Describing sources

https://nbviewer.jupyter.org/github/danielfrg/word2vec/blob/master/examples/word2vec.ipynb (https://nbviewer.jupyter.org/github/danielfrg/word2vec/blob/master/examples/word2vec.ipynb)

https://github.com/danielfrg/word2vec (https://github.com/danielfrg/word2vec)

Function testing

In [1]:
```
import word2vec
```

Create text8-phrases file, better for word2vec input according to source

In [42]:
```
with open("text8") as myfile:
    firstNlines=myfile.readlines()[0:5] #put here the interval you want
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [2]:
```
word2vec.word2phrase('text8', 'text8-phrases', verbose=True)
```

```
Starting training using file text8
Words processed: 17000K     Vocab size: 4399K
Vocab size (unigrams + bigrams): 2419827
Words in train file: 17005206
Words written: 17000K
```

Train word2vec model -> create word vectors in binary format

```
In [3]: word2vec.word2vec('text8-phrases', 'text8.bin', size=100, verbose=True)
```

```
Starting training using file text8-phrases
Vocab size: 98331
Words in train file: 15857306
Alpha: 0.000002  Progress: 100.04%  Words/thread/sec: 285.58k  ec: 26
0.55k  read/sec: 279.33k  ords/thread/sec: 280.44k  sec: 280.04k  : 0.
023684  Progress: 5.28%  Words/thread/sec: 282.78k  s/thread/sec: 283.
70k   Words/thread/sec: 283.88k  5  Progress: 7.87%  Words/thread/sec:
283.75k  .022819  Progress: 8.74%  Words/thread/sec: 283.33k  /thread/
sec: 283.85k  ds/thread/sec: 283.81k  73  Progress: 11.32%  Words/thre
ad/sec: 284.11k  .021954  Progress: 12.20%  Words/thread/sec: 284.82k
read/sec: 284.00k   Progress: 13.91%  Words/thread/sec: 283.94k  ha:
0.021304  Progress: 14.80%  Words/thread/sec: 284.48k  ead/sec: 284.68
k  ress: 17.39%  Words/thread/sec: 284.36k  k  Words/thread/sec: 284.6
1k  9998  Progress: 20.03%  Words/thread/sec: 285.08k   0.019781  Prog
ress: 20.89%  Words/thread/sec: 284.89k  .75%  Words/thread/sec: 285.0
5k  ds/thread/sec: 284.95k  ress: 23.48%  Words/thread/sec: 285.13k  P
rogress: 24.37%  Words/thread/sec: 285.26k    Words/thread/sec: 285.39
k  .018266  Progress: 26.95%  Words/thread/sec: 284.95k  5.24k  ress:
28.68%  Words/thread/sec: 285.34k  s: 29.54%  Words/thread/sec: 284.96
k  Words/thread/sec: 284.98k   0.017188  Progress: 31.26%  Words/threa
d/sec: 285.21k  k  ead/sec: 285.21k  : 0.016532  Progress: 33.88%  Wor
ds/thread/sec: 285.08k   34.74%  Words/thread/sec: 285.01k  5  Progres
s: 35.59%  Words/thread/sec: 285.20k  .015883  Progress: 36.48%  Word
s/thread/sec: 285.17k  %  Words/thread/sec: 285.26k  85.05k  ress: 39.
97%  Words/thread/sec: 285.26k  /sec: 285.26k  d/sec: 285.36k  d/sec:
285.20k  .014148  Progress: 43.42%  Words/thread/sec: 285.27k  .013712
Progress: 45.17%  Words/thread/sec: 285.38k  .03%  Words/thread/sec: 2
85.49k  ad/sec: 285.23k  0.012847  Progress: 48.63%  Words/thread/sec:
285.30k  ead/sec: 285.35k  : 285.34k  ress: 51.22%  Words/thread/sec:
285.23k  ss: 52.08%  Words/thread/sec: 285.31k  5.20k  Words/thread/se
c: 285.31k  a: 0.011117  Progress: 55.55%  Words/thread/sec: 285.26k
lpha: 0.010903  Progress: 56.40%  Words/thread/sec: 285.15k  thread/se
c: 285.28k  25k  85.33k  ad/sec: 285.33k  read/sec: 285.31k  : 285.36k
63.38%  Words/thread/sec: 285.47k  pha: 0.008939  Progress: 64.26%  Wo
rds/thread/sec: 285.38k  5.12%  Words/thread/sec: 285.37k  ress: 66.8
4%  Words/thread/sec: 285.35k   67.70%  Words/thread/sec: 285.42k  ea
d/sec: 285.38k  rds/thread/sec: 285.35k  /thread/sec: 285.32k  s: 71.1
7%  Words/thread/sec: 285.32k  993  Progress: 72.04%  Words/thread/se
c: 285.41k  ha: 0.006778  Progress: 72.90%  Words/thread/sec: 285.39k
.34k  k   285.33k  gress: 78.09%  Words/thread/sec: 285.23k  ec: 285.3
8k   80.74%  Words/thread/sec: 285.51k  .47k  d/sec: 285.44k  84.22%
Words/thread/sec: 285.47k  ess: 85.08%  Words/thread/sec: 285.46k  /th
read/sec: 285.52k  03298  Progress: 86.82%  Words/thread/sec: 285.38k
k  285.47k  %  Words/thread/sec: 285.35k  1  Progress: 91.13%  Words/t
hread/sec: 285.35k  pha: 0.002003  Progress: 92.00%  Words/thread/sec:
285.31k  ess: 92.86%  Words/thread/sec: 285.36k  70  Progress: 93.73%
Words/thread/sec: 285.44k  : 0.001352  Progress: 94.61%  Words/thread/
sec: 285.36k  rds/thread/sec: 285.29k   Progress: 98.06%  Words/threa
d/sec: 285.39k  0319  Progress: 98.75%  Words/thread/sec: 285.39k  .00
0159  Progress: 99.38%  Words/thread/sec: 285.46k
```

Create vector clusters based on trained model

In [4]:
```
word2vec.word2clusters('text8', 'text8-clusters.txt', 100, verbose=True
```

```
Starting training using file text8
Vocab size: 71291
Words in train file: 16718843
Alpha: 0.000002  Progress: 100.03%  Words/thread/sec: 283.28k    Progr
ess: 0.85%  Words/thread/sec: 268.99k  585  Progress: 1.67%  Words/thr
ead/sec: 283.56k  a: 0.024387  Progress: 2.46%  Words/thread/sec: 272.
15k  5k  023763  Progress: 4.96%  Words/thread/sec: 279.94k   Progres
s: 5.79%  Words/thread/sec: 282.81k  read/sec: 282.25k  : 281.97k  :
9.07%  Words/thread/sec: 281.75k  5  Progress: 9.91%  Words/thread/se
c: 283.11k  sec: 282.07k  0%  Words/thread/sec: 281.80k  ec: 282.71k
: 282.42k  ha: 0.021513  Progress: 13.96%  Words/thread/sec: 282.16k
14.80%  Words/thread/sec: 283.07k  ress: 16.40%  Words/thread/sec: 28
2.38k  /sec: 282.76k  ec: 282.60k    Progress: 18.86%  Words/thread/se
c: 282.73k  3.03k  ss: 20.53%  Words/thread/sec: 283.09k  ha: 0.019465
Progress: 22.15%  Words/thread/sec: 283.10k   0.019258  Progress: 22.9
8%  Words/thread/sec: 283.13k   23.80%  Words/thread/sec: 283.15k  864
3  Progress: 25.44%  Words/thread/sec: 283.32k  : 0.018439  Progress:
26.25%  Words/thread/sec: 283.18k  %  Words/thread/sec: 283.22k  ress:
27.89%  Words/thread/sec: 283.39k  ds/thread/sec: 283.16k  : 29.51%  W
ords/thread/sec: 283.33k  k  31.95%  Words/thread/sec: 283.24k  ords/t
hread/sec: 283.34k  lpha: 0.016607  Progress: 33.59%  Words/thread/se
c: 283.01k  ogress: 34.42%  Words/thread/sec: 283.29k   Words/thread/s
ec: 283.35k  lpha: 0.015379  Progress: 38.50%  Words/thread/sec: 283.1
7k  ess: 39.32%  Words/thread/sec: 283.22k  4967  Progress: 40.14%  Wo
rds/thread/sec: 283.32k   1.78%  Words/thread/sec: 283.50k   Words/thr
ead/sec: 283.41k   0.013944  Progress: 44.24%  Words/thread/sec: 283.4
8k  3532  Progress: 45.88%  Words/thread/sec: 283.49k  ec: 283.61k  /t
hread/sec: 283.45k  read/sec: 282.89k  s: 56.51%  Words/thread/sec: 28
2.97k  ds/thread/sec: 283.06k    Progress: 59.80%  Words/thread/sec: 2
83.04k  ead/sec: 283.07k   .08%  Words/thread/sec: 283.14k  ds/thread/
sec: 283.20k  08625  Progress: 65.51%  Words/thread/sec: 283.06k  283.
18k  ss: 69.61%  Words/thread/sec: 283.18k  s: 70.42%  Words/thread/se
c: 283.06k  Words/thread/sec: 283.08k  rogress: 73.68%  Words/thread/s
ec: 283.04k  rogress: 75.33%  Words/thread/sec: 282.98k   2k   Progres
s: 77.76%  Words/thread/sec: 282.92k  83.17k  ec: 283.23k  ords/threa
d/sec: 283.12k  2k  lpha: 0.004120  Progress: 83.53%  Words/thread/se
c: 283.04k  k  283.06k  3.08k  pha: 0.002687  Progress: 89.27%  Words/
thread/sec: 283.08k  283.07k  02k   0.001875  Progress: 92.51%  Words/
thread/sec: 283.07k  ha: 0.001673  Progress: 93.32%  Words/thread/sec:
283.00k  ead/sec: 283.07k  ha: 0.001058  Progress: 95.78%  Words/threa
d/sec: 283.02k  40%  Words/thread/sec: 283.04k   ogress: 99.63%  Word
s/thread/sec: 283.18k
```

Predictions

In [5]:
```
%load_ext autoreload
%autoreload 2
```

In [7]:
```
model = word2vec.load('text8.bin')
```

```
In [8]: model.vocab
```

```
Out[8]: array(['</s>', 'the', 'of', ..., 'denishawn', 'tamiris', 'dolophine'],
              dtype='<U78')
```

```
In [9]: model.vectors.shape
```

```
Out[9]: (98331, 100)
```

```
In [10]: #retrieve vector of individual words
         model['dog'].shape
```

```
Out[10]: (100,)
```

```
In [11]: model['dog'][:10]
```

```
Out[11]: array([-0.15531589, -0.06376425,  0.14083751, -0.1079576 ,  0.1532326
         5,
                 0.03200457,  0.07057863,  0.16911601, -0.09268221,  0.0911236
         3])
```

```
In [12]: model.distance("dog", "cat", "animal")
```

```
Out[12]: [('dog', 'cat', 0.8733129767728741),
          ('dog', 'animal', 0.5059533220887221),
          ('cat', 'animal', 0.6191897081668528)]
```

```
In [13]: # get most similar words from vocab
         indexes, metrics = model.similar("cleaning")
```

```
In [14]: indexes, metrics
```

```
Out[14]: (array([15037, 13633,  4438,  6167, 14544, 15467, 14727,  4541, 14426,
                 21367]),
          array([0.76619445, 0.75956396, 0.75780988, 0.75715287, 0.75610959,
                 0.75096562, 0.74635639, 0.74605216, 0.74550337, 0.73501854]))
```

```
In [15]: model.vocab[indexes]
```

```
Out[15]: array(['needles', 'lining', 'cutting', 'smoke', 'drying', 'spray',
                'washing', 'clean', 'toilet', 'punches'], dtype='<U78')
```

```
In [16]: model.generate_response(indexes, metrics)
```

```
Out[16]: rec.array([('needles', 0.76619445), ('lining', 0.75956396),
                    ('cutting', 0.75780988), ('smoke', 0.75715287),
                    ('drying', 0.75610959), ('spray', 0.75096562),
                    ('washing', 0.74635639), ('clean', 0.74605216),
                    ('toilet', 0.74550337), ('punches', 0.73501854)],
                   dtype=[('word', '<U78'), ('metric', '<f8')])
```

```
In [17]: model.generate_response(indexes, metrics).tolist()
```

```
Out[17]: [('needles', 0.7661944532619005),
          ('lining', 0.7595639637267579),
          ('cutting', 0.757809882995842),
          ('smoke', 0.7571528712778193),
          ('drying', 0.7561095863483643),
          ('spray', 0.75096562260729),
          ('washing', 0.7463563944841782),
          ('clean', 0.7460521583237942),
          ('toilet', 0.745503365818746),
          ('punches', 0.7350185360787325)]
```

```
In [19]: #Since we trained the model with the output of word2phrase
         #we can ask for similarity of "phrases",
         #basically compained words such as "Los Angeles"
```

```
In [18]: indexes, metrics = model.similar('los_angeles')
         model.generate_response(indexes, metrics).tolist()
```

```
Out[18]: [('san_francisco', 0.8956202717740596),
          ('san_diego', 0.8720779739303786),
          ('las_vegas', 0.8479347793641956),
          ('miami', 0.8441680994381828),
          ('seattle', 0.8181362149494511),
          ('dallas', 0.8122959413318805),
          ('cincinnati', 0.8112546882982246),
          ('chicago_illinois', 0.8108507423510785),
          ('chicago', 0.8104461988728474),
          ('atlanta', 0.8099587971482523)]
```

```
In [ ]: # anapgoes can be used to find most common pairs to vocab defined
```

```
In [25]: # king woman related but not having to do with man

         indexes, metrics = model.analogy(pos=['king', 'woman'] , neg=['man'])
         model.generate_response(indexes, metrics).tolist()
```

```
Out[25]: [('queen', 0.28827208280388067),
          ('son', 0.27402954107792776),
          ('prince', 0.27114783997707353),
          ('empress', 0.2697223148997432),
          ('wife', 0.26822820479497345),
          ('emperor', 0.26539679464654614),
          ('heir', 0.2634333640987522),
          ('monarch', 0.26227516880664337),
          ('regent', 0.26159308272388376),
          ('throne', 0.26158790871345405)]
```

In [26]:
```
indexes, metrics = model.analogy(pos=['king', 'woman'] , neg=['girl'])
model.generate_response(indexes, metrics).tolist()
```

Out[26]:
```
[('pope', 0.3189770977368626),
 ('monarch', 0.31452438698331164),
 ('emperor', 0.3047052064269137),
 ('crown', 0.29566258002940926),
 ('throne', 0.294397242488326),
 ('bishop', 0.2897745200361802),
 ('ruler', 0.28262608048742144),
 ('vassal', 0.2804152365788582),
 ('papacy', 0.27796519502058353),
 ('sultan', 0.27624097930010416)]
```

Clusters

In [28]:
```
clusters = word2vec.load_clusters('text8-clusters.txt')
```

In [29]:
```
clusters.get_words_on_cluster(90).shape
```

Out[29]:
```
(265,)
```

In [30]:
```
clusters.get_words_on_cluster(90)[:10]
```

Out[30]:
```
array(['making', 'complex', 'physical', 'basic', 'simple', 'alternativ
e',
       'techniques', 'internal', 'advanced', 'extensive'], dtype='<U2
9')
```

In [31]:
```
model.clusters = clusters
```

In [48]:
```
indexes, metrics = model.analogy(pos=["paris", "germany"], neg=["russia
```

In [49]:
```
model.generate_response(indexes, metrics).tolist()
```

Out[49]:
```
[('vienna', 0.3182190817196321, 82),
 ('leipzig', 0.3149545361216135, 41),
 ('berlin', 0.3093381772258632, 20),
 ('munich', 0.2955701688964713, 2),
 ('venice', 0.28612706318526654, 23),
 ('bonn', 0.27919212298639284, 18),
 ('florence', 0.27428765795728605, 89),
 ('milan', 0.2724240128320671, 77),
 ('aachen', 0.2717335462511236, 23),
 ('prague', 0.27077686954053837, 45)]
```

In [ ]:
```

```