Word Embedding / Text Processing
Practice with Python

2018. 5. 11.

Lee, Gyeongbok

DI-LAB
Data Intelligence Laboratory

Contents

- Word Embedding
  - Libraries: gensim, fastText
  - Embedding alignment (with two languages)

- Text/Language Processing
  - POS Tagging with NLTK/koNLPy
  - Text similarity (jellyfish)

Gensim

- Open-source vector space modeling and topic modeling toolkit implemented in Python
  - designed to handle large text collections, using data streaming and efficient incremental algorithms
  - Usually used to make word vector from corpus

- Tutorial is available here:
  - https://github.com/RaRe-Technologies/gensim/blob/develop/tutorials.md#tutorials
  - https://rare-technologies.com/word2vec-tutorial/

- Install
  - `pip install gensim`

Gensim for Word Embedding

- Logging

```python
import logging
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)
```

- Input Data: list of word's list
  - Example: I have a car , I like the cat

    → [["I", "have", "a", "car"], ["I", "like", "the", "cat"]]
  - For list of the sentences, you can make this by:

```python
>>> sents = ["I have a car", "I like the cat"]
>>> sents_lw = [v.split(" ") for v in sents]
>>> print(sents_lw)
[['I', 'have', 'a', 'car'], ['I', 'like', 'the', 'cat']]
```

Gensim for Word Embedding

- If your data is already preprocessed…
  - One sentence per line, separated by whitespace
    → LineSentence (just load the file)

class `gensim.models.word2vec.LineSentence`(*source*, *max_sentence_length=10000*, *limit=None*)

Bases: `object`

Simple format: one sentence = one line; words already preprocessed and separated by whitespace.

source can be either a string or a file object. Clip the file to the first *limit* lines (or not clipped if limit is None, the default).

Example:

```
sentences = LineSentence('myfile.txt')
```

Or for compressed files:

```
sentences = LineSentence('compressed_text.txt.bz2')
sentences = LineSentence('compressed_text.txt.gz')
```

  - Try with this:
    - http://an.yonsei.ac.kr/corpus/example_corpus.txt

From https://radimrehurek.com/gensim/models/word2vec.html

Gensim for Word Embedding

- If the input is in multiple files or file size is large:
  - Use custom iterator and `yield`

```
class MySentences(object):
    def __init__(self, dirname):
        self.dirname = dirname

    def __iter__(self):
        for fname in os.listdir(self.dirname):
            for line in open(os.path.join(self.dirname, fname)):
                yield line.split()

sentences = MySentences('/some/directory') # a memory-friendly iterator
model = gensim.models.Word2Vec(sentences)
```

From https://rare-technologies.com/word2vec-tutorial/

Gensim for Word Embedding

- `gensim.models.Word2Vec` Parameters

class `gensim.models.word2vec.Word2Vec`(*sentences=None, size=100, alpha=0.025, window=5, min_count=5, max_vocab_size=None, sample=0.001, seed=1, workers=3, min_alpha=0.0001, sg=0, hs=0, negative=5, cbow_mean=1, hashfxn=<built-in function hash>, iter=5, null_word=0, trim_rule=None, sorted_vocab=1, batch_words=10000, compute_loss=False, callbacks=()*)

  - `min_count`: ignore if word appears <=N times in the corpus
  - `window`: size of window (2n+1)

  - `size`: dimension of vector
  - `iter`: how many iteration for training
  - `sg`: CBOW(0) or Skip-gram(1)
    - CBOW is fast, Skip-gram usually give better result in many tasks

Full parameter list:
https://radimrehurek.com/gensim/models/word2vec.html#gensim.models.word2vec.Word2Vec

Gensim for Word Embedding

- Save as trained model
  - `model.save(filename)`

```
word2vec_test.model
word2vec_test.model.syn1neg.npy
word2vec_test.model.wv.syn0.npy
```

- Load trained model
  - `model = Word2Vec.load(filename)`
  - You can re-train this model (example: adding data)

Gensim for Word Embedding

- Save as vector file (not model)
  - ```
    word_vectors = model.wv
    word_vectors.save(filename)
    ```
  - Vector is saved as `KeyedVector` format
    ```
    word2vec_test.vector
    word2vec_test.vector.syn0.npy
    ```
- Load vector file
  - `word_vectors = KeyedVectors.load(fname)`
  - Note: this embedding cannot be re-trained!
    ```

Gensim for Word Embedding

- Save as (readable) text format
  - `word_vectors = KeyedVectors.load(vector_fn)`
    `word_vectors.save_word2vec_format(fname=filename)`

```
2521005 150
. 0.516648 0.687496 -1.237006 1.066626 0.362559 -0.608413 5.050340 1.255755 1.4893
82 1.206372 1.130537 -2.700337 0.643328 -1.141331 -0.051469 -1.906821 0.619856 -2.
1.171760 1.035450 3.101179 -1.478003 0.957970 0.935613 1.769591 -3.924809 2.420725
3 2.137557 2.018615 -1.068125 2.494286 1.570803 0.780557 1.008443 -0.715443 0.3561
25 -1.185320 -1.433047 1.631851 -2.264135 -0.283224 0.711403 2.483921 1.350167 4.8
2.141216 3.073809 -0.473251 -1.451827 -1.655692 -0.860397 0.168624 -0.306036 -1.16
205954 -0.233533 -3.188938 -2.754726 -3.693871 -0.495636 -1.393157 -0.899494 -2.15
95227 -0.538314 -1.932268 1.600755 -0.598139 2.595804 -1.622842 2.254374 -1.612787
 2.407951 2.500426 -0.772812 0.727460 0.003763 -0.198852 -0.808186 -0.366254 -3.00
653020 2.072955 2.400848 -1.272325 3.111864 -0.422612 -2.696854 -1.681894 -0.98204
512 1.702116 -1.107163 -1.380732 -1.276758 0.545223 1.280401 2.593850 1.952863 0.1
-2.527110 0.312380
이 -0.603443 0.666823 0.634283 -0.270815 -1.076220 1.049922 5.356484 3.686774 0.48
214908 1.640601 2.940129 -0.124329 1.011352 0.693242 -4.617343 -1.485568 2.903840
2 -0.994733 -1.023390 4.007199 -0.564114 5.584338 -1.735686 5.433420 1.210321 4.04
58906 2.375745 4.616917 1.013108 -2.337668 -2.082066 1.783525 2.435639 0.226523 0.
.547274 -1.781837 -0.106326 1.374207 3.387413 -4.889619 -1.505852 0.338230 -2.8901
039 1.281639 -1.920905 5.578635 1.092317 -1.694160 -0.853831 -0.192298 -3.007503 2
.646364 1.850241 -0.371422 1.335344 -1.995383 -0.808729 0.352333 3.593874 -1.27349
3 -3.643722 0.690935 5.189842 3.566983 2.753287 -4.388271 0.617726 2.565259 1.3784
227 -2.069160 0.660754 3.038895 -0.749613 -0.784138 3.539360 0.445532 -2.331978 1.
1.668694 1.606663 -2.598107 0.890294 -0.994091 2.887277 -4.815674 -0.073630 1.1580
68548 2.118084 3.885089 -0.846701 2.503504 4.909210 4.988792 0.475735 2.786619 -2.
```

# of words, dimension of vector

Gensim for Word Embedding

- Load from existing word embedding
  - Including pre-trained GloVe vector
    - https://nlp.stanford.edu/projects/glove/

```python
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format(filename, binary=False)  # C text format
```

  - binary=True if the file contains binary-format vectors


```
2521005 150
.   C^D?½ÿ/?4V<9e>¿3<87><88>?Wi¹>úÀ^[¿c<9c>i@<97>¼ ?<8a>£¾?|¨k?^_]<9c>?E<81>^K?åëÁ¿â¤ü?Þ
Ú®^^?-ê^RÀ<89>^U<85>¿7¹ø¿AÏ_¿¿Ê7>=ü<95>?<9f><89><84>?¸yF@4/½¿<8c>=u?R<84>o?õ<81>â?^Q0{À
¸<88>¿`¢^_@^P^PÉ?<90>ÒG?¬^T<81>?E'7¿<9f>\¶>ªgÕ?XAÌ?<8d>o÷?Ï!>?ý <99>¿<8f>¸<97>¿^Sn·¿~àÐ
Ç¬¾ä<92>pÀ³4^C¿ùñ@¾-     @H¹D@íMò¾tÕ¹¿³íÓ¿öB\¿¼«,>È°<9c>¾Ó2<95>¿¿J3@^ABÌ¿^]R^@ÀªÛ&@°\<9
><8d>+?ÔÄ¶¿i±7>¼Q<9a>?'^FÃ½ðÎ   ¿<8b>T÷¿<8c>åÌ? ^_Y¿§!&@K¹Ï¿¨G^P@ÏoÎ¿<9f>/^H?hü#À<8f>Ã
e>v;ë<9f>K¾@åN¿<95><85>»¾ýd@À¢-<95>?¾¸h?Q^Q6¿^T^Nr¿K,'¿K«^D@<80>§^Y@<8c>Û¢¿È(G@<93>`Ø¾A
Ù?<81>·<8d>¿×»°¿Ò1£¿¾<93>^K?2ä£?£^A&@i÷ù?<86>=^D>t_-¿ÕKç¾fÇ^XÀMC^O¾,¼!ÀHð<9f>>ì<9d>´ 8{
k@é^Hø>ÛÃ<8f>Àa<94>5@½}b@qßB¿ø^P\¾9ÿÑ?^T+<@: þ½ýs<81>?Gx1?FÁ<93>À^['¾¿<82>Ø9@·Vç>ú<87>Þ
Ëi^P¿æ²²@ô*Þ¿<94>Þ-@Îë<9a>?^[U<81>@<9e>Pì@q^Lä?¾§T¿nyt@Oø^VÀ6^L^X@È½<93>@<82>-<81>?\<9c
dä?w¨Ñ@Îi^D@9êB¿D<83><91>À?^Sä¿SÁÙ½^Aæ¯?^ËX@Âw<9c>ÀÅ¿À¿<89>,->§ø8Àö^Y<8b>ÀýyÉ?<98>¼à>áü
Z¿§éD¾îz@À¾"/@èR^X@¥1ó¿â®Ò?ðç^U@^F^i@°Ôì?^O+¾¾<91>ìª?³hý¿ß^HO¿^Ce´>   ^Bf@Ñ^A£¿âKÄ=<85>
50@¸l<8c>ÀO#^^?2-$@·p°?è^X4¿±Jg@GuÝ><89>7d?<90>â<89>¿ m^DÀ4')?A}B@<9f>æ?¿E½H¿Þ<84>b@È^\
&ÀMêc?Á|~¿&É8@^@^Z<9a>À<94>Ë<96>½^?;<94>?ÐO^@ÀV}mÀ<9b><88><80>?‰»oÀ¾d<95>À²<8e>^G@L¥x@b
e>¯^Y<µk*À.æ<92>½Ãj^_À^M<95>^FÀì<9d><98> a<90>»¿<80>Ë-ÀW<90>1@^E·3ÀÞH`¿¯<9f>è¿ûÄ<97>?Õä
?<9e><96>^]>>o^T¾ÒQ<9f>?<91> óÀjä^KÀ~Q<9d>?<9b>OÜ¿è<99>d@ºÔµ>4æà?'v<9d>?{ÍpÀxk=¿5ã«?<81
1><90>¾?¯Ìi@<9b>Ü$?ßî5À<8c>^Zâ>I«<80>?<9b>qç?8+YÀ´H<86>@-<90>^LÀ'ñz¾Ñ¹<8f>@Êo=@z'M¿^@<8
(ä¿á÷Û>KTD?Ê98Àõ<9d>F?{ø^[ÀÝ
```

FastText

- By Facebook Research
  - https://github.com/facebookresearch/fastText
  - Will be used for obtaining vectors from pre-trained model
    - You may check github repo for training model

- Installation
  - `git clone https://github.com/facebookresearch/fastText.git`
    - or download from Clone or download - **Download ZIP**
  - Move to `fastText` folder
  - `pip install .`

FastText Pre-trained Embeddings

- Download pre-trained vector here:
  - https://github.com/facebookresearch/fastText/blob/master/pretrained-vectors.md
  - 300 Dimension

Models

The models can be downloaded from:

Abkhazian: *bin+text*, *text*	Acehnese: *bin+text*, *text*	Adyghe: *bin+text*, *text*
Afar: *bin+text*, *text*	Afrikaans: *bin+text*, *text*	Akan: *bin+text*, *text*
Albanian: *bin+text*, *text*	Alemannic: *bin+text*, *text*	Amharic: *bin+text*, *text*
Anglo_Saxon: *bin+text*, *text*	Arabic: *bin+text*, *text*	Aragonese: *bin+text*, *text*
Aramaic: *bin+text*, *text*	Armenian: *bin+text*, *text*	Aromanian: *bin+text*, *text*
Assamese: *bin+text*, *text*	Asturian: *bin+text*, *text*	Avar: *bin+text*, *text*
Aymara: *bin+text*, *text*	Azerbaijani: *bin+text*, *text*	Bambara: *bin+text*, *text*
Banjar: *bin+text*, *text*	Banyumasan: *bin+text*, *text*	Bashkir: *bin+text*, *text*
Basque: *bin+text*, *text*	Bavarian: *bin+text*, *text*	Belarusian: *bin+text*, *text*

FastText Pre-trained Embeddings

- If you just want to use existing embedding:
  - Only download text and use it with gensim
- If you want to get embedding for arbitrary words (most cases, because it is the reason to use FastText)
  - You must download bin+text!
  - This file contains all parameters for the model

FastText Pre-trained Embeddings

- How to get vector?
  - Load pre-trained model
    - `from fastText import load_model`
      `model = load_model("wiki.en.bin")`
  - Get vector for the word
    - `model.get_word_vector("hello")`
    - Returns numpy array

```
Python 3.6.3 (default, May  2 2018, 15:28:01)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from fastText import load_model
>>> model = load_model("wiki.en.bin")
>>> vector = model.get_word_vector("hello")
>>> print(type(vector), vector.shape)
<class 'numpy.ndarray'> (300,)
```

Measurement?

- Cosine-similarity
  - Implement yourself (with numpy)
    - https://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists
  - scikit-learn (machine learning package)
    - `pip install scikit-learn`
    - `from sklearn.metrics.pairwise import cosine_similarity`
      `sim = cosine_similarity([A], [B])`
    - Useful if you needs to compare more than two vectors
      - `sim = cosine_similarity([X], [a, b, c, d])`

Embedding Alignment

- Pre-trained matrix & converter (for fastText)
  - https://github.com/Babylonpartners/fastText_multilingual

- You can also check this code (if you want to implement for your own embedding)
  - https://gist.github.com/quadrismegistus/09a93e219a6ffc4f216fb85235535faf

- Bilingual dictionaries (provided by facebook)
  - https://s3.amazonaws.com/arrival/dictionaries/ko-en.txt
  - https://s3.amazonaws.com/arrival/dictionaries/en-ko.txt

NLTK (Natural Language Toolkit)

- Platform for building Python programs to work with human language data
  - Provides easy-to-use interfaces to over 50 corpora and lexical resources (WordNet, stopwords, reuters, etc)
  - Tokenize, Tagging, NER, Parse tree, …

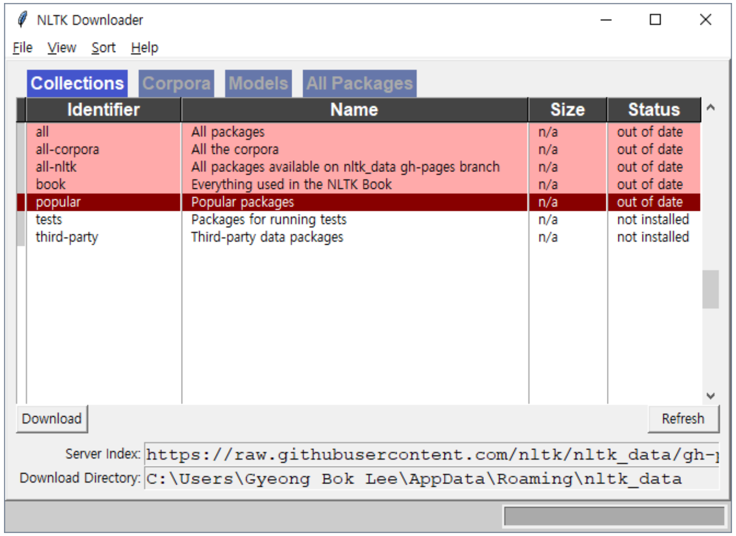
- Install: `pip install nltk`

NLTK (Natural Language Toolkit)

- Download corpus (write this at console/IDLE)
  - `import nltk`
    `nltk.download()`
  - Follow the instructions
    - Installing `popular` is sufficient for many cases
    - You may install other packages in you needs
      - Example: `wordnet`

```
>>> import nltk
>>> nltk.download()
NLTK Downloader
---------------------------------------------------------------------------
    d) Download    l) List    u) Update    c) Config    h) Help    q) Quit
---------------------------------------------------------------------------
Downloader> d

Download which package (l=list; x=cancel)?
  Identifier> popular
```



POS Tagging

- Tokenize
  - nltk.word_tokenize(sentence)
- POS Tagging
  - nltk.pos_tag(tokens)
  - The full list of the tags (Penn Treebank) http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

```
>>> import nltk
>>> sentence = """At eight o'clock on Thursday morning
... Arthur didn't feel very good."""
>>> tokens = nltk.word_tokenize(sentence)
>>> tokens
['At', 'eight', "o'clock", 'on', 'Thursday', 'morning', 'Arthur', 'did', "n't", 'feel',
'very', 'good', '.']
>>> tagged = nltk.pos_tag(tokens)
>>> tagged[0:6]
[('At', 'IN'), ('eight', 'CD'), ("o'clock", 'NN'), ('on', 'IN'), ('Thursday', 'NNP'), ('
morning', 'NN')]
```

POS Tagging

- Korean: koNLPy
  - http://konlpy-ko.readthedocs.io/ko/v0.4.3/

```
>>> from konlpy.tag import Twitter
>>> twitter = Twitter()
>>> print(twitter.morphs(u'단독입찰보다 복수입찰의 경우'))
['단독', '입찰', '보다', '복수', '입찰', '의', '경우', '가']
>>> print(twitter.nouns(u'유일하게 항공기 체계 종합개발 경험을 갖고 있는 KAI는'))
['유일하', '항공기', '체계', '종합', '개발', '경험']
>>> print(twitter.phrases(u'날카로운 분석과 신뢰감 있는 진행으로'))
['분석', '분석과 신뢰감', '신뢰감', '분석과 신뢰감 있는 진행', '신뢰감 있는 진행', '진행'
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ'))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되나욬', 'Noun'), ('ㅋㅋ', '
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ', norm=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되', 'Verb'), ('나요', 'Eom
>>> print(twitter.pos(u'이것도 되나욬ㅋㅋ', norm=True, stem=True))
[('이', 'Determiner'), ('것', 'Noun'), ('도', 'Josa'), ('되다', 'Verb'), ('ㅋㅋ', 'Ko
```

POS Tagging

- Contains 5 packages
  - `Hannanum`, `Kkma`, `Komoran`, `Mecab`, `Twitter`
  - Recommend to use `Twitter` tagger…
- Web API is also available for `Twitter` tagger
  - https://github.com/open-korean-text/open-korean-text

Web API Service

open-korean-text-api
이 API 서비스는 Heroku 서버에서 제공되며(Domain: https://open-korean-text.herokuapp.com/) 현재 정규화(normalization), 토큰화(tokenization), 어근화(stemmin), 어구 추출(phrase extract) 서비스를 제공합니다.

각 서비스와 사용법은 다음과 같습니다.
`normalize`, `tokenize`, `stem`, `extractPhrases` 가 각 서비스의 **Action** 이 되며 **Query parameter** 는 `text` 입니다.

서비스	사용법
정규화	https://open-korean-text.herokuapp.com/normalize?text=오픈코리안텍스트
토큰화	https://open-korean-text.herokuapp.com/tokenize?text=오픈코리안텍스트
어근화	https://open-korean-text.herokuapp.com/stem?text=오픈코리안텍스트
어구 추출	https://open-korean-text.herokuapp.com/extractPhrases?text=오픈코리안텍스트

Text Similarity

- Not semantic similarity
  - Use word embedding for this case
- Useful for dealing with typo
  - Ex: `similarity` vs `simliarity`
- Metrics for measuring similarity
  - Edit distance (levenshtein distance)
  - Jaro distance

The Jaro Similarity sim_j of two given strings s_1 and s_2 is

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Where:

- $|s_i|$ is the length of the string s_i;
- m is the number of *matching characters* (see below);
- t is half the number of *transpositions* (see below).

Text Similarity

- You can use `jellyfish` package
  - https://github.com/jamesturk/jellyfish
  - `pip install jellyfish`

Included Algorithms

String comparison:

- Levenshtein Distance
- Damerau-Levenshtein Distance
- Jaro Distance
- Jaro-Winkler Distance
- Match Rating Approach Comparison
- Hamming Distance

Text Similarity

- Edit Distance (Levenshtein Distance)
  - # of character changes needed
  - Small number means similar text
- Jaro Distance
  - Defined by right formula:
    - https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance
  - Large value means similar text

$$sim_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

```
>>> import jellyfish
>>> ld = jellyfish.levenshtein_distance
>>> jd = jellyfish.jaro_distance
>>> ld("jellyfish", "swordfish")
5
>>> jd("jellyfish", "swordfish")
0.6296296296296297
>>> ld("history", "ancient")
7
>>> jd("history", "ancient")
0.42857142857142855
```