

딥러닝을 활용한 자연어 분석

Word2Vec, Doc2Vec

김용범

무영인터네셔널

머리/Noun
label 머리/Noun

Show All Data

Isolate points

Search

머리

neighbors

distance

Nearest points in the

손질/Noun

들어오다/Verb

간/Noun

미용/Noun

분/Noun

호가/Noun

진도군/Noun

경호실/Noun

조도면/Noun

도기/Noun

호보/Noun

응당/Noun

BOOKMARKS (0)

Jupyter Notebook 소스보기

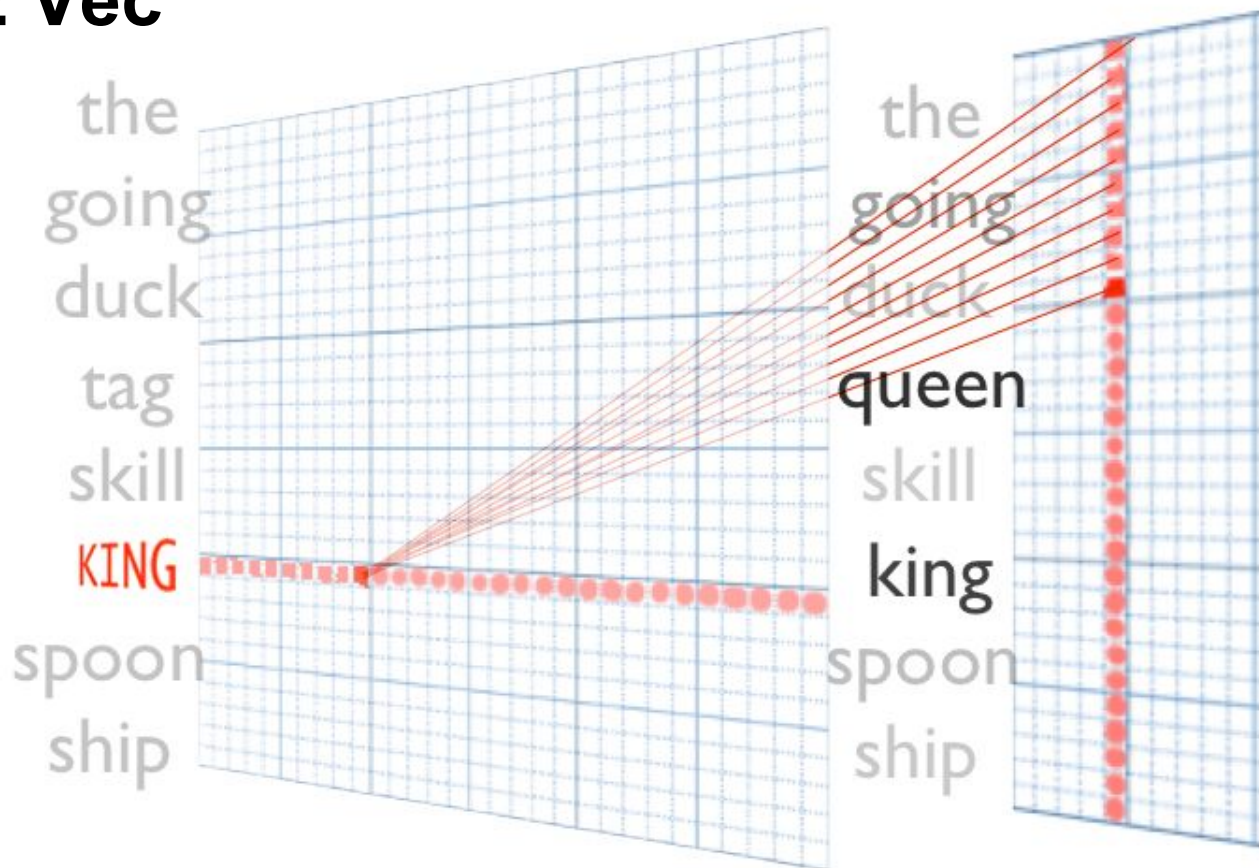
http://nbviewer.jupyter.org/github/YongBeomKim/nltk_tutorial/blob/master/04.word2vec.ipynb

Word 2 Vec

임베딩 - 원시 데이터(raw data)를 학습 후 축소된 숫자 목록으로 변환

1. **Tf-idf** : 벡터화에 바탕을 둔 **용어빈도/ 역 문서 빈도** 를 활용
2. **One-hot Encoding** : 단어간 유사도는 알기 어렵다
3. **Word2Vec** : Mikolov가 고안한 방법으로 "**주변 단어를 보면
그 단어를 알 수 있다**" (John Firth) 에서 착안

Word 2 Vec



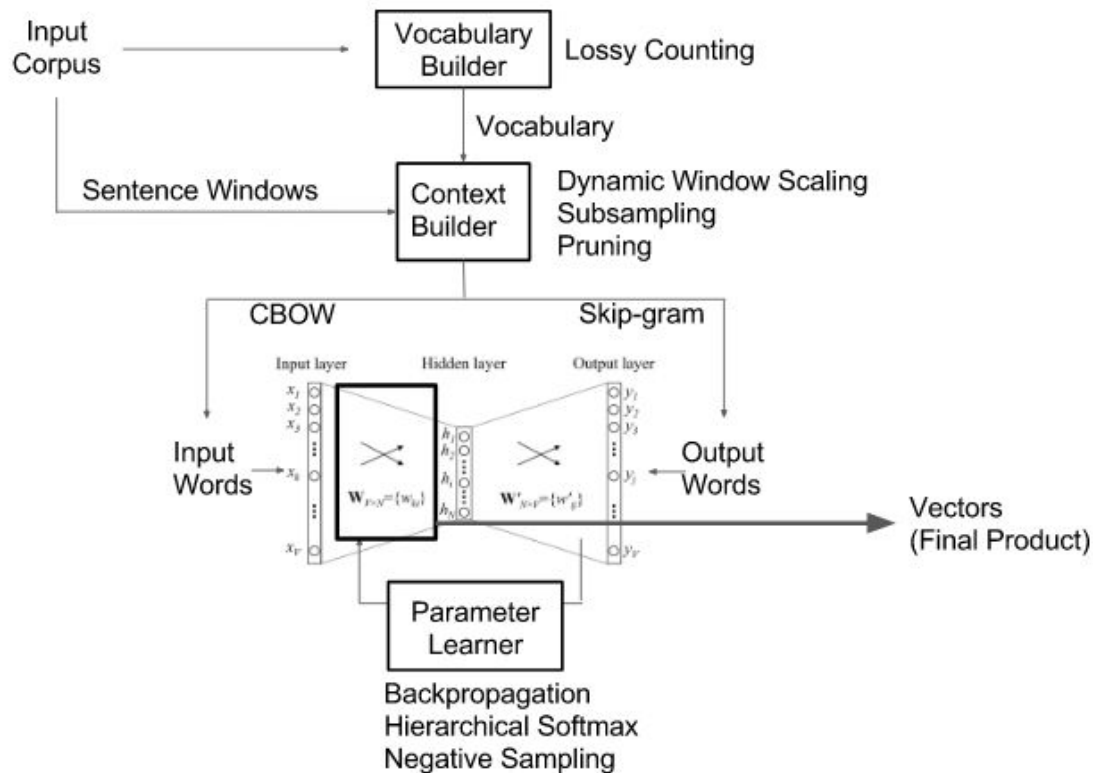
Word 2 Vec

CBOW skip-gram

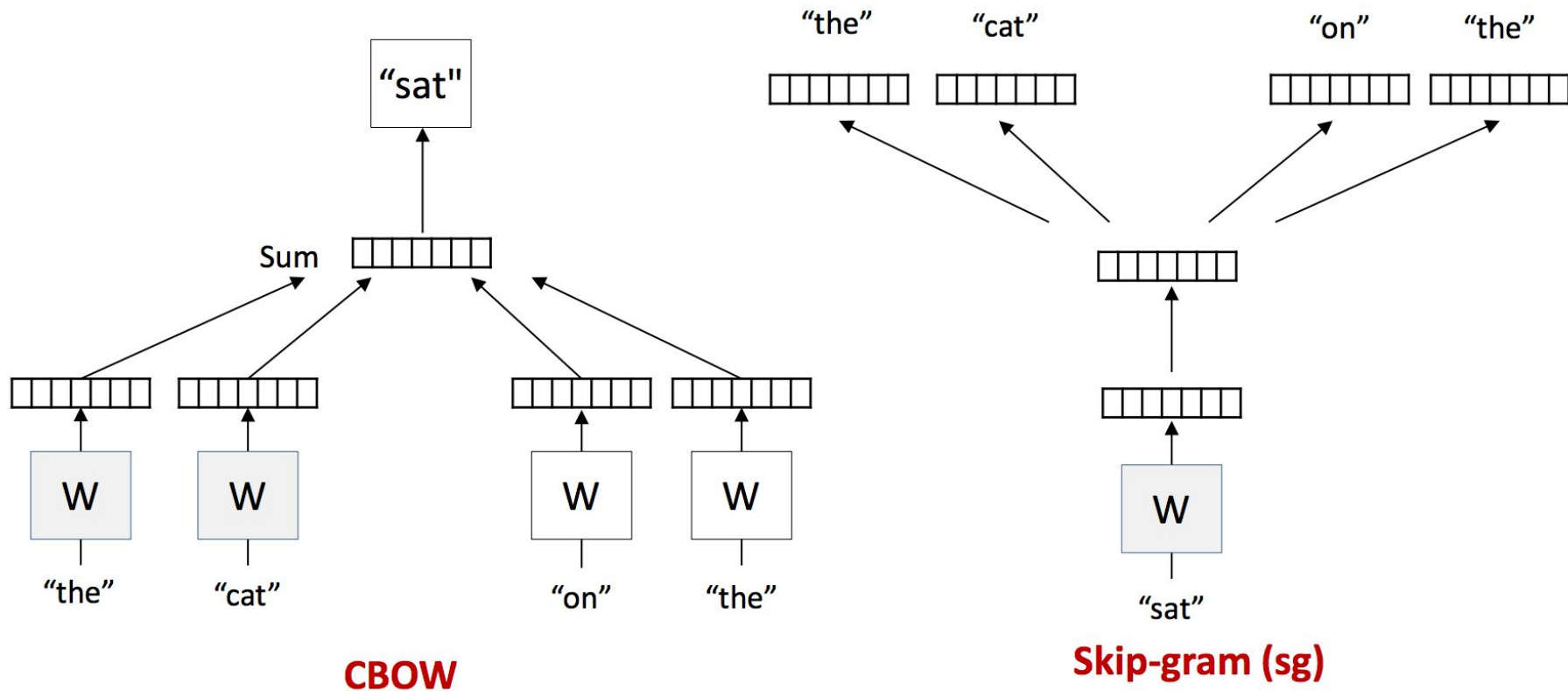
Word 2 Vec

1. 2013년 구글에서 개발/ 공개한 기법
2. 고밀도 단어벡터공간에 단어간 유사도(코사인유사도)를 표현
3. **(Continuous Bag of Word)** - 문맥 속 어휘들로 모델을 만들고 **Target** 단어를 예측한다
4. **(Skip Gram)** - **Target** 단어를 중심으로 모델을 만들고 문맥 요소들을 예측한다

Word 2 Vec



Word 2 Vec



CBOW - Continuous Bag-of-Words

1. 주변의 **token**을 대상으로 모델을 생성
2. 문장의 여러 단어들 가운데, 빈 단어를 채운다
3. 단어 앞뒤로 **여유분의 token**을 선택
4. 이들을 활용하여 **적합한 내용을 유추하는 Network**를 생성한다

Skip Gram

1. 주어진 1개의 **token** 을 갖고서 **주변 단어들을 유추**한다
2. **단어간의 빈도**를 활용하여 가까운 단어일수록 가깝고, 먼 단어일수록 빈도가 낮음을 활용한다
3. **샘플링 기준이 되는 단어**를 몇개로 정하는지에 따라 연산시간이 많이 차이난다 (다양한 기법이 가능)
4. **Skip-gram**이 더 좋은 결과를 보여준다

Word 2 Vec

Tensorflow

Tensorflow - 전처리

word_sequence : 단어 배열 원본

word_list : 단어 배열 중복제거

word_dict : **word_list**의 인덱스 배열을 생성

```
In [1]: f = open('./data/베를린선언.txt', 'r')
        texts_Berlin_raw = f.read()
        f.close()
        texts_Berlin = texts_Berlin_raw.split('\n\n')
```

```
In [2]: import string
        punct = string.punctuation
        punct = [punct[i] for i in range(len(punct))]
        punct = punct + ['\n', ')', '(']

        for dump in punct:
            texts_Berlin = [txt.replace(dump, '') for txt in texts_Berlin]
        print(len(texts_Berlin))
```

```
In [3]: word_sequence = " ".join(texts_Berlin).split()
        word_list = " ".join(texts_Berlin).split()
        word_list = list(set(word_list))
        word_dict = {w: i for i, w in enumerate(word_list)}
```

Tensorflow - 변수/ 함수 설정

```
In [6]: training_epoch = 300
learning_rate = 0.1
batch_size = 20
embedding_size = 2 # 단어벡터 임베딩 차원 (x, y 2개만 출력)
num_sampled = 15 # 모델의 nce_loss 샘플크기 (batch_size 보다 작아야 한다)
voc_size = len(word_list) # 총 단어 갯수
```

```
In [7]: inputs = tf.placeholder(tf.int32, shape=[batch_size])
labels = tf.placeholder(tf.int32, shape=[batch_size, 1])

# word2vec 모델의 결과를 저장할 임베딩 벡터를 정의
embeddings = tf.Variable(tf.random_uniform([voc_size, embedding_size], -1.0, 1.0))
selected_embed = tf.nn.embedding_lookup(embeddings, inputs)

# nce_loss 함수에서 사용할 변수들을 정의합니다.
nce_weights = tf.Variable(tf.random_uniform([voc_size, embedding_size], -1.0, 1.0))
nce_biases = tf.Variable(tf.zeros([voc_size]))

# nce_loss 함수는 tf.nn.nce_loss 함수를 사용
loss = tf.reduce_mean(
    tf.nn.nce_loss(nce_weights, nce_biases, labels, selected_embed, num_sampled, voc
train_op = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

Tensorflow - 모델 학습

```
In [8]: with tf.Session() as sess:
        init = tf.global_variables_initializer()
        sess.run(init)
        for step in range(1, training_epoch + 1):
            batch_inputs, batch_labels = random_batch(skip_grams, batch_size)
            _, loss_val = sess.run([train_op, loss],
                                   feed_dict={inputs: batch_inputs,
                                               labels: batch_labels})

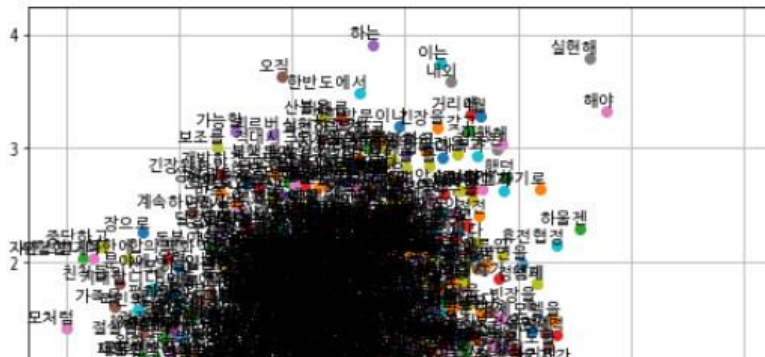
            if step % 30 == 0:
                print("loss at step ", step, ": ", loss_val)
        # with 구문 안에서는 sess.run 대신 간단히 eval() 함수를 사용할 수 있습니다.
        trained_embeddings = embeddings.eval()
```

```
loss at step 30 : 41.542034
loss at step 60 : 41.47303
loss at step 90 : 25.07451
loss at step 120 : 33.435047
loss at step 150 : 32.99861
loss at step 180 : 10.731102
loss at step 210 : 36.63857
loss at step 240 : 22.269367
loss at step 270 : 32.060616
loss at step 300 : 13.882846
```

Tensorflow - 결과출력

```
In [9]: %matplotlib inline
from matplotlib import font_manager, rc
font_fname = './data/D2Coding.ttf'
font_name = font_manager.FontProperties(fname=font_fname).get_name()
rc('font', family=font_name)
```

```
In [10]: import matplotlib.pyplot as plt
plt.figure(figsize=(8,8))
for i, label in enumerate(word_list):
    x, y = trained_embeddings[i]
    plt.scatter(x, y)
    plt.annotate(label, xy=(x, y), xytext=(5, 2),
                  textcoords='offset points', ha='right', va='bottom')
plt.grid(True)
plt.show()
```



Word 2 Vec

gensim

`pip install --upgrade gensim`

1. Why is **Gensim Word2Vec** so much **faster** than **Keras GPU**? [[link](#)]
2. 데이터와 모델을 저장하고, 호출하는 방식을 잘 익히자
3. 주요한 기능을 메소드 함수로 제공



<https://radimrehurek.com/gensim/install.html>

Twitter 한글 Tag 추가 후 전처리

```
In [11]: %%time
# 텍스트를 한 줄씩 문법 tag를 추가한다
results, lines = [], texts_Berlin
from konlpy.tag import Twitter
twitter = Twitter()

for line in lines:
    malist = twitter.pos(line, norm=True, stem=True)
    result = [ word[0] for word in malist # 어미/조사/구두점 제외
               if not word[1] in ["Eomi", "Josa", "Punctuation"] ]
    rl = (" ".join(result)).strip()
    results.append(rl)

texts_file = './data/Berlin.tagged'
with open(texts_file, 'w', encoding='utf-8') as fp:
    fp.write("\n".join(results))
```

CPU times: user 5.85 s, sys: 134 ms, total: 5.99 s
Wall time: 2.07 s

gensim 저장된 Text 결과물 살펴보기

```
In [13]: ! cat ./data/Berlin.tagged | head -n 5
```

존경 하다 독일 국민 여러분
고국 계시다 국민 여러분
하울 젠 괴르버 재단 이사 님 모드 전 동독 총리 님 비룻 하다 내외 귀빈 여러분
먼저 냉전 분단 넘다 통일 이루다
그 힘 유럽 통합 국제 평화 선도 있다

Word2Vec 학습 후 모델 저장

```
In [13]: %%time
# Word2Vec 데이터 모델을 학습한 뒤 저장한다
from gensim.models import word2vec
data = word2vec.LineSentence(texts_file)
model = word2vec.Word2Vec(data, size=200, window=10, hs=1, min_count=2, sg=1)
model.save("./data/Berlin.model")
print("ok")
```

ok

CPU times: user 304 ms, sys: 27.8 ms, total: 332 ms

Wall time: 325 ms

```
from gensim.models import Word2Vec
```

```
Word2Vec(data, size=100, window = 2, min_count=50, workers=4, iter=100, sg=1)
```

1. **size = 100** : 100차원 벡터를 사용 (크면 차원의 저주)
2. **window = 2** : 주변 단어(window)는 앞 뒤 두개
3. **min_count = 50** : 출현 빈도가 50번 미만인 단어는 제외
4. **iter = 100** : 멀티코어를 활용 100번 반복 (Multi Thread)
5. **sg = 1** : CBOW, Skip-Gram 중 **Skip-Gram**를 사용

모델의 활용 - 모델 생성후에는 이것만 실행하면 된다

```
In [14]: from gensim.models import word2vec  
model = word2vec.Word2Vec.load('./data/Berlin.model')
```

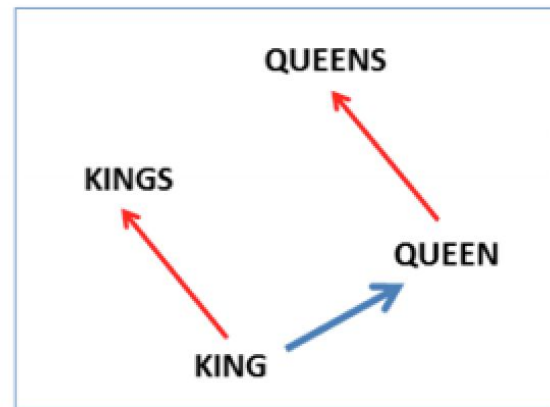
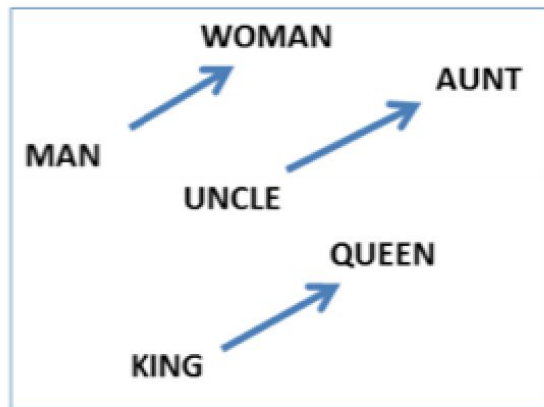
```
In [15]: model.wv.most_similar(positive=['한반도'])
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated as `np.int64 == np.dtype(int).type`.  
    if np.issubdtype(vec.dtype, np.int):
```

```
Out[15]: [('적', 0.9921074509620667),  
          ('하다', 0.9911549091339111),  
          ('있다', 0.9908924698829651),  
          ('정치', 0.9889532327651978),  
          ('경제', 0.9877470135688782),  
          ('군사', 0.9866889715194702),  
          ('이다', 0.9864642024040222),  
          ('것', 0.9863808155059814),  
          ('수', 0.9858399629592896),  
          ('세계', 0.9850960969924927)]
```

단어들의 벡터 연산

1. 긍정/부정 (벡터의 방향성) 관계망도 분석 가능하다
2. 하지만 이는 연산결과일 뿐, 구체적 내용분석은 분야의 전문지식을 갖고서 별도 작업을 해야한다



(Mikolov et al., NAACL HLT, 2013)

Word 2 Vec - 단어간의 벡터관계 활용

```
In [19]: model.wv.most_similar(positive=['북한', '한반도'],  
                                negative=['전쟁'])
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of  
the second argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated  
as `np.int64 == np.dtype(int).type`.  
    if np.issubdtype(vec.dtype, np.int):
```

```
Out[19]: [('하다', 0.859728217124939),  
          ('것', 0.8535114526748657),  
          ('있다', 0.8520656824111938),  
          ('이다', 0.8445901274681091),  
          ('적', 0.8365031480789185),  
          ('되다', 0.8355264067649841),  
          ('협력', 0.818851888179779),  
          ('냉전', 0.8153940439224243),  
          ('들', 0.8132228851318359),  
          ('정치', 0.809958279132843)]
```

시각화 - 2차원 데이터로 차원 축소

```
In [18]: # model.wv.vocab : { word: object of numeric vector }  
vocab = list(model.wv.vocab)  
X = model[vocab]
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/ipykernel_launcher.py:3: DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
```

This is separate from the ipykernel package so we can avoid doing imports until

```
In [19]: from sklearn.manifold import TSNE  
tsne = TSNE(n_components=2)  
X_tsne = tsne.fit_transform(X)
```

```
In [20]: import pandas as pd  
df = pd.DataFrame(X_tsne, index=vocab, columns=['x', 'y'])  
df.head()
```

```
Out[20]:
```

	x	y
존경	-4.606161	20.196222
하다	10.832066	-31.455429
독일	8.204652	-25.016357



TSNE - t-distributed Stochastic Neighbor Embedding

고차원 공간에서의 **유클리디안 거리측정방법**을 활용하여
데이터 포인트의 **유사성**을 표현하는
조건부 확률로 변환하는 방법

단점으로는 **조건부 확률의 기준**이 정해져 있지 않아서
생성시 마다 모양이 다르다

matplotlib - 2차원 데이터로 차원 축소

```
In [21]: %matplotlib inline
from matplotlib import font_manager, rc
font_fname = './data/D2Coding.ttf'
font_name = font_manager.FontProperties(fname=font_fname).get_name()
rc('font', family=font_name)

import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,12))
plt.grid(True)
ax = fig.add_subplot(1, 1, 1)
ax.scatter(df['x'], df['y'])
for word, pos in df.iterrows():
    ax.annotate(word, pos)
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:106: M
same arguments as a previous axes currently reuses the earlier instance. In a future version,
eanwhile, this warning can be suppressed, and the future behavior ensured, by passing a unique
warnings.warn(message, mplDeprecation, stacklevel=1)
```



Doc 2 Vec

Doc 2 Vec

비지도 학습

Doc 2 Vec

1. **Word2Vec** 는 개별 단어 **Token**의 관계를 학습
2. **Doc2Vec**는 문장, 단락, 문서와 같은 더 큰 블록에 대한 연속표현을 비지도 학습으로 모델을 생성
3. 학습 데이터의 성격이 유사할수록 관계망이 잘 생성된다
4. **GloVe** 알고리즘(2014) / embedding 결과에 **tf/idf** 가중치를 곱한 평균을 활용방법 등 다양한 대안들이 모색

<https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/09/glove/>

Doc 2 Vec - 데이터 호출 및 전처리

```
In [22]: from konlpy.tag import Twitter
twitter = Twitter()

def read_data(filename):
    with open(filename, 'r') as f:
        data = [line.split('\t') for line in f.read().splitlines()]
    from random import randint
    random_data = [data[randint(1, len(data))] for no in range(int(len(data)/10))]
    return random_data

def tokenize(doc):
    # norm, stem은 optional
    return ['/'.join(t) for t in twitter.pos(doc, norm=True, stem=True)]
```

```
In [23]: from collections import namedtuple
train_data = read_data('data/ratings_train.txt')
train_docs = [(tokenize(row[1]), row[2]) for row in train_data[1:]]
TaggedDocument = namedtuple('TaggedDocument', 'words tags')
tagged_train_docs = [TaggedDocument(d, [c]) for d, c in train_docs]
```


Doc 2 Vec - 모델 파라미터 설정 및 학습

```
In [26]: %%time
from gensim.models import doc2vec
doc_vectorizer = doc2vec.Doc2Vec(vector_size=300, alpha=0.025, min_alpha=0.025, seed=1234)
doc_vectorizer.build_vocab(tagged_train_docs)

for epoch in range(10):
    doc_vectorizer.train(tagged_train_docs,
                        total_examples = doc_vectorizer.corpus_count,
                        epochs = doc_vectorizer.epochs)
    doc_vectorizer.alpha -= 0.002
    doc_vectorizer.min_alpha = doc_vectorizer.alpha

# 학습이 완료된 모델의 데이터를 저장한다
doc_vectorizer.save('data/doc2vec.model')
```

CPU times: user 53.4 s, sys: 3.87 s, total: 57.3 s

Wall time: 26.8 s

Doc 2 Vec - 저장된 모델 활용하기

```
In [10]: # 저장된 모델을 호출하여 활용한다
from gensim.models import doc2vec
from pprint import pprint
doc_vectorizer = doc2vec.Doc2Vec.load('data/doc2vec.model')
pprint(doc_vectorizer.wv.most_similar('공포/Noun'))
```

```
[('시리즈/Noun', 0.7168623805046082),
 ('에겐/Josa', 0.7058377265930176),
 ('에서/Noun', 0.67695552110672),
 ('중/Suffix', 0.6729834079742432),
 ('SF/Alpha', 0.6674826145172119),
 ('초등학교/Noun', 0.6674530506134033),
 ('메다/Verb', 0.6669254899024963),
 ('공포영화/Noun', 0.664033055305481),
 ('역대/Noun', 0.6585553288459778),
 ('칼/Noun', 0.6533156037330627)]
```

Doc 2 Vec - 단어간 벡터연산 활용

```
In [32]: pprint(doc_vectorizer.wv.most_similar(positive=['여자/Noun', '최악/Noun'],  
                                                negative=['남자/Noun']))
```

```
[('근본/Noun', 0.36848971247673035),  
 ('개판/Noun', 0.3549008369445801),  
 ('최고/Noun', 0.32961606979370117),  
 ('대작/Noun', 0.3263331949710846),  
 ('최강/Noun', 0.3136056363582611),  
 ('최고봉/Noun', 0.312497615814209),  
 ('역작/Noun', 0.3085203468799591),  
 ('짱/Noun', 0.30231973528862),  
 ('수작/Noun', 0.29365360736846924),  
 ('제일/Noun', 0.29353460669517517)]
```

```
/home/markbaum/Python/python/lib/python3.6/site-packages/gensim/matutils.py:737: FutureWarning: Conversion of  
the second argument of issubdtype from `int` to `np.signedinteger` is deprecated. In future, it will be treated  
as `np.int64 == np.dtype(int).type`.  
    if np.issubdtype(vec.dtype, np.int):
```

Doc 2 Vec - 단어 묶음을 활용하여 벡터간 Cosin 유사도 측정
문장이 짧은 경우에는
Word 2 Vec 에 비해 유의미한 벡터를 찾기 힘들다.

```
In [37]: doc_vectorizer.infer_vector(['호리/Noun', '여자/Noun', '공포/Noun'])[:10]
```

```
Out[37]: array([-0.01689258,  0.00375193,  0.00620129,  0.01125604,  0.0064879 ,  
               -0.00527091,  0.0069678 , -0.00344358, -0.01045213,  0.00610426],  
          dtype=float32)
```

```
In [35]: doc_vectorizer.infer_vector(['공포/Noun', '연설/Noun', '역작/Noun']).sum()
```

```
Out[35]: 0.028070673
```