragraph*.

2.3.5 Lemmatisation, Stemming dan Normalisation

Lemmatization dan stemming merupakan bagian dari proses normalisasi. Kedua proses ini mencoba mengenali bentuk asli sebuah kata. Secara khusus, *Lemmatisation* didefinisikan sebagai proses mengembalikan kata kedalam bentuk lemma, misal appeared `->` appear. Proses ini menggunakan sejumlah aturan untuk pola kata umum, dan *table look-up* untuk pola kata yang tidak lazim. Sedangkan *stemming* kurang lebih sama dengan proses *lematisasi* namun tanpa disertai dengan verifikasi apakah stem yang dihasilkan sudah sesuai dengan lemma-nya.

Perhatikan contoh berikut:

```
>>> stemmer = nltk.PorterStemmer()
>>> verbs = ['appears', 'appear', 'appeared', 'calling', 'called']
>>> stems = []
>>> for verb in verbs:
...     stemmed_verb = stemmer.stem(verb)
...     stems.append(stemmed_verb)
...
>>> stems
['appear', 'appear', 'appear', 'call', 'call']
>>> set(stems)
set(['call', 'appear'])
>>> sorted(set(stems))
['appear', 'call']
>>>
```

2.4 Aturan Penamaan di NLTK

Sebelum kita melanjutkan penggunaan NLTK dalam tugas kita, penting untuk diketahui aturan penamaan (*naming conventions*) yang digunakan dalam pustaka NLTK. Tingkat teratas dari pustaka ini disebut dengan `nltk`, dan kita dapat memanggil pustaka dibawahnya dengan menggunakan tanda titik, misalnya `ntlk.corpus` dan `nltk.utilities`. Atau kita juga dapat menggunakan teknik `from ... import ...` di Python.

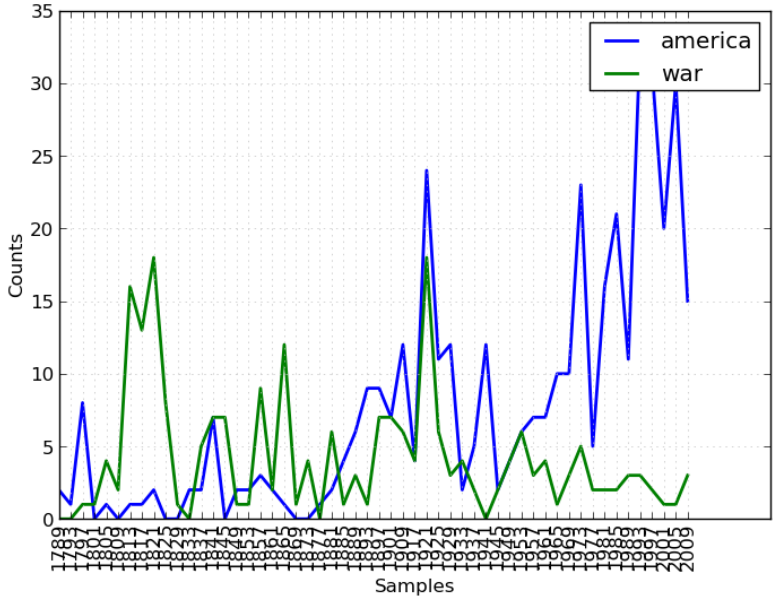
Jadi tunggu apa lagi? Nyalakan Python Interpreter Anda dan mari kita mulai eksplorasi!

```
>>> # import the gutenberg collection
>>> from nltk.corpus import gutenberg
>>> # what corpora are in the collection
>>> print gutenberg.fileids()
['austen-emma.txt', 'austen-persuasion.txt', 'austen-sense.txt',
'bible-kjv.txt', 'blake-poems.txt', 'bryant-stories.txt',
'burgess-busterbrown.txt', 'carroll-alice.txt', 'chesterton-ball.txt',
'chesterton-brown.txt', 'chesterton-thursday.txt', 'edgeworth-parents.txt',
'melville-moby_dick.txt', 'milton-paradise.txt', 'shakespeare-caesar.txt',
'shakespeare-hamlet.txt', 'shakespeare-macbeth.txt', 'whitman-leaves.txt']
>>> # import FreqDist class
>>> from nltk.probability import FreqDist
>>> # create frequency distribution object
>>> fd = FreqDist()
>>> # for each token in the relevant text, increment its counter
>>> for word in gutenberg.words('austen-persuasion.txt'):
...     fd.inc(word)
...
>>> print fd.N() # total number of samples
98171
>>> print fd.B() # numver of bins or unique samples
6132
>>> # get a list of the top 10 words sorted by frequency
>>> for word in fd.keys()[:10]:
...     print word, fd[word]
...
, 6750
the 3120
to 2775
. 2741
and 2739
of 2564
a 1529
in 1346
was 1330
; 1290
>>>
```

Contoh lain, mengetahui distribusi frekuensi kata `america` dan `war` pada Inaugural corpus.

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()[:2]
['1789-Washington.txt', '1793-Washington.txt']
>>> import nltk
>>> cfd = nltk.ConditionalFreqDist(
... (target, fileid[:4])
... for fileid in inaugural.fileids()
... for w in inaugural.words(fileid)
... for target in ['america', 'war']
... if w.lower().startswith(target))
>>> cfd.plot()
```

Hasilnya:



Contoh berikutnya, mengetahui seberapa sering sebuah kata muncul pada corpus dengan genre yang berbeda-beda:

```
>>> from nltk.corpus import brown
>>> from nltk import FreqDist
>>> verbs = ['should', 'may', 'can']
>>> genres = ['news', 'government', 'romance']
>>> for g in genres:
...     words = brown.words(categories=g)
...     freq = FreqDist([w.lower() for w in words if w.lower() in verbs])
...     print g, freq
...
news <FreqDist: 'can': 94, 'may': 93, 'should': 61>
government <FreqDist: 'may': 179, 'can': 119, 'should': 113>
romance <FreqDist: 'can': 79, 'should': 33, 'may': 11>
```

Dari hasil terlihat bahwa pada genre news, kata can paling sering muncul, sedangkan kata may paling sering muncul pada genre government, dan pada genre romance kata yang sering muncul adalah can.

Contoh berikutnya, kita akan gunakan corpus Wordnet yang berisi kamus besar dalam bahasa Inggris, di mana kita dapat mendefinisikan sinonim, antonim, hypernim, hyponims dan depth of a synset[2].

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
```

[2]kumpulan satu atau lebih sinonim

```
>>> wn.synset('car.n.01').lemma_names
['car', 'auto', 'automobile', 'machine', 'motorcar']
>>> wn.synset('car.n.01').definition
'a motor vehicle with four wheels; usually propelled by an \
internal combustion engine'
>>> for synset in wn.synsets('car')[1:3]:
...     print synset.lemma_names
...
['car', 'railcar', 'railway_car', 'railroad_car']
['car', 'gondola']
>>> wn.synset('walk.v.01').entailments() # walking involves stepping
[Synset('step.v.01')]
>>>
```

Pada contoh di atas kita menggunakan kata *motorcar*. Dari *corpus wordnet*, kita dapat mengetahui bahwa kata *motorcar* memiliki sinonim dengan kata *car, auto, automobile, machine*, dan *motorcar*. Dalam NLP, sinonim didefinisikan sebagai kata yang berada dalam kategori yang sama.

Perintah selanjutnya adalah untuk mengetahui definisi dari suatu kata, dalam contoh di atas kata *car* dengan memanggil atribut `definition`. Perintah selanjutnya adalah mengetahui subkategori dari kata *car*, menggunakan atribut `lemma_names`.

Perintah terakhir, fungsi `entailments` adalah untuk mengetahui kata kerja lain yang mungkin terlibat dalam sebuah kata kerja. Bagaimana NLP bisa mengetahui hal ini? Karena kata *stepping* merupakan cabang dari kata *walking* (`Tree`).

3 TextBlob

`TextBlob`[3] merupakan pustaka Python yang berguna untuk pemrosesan data tekstual. Pustaka ini memiliki API yang sederhana dan konsisten ketika digunakan untuk tugas-tugas NLP seperti *POS tagging, noun phrase extraction, sentiment analysis, classification, translation*, dan masih banyak lagi. `TextBlob` dikembangkan dari pustaka `NLTK`[4] dan `pattern`.[5]

Contoh sederhana penggunaan pustaka `TextBlob`:

```
>>> from textblob import TextBlob
>>> b = TextBlob("Saya sedang belajar Python")
>>> b.detect_language()
ub'id'
>>> b.words
WordList([u'Saya', u'sedang', u'belajar', u'Python'])
>>> b.sentences
[Sentence("Saya sedang belajar Python")]
>>> b.translate(to="en")
TextBlob("I 'm learning Python")
```

Keren ya ... :)

[3] http://textblob.readthedocs.org/en/latest/index.html
[4] http://www.nltk.org
[5] http://www.clips.ua.ac.be/pages/pattern-en

4 Konsep Dasar NLP

4.1 Naive Bayes Classifier

Berikut ini kutipan dari Wikipedia[6] tentang naive bayes classifier:

> A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be "independent feature model". An overview of statistical classifiers is given in the article on Pattern recognition.

5 Reference

http://www.umiacs.umd.edu/~jimmylin/CMSC723-2009-Fall/readings/crossroads.pdf

http://www.youtube.com/watch?v=AOU-Yw1qdJs

http://cgi.cse.unsw.edu.au/~handbookofnlp/index.php?n=Chapter7.Chapter7

http://www.loria.fr/~gardent/applicationsTAL/slides/l4-nltk-corpora-tokenization-2x2.pdf

[6] https://en.wikipedia.org/wiki/Naive_Bayes_classifier