Google Colab Link: https://drive.google.com/file/d/1pbUB1ytm4YwqbxfmysS5Wt_WZXWAdANEJ/view?usp=sharing

Lab Exercise 9: Word Embeddings

In this lab exercise, we will create word vectors (embeddings) using word embedding algorithms.

Apply these algorithms for the twitter data (extracted in the earlier exercise) or on any corpus of your choice (https://www.corpusdata.org/formats.asp).

1. Apply the following word embeddings on the concerned corpus:

a. GloVe
b. Word2Vec
c. FastText

You may update the python notebooks shared.
Check the correctness of the model by plugging in word similarities

In [1]:

```
from glove import Corpus, Glove
import re
import glob
from nltk.tokenize import sent_tokenize
import string
import pandas as pd
import codecs
```

In [2]:

```
def preprocess(text):
    text = text.lower()
    text = text.replace('\n',' ')
    text = text.replace("-"," ")
    p = string.punctuation.replace(".","")
    text = text.translate(str.maketrans('', '', p))
```

```python
    lines = sent_tokenize(text)
    lines = list(filter(None, lines))
    return lines
```

In [3]:
```python
file_path = r'G:\spark_big_files\wordLemPoS.txt'
```

In [4]:
```python
import csv
data = pd.read_csv(file_path, encoding = 'unicode_escape', delimiter = "\t", quoting=csv.QUOTE_NONE, engine='c')
```

In [5]:
```python
data.columns
```

Out[5]:
```
Index(['textID', 'ID(seq)', 'word', 'lemma', 'PoS'], dtype='object')
```

In [6]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2286764 entries, 0 to 2286763
Data columns (total 5 columns):
 #   Column   Dtype
---  ------   -----
 0   textID   int64
 1   ID(seq)  int64
 2   word     object
 3   lemma    object
 4   PoS      object
dtypes: int64(2), object(3)
memory usage: 87.2+ MB
```

In [7]:
```python
data[data['word'].isna()]
```

Out[7]:

	textID	ID(seq)	word	lemma	PoS
29212	19514	1694609729	NaN	NaN	NN1
489335	1234514	59402633	NaN	NaN	ZZ1
1267235	10913514	868222484	NaN	NaN	JJ
1770232	25980514	1525068348	NaN	na	FW_FU

In [8]:

```
data.dropna(subset=['word'],inplace=True)
data.dropna(subset=['lemma'],inplace=True)
```

In [9]:

```
data
```

Out[9]:

	textID	ID(seq)	word	lemma	PoS
1	1514	1624977	Albert	albert	NP1
2	1514	1624978	of	of	IO
3	1514	1624979	Prussia	prussia	NP1
5	1514	1624981	17	17	MC
6	1514	1624982	May	may	NPM1
...	...	...	...	...	...
2286758	43534514	2173506790	's	's	GE
2286759	43534514	2173506791	disease	disease	NN1
2286761	43534514	2173506793	He	he	PPHS1
2286762	43534514	2173506794	was	be	VBDZ
2286763	43534514	2173506795	71.	71	MC

1835763 rows × 5 columns

In [10]:

```
grp = data.groupby(data['textID'])['lemma'].apply(list)
sent_list = grp.to_list()
print(sent_list[0][-10:])
```

```
['have', 'two', 'child', 'frederick', '#', 'ancestor', '#', '#', 'note', '#']
```

Glove

In [11]:

```
corpus = Corpus()
corpus.fit(sent_list, window=10)
```

In [12]:
```python
glove = Glove(no_components=25) #size of vectors
```

The glove.fit() takes:

1. cooccurence_matrix: the matrix of word-word co-occurrences
2. epochs: number of times the dataset is processed
3. no_of_threads: number of threads for parallel processing

In [13]:
```python
import time
start = time.time()
glove.fit(corpus.matrix, epochs=50, no_threads=4)## co-occ --> word embeddings
glove.add_dictionary(corpus.dictionary)
glove.save('glove_wiki.model')
end = time.time()
end-start
```

Out[13]:
```
121.55508780479431
```

In [14]:
```python
len(corpus.dictionary)
```

Out[14]:
```
65114
```

In [15]:
```python
glove.word_vectors[glove.dictionary['time']]
```

Out[15]:
```
array([ 0.25882378, -0.01605447, -0.83475089,  0.41125547, -0.15647466,
       -0.4307266 , -0.26203901,  0.4794978 ,  0.85693778, -0.02599827,
       -1.07619821, -0.17676722, -0.34512053,  0.49827201, -0.32458597,
       -0.74334507,  0.2796751 ,  0.58912661, -0.64171156, -0.82027148,
       -0.38205972,  0.3153888 , -0.10029198, -0.04701322,  0.55291504])
```

In [16]:
```python
words = ['art','school','king','code','man','ancient','marry']
for i in range(len(words)):
    print(words[i], end="\t==>  ")
    similar = glove.most_similar(words[i], number=8)
    for j in range(len(similar)):
```

```
        print(similar[j][0],end =", ")
    print("\n")
```

```
art     ==>  contemporary, museum, fine, exhibition, science, society, medical,

school  ==>  high, student, education, attend, graduate, public, secondary,

king    ==>  iii, frederick, knight, henry, duke, prince, stephen,

code    ==>  protocol, foundation, treatment, celtic, urban, regional, salvation,

man     ==>  woman, young, 1253, child, title, person, smiley-faces,

ancient ==>  presentation, municipality, whereas, concept, greek, chain, mass,

marry   ==>  tony, retire, bah'u'llh, father, divorce, she, succeed,
```

Word2Vec

In [17]:

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

In [18]:

```
import time
start = time.time()
cbow = Word2Vec(sent_list, vector_size = 50, window = 5, sg = 0) #sg=0 -CBoW - gensim 4
end = time.time()
end-start
```

Out[18]:

```
10.87619400024414
```

In [19]:

```
cbow.save("cbow_wiki.model")
```

In [20]:

```
cbow2=Word2Vec.load("cbow_wiki.model")
cbow = cbow2
```

In [21]:

```
words = ['art','school','king','code','man','ancient','marry']
for i in range(len(words)):
```

```
    print(words[i], end="\t==>  ")
    similar = cbow.wv.most_similar(words[i], topn = 7)
    for j in range(len(similar)):
        print(similar[j][0],end =", ")
    print("\n")
```

```
art     ==>  science, literature, contemporary, architecture, literary, exhibition, journal,

school  ==>  college, degree, graduate, secondary, boarding, student, attend,

king    ==>  dynasty, duke, count, iii, emperor, lord, prince,

code    ==>  climate, operator, iata, disposal, seychelles, conservation, broad,

man     ==>  woman, girl, boy, contestant, ever, hero, dead,

ancient ==>  modern, origin, medieval, greek, historical, culture, roman,

marry   ==>  daughter, die, henry, elizabeth, succeed, margrave, son,
```

In [22]:

```
##Only Once
start = time.time()
skipgram = Word2Vec(sent_list, vector_size = 50, window = 5, sg = 1) #skipgram
#skipgram = Word2Vec(sent, size = 50, window = 5, sg = 1)
end = time.time()
end-start
```

Out[22]:

```
30.56820583343506
```

In [23]:

```
skipgram.save('skipgram_wiki.model')
skipgram=Word2Vec.load("skipgram_wiki.model")
```

In [24]:

```
words = ['art','school','king','code','man','ancient','marry']
for i in range(len(words)):
    print(words[i], end="\t==>  ")
    similar = skipgram.wv.most_similar(words[i], topn = 7)
    for j in range(len(similar)):
```

```
        print(similar[j][0],end =", ")
    print("\n")
```

```
art     ==>  contemporary, performing, sculpture, architecture, visual, ballet, guild,

school  ==>  elementary, preparatory, surrattsville, grammar, secondary, vocational, college,

king    ==>  duke, iv, augustus, vii, sigismund, bohemia, iii,

code    ==>  seychelles, penal, iata, icao, postal, ethiopia, zip,

man     ==>  woman, discus, mega, jump, horse, individual, hero,

ancient ==>  medieval, norse, inscription, buddha, buddhist, mythology, prehistoric,

marry   ==>  married, daughter, onassis, elizabeth, granddaughter, die, heiress,
```

FastText

In [25]:

```
from gensim.models import FastText
```

In [26]:

```
model = FastText(sent_list, vector_size=50, window=5)
```

In [27]:

```
words = ['art','school','king','code','man','ancient','marry']
for i in range(len(words)):
    print(words[i], end="\t==>  ")
    similar = model.wv.most_similar(words[i], topn = 7)
    for j in range(len(similar)):
        print(similar[j][0],end =", ")
    print("\n")
```

```
art     ==>  artur, arte, museo, musicology, architecture, artwork, arc,

school  ==>  preschool, high-school, schooling, schoolhouse, wool, college, schmidt,

king    ==>  kings, kink, kingman, viking, walking, kingsley, mafeking,

code    ==>  codex, codeine, cocktail, diode, cordata, coded, ode,
```

```
man      ==>  woman, goodman, huffman, chapman, lehman, manx, huaman,

ancient ==>  modernity, scriptural, demonic, morality, monastic, subculture, geologic,

marry   ==>  married, *barry, *harry, garry, harry, barry, marlene,
```