Google Colab Link: https://drive.google.com/file/d/1idW877tYEq4RWAuo4Ktst4VNGoe1nCAg/view?usp=sharing

Lab Exercise 10: Pre-trained Models

In this lab exercise, we will create document vectors from pre-trained models.

1. Download the pre-trained models for the following word embedding models - check the notebooks uploaded.
   a. GloVe
   b. Word2Vec

2. Create document vectors by the following formula: doc_Veci = sum(wj)
   a. doc_Veci : ith document in the corpus.
   b. wj : word vector of jth word in the document. The word vector is taken model.
   c. (Out-Of-Vocabulary) OOV words can be ignored.

3. (Optional) For fastText, download a non-English language. Test few words similarities.

In [28]:

```
import warnings
warnings.simplefilter("ignore", UserWarning)
from glove import Corpus, Glove
import re
import glob
from nltk.tokenize import word_tokenize
import string
import pandas as pd
import codecs
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
from gensim.models import FastText
```

```python
from nltk.corpus import stopwords
import string
```

In [2]:

```python
def preprocess(text):
    text = text.lower()
    text = text.replace('\n',' ')
    text = text.replace("-"," ")
    p = string.punctuation
    text = text.translate(str.maketrans('', '', p))
    printable = set(string.printable)
    text = ''.join(filter(lambda x: x in printable, text))
    lines = word_tokenize(text)
    stop_words = set(stopwords.words('english'))
    filtered_new_toekns = [w for w in lines if not w.lower() in stop_words]
    lines = list(filter(None, filtered_new_toekns))
    return lines
```

In [3]:

```python
document_list = []
with open("AI_corpus.txt") as f:
    document_list = f.readlines()
document_list_tokens = []
for d in document_list:
    filtered_new_toekns = preprocess(d)
    document_list_tokens.append(filtered_new_toekns)
print(document_list_tokens)
```

```
[['supervised', 'learning', 'data', 'point', 'labeled', 'associated', 'category', 'value', 'interest', 'example', 'categorical', 'label', 'as
signing', 'image', 'either', 'cat', 'dog', 'example', 'value', 'label', 'sale', 'price', 'associated', 'used', 'car', 'goal', 'supervised',
'learning', 'study', 'many', 'labeled', 'examples', 'like', 'able', 'make', 'predictions', 'future', 'data', 'points', 'example', 'identifyin
g', 'new', 'photos', 'correct', 'animal', 'assigning', 'accurate', 'sale', 'prices', 'used', 'cars', 'popular', 'useful', 'type', 'machine',
'learning'], ['unsupervised', 'learning', 'data', 'points', 'labels', 'associated', 'instead', 'goal', 'unsupervised', 'learning', 'algorith
m', 'organize', 'data', 'way', 'describe', 'structure', 'unsupervised', 'learning', 'groups', 'data', 'clusters', 'k', 'means', 'finds', 'dif
ferent', 'ways', 'looking', 'complex', 'data', 'appears', 'simpler'], ['reinforcement', 'learning', 'algorithm', 'gets', 'choose', 'action',
'response', 'data', 'point', 'common', 'approach', 'robotics', 'set', 'sensor', 'readings', 'one', 'point', 'time', 'data', 'point', 'algorit
hm', 'must', 'choose', 'robots', 'next', 'action', 'also', 'natural', 'fit', 'internet', 'things', 'applications', 'learning', 'algorithm',
```

```
'also', 'receives', 'reward', 'signal', 'short', 'time', 'later', 'indicating', 'good', 'decision', 'based', 'signal', 'algorithm', 'modifies', 'strategy', 'order', 'achieve', 'highest', 'reward'], ['deep', 'learning', 'subset', 'machine', 'learning', 'thats', 'based', 'artificial', 'neural', 'networks', 'learning', 'process', 'deep', 'structure', 'artificial', 'neural', 'networks', 'consists', 'multiple', 'input', 'output', 'hidden', 'layers', 'layer', 'contains', 'units', 'transform', 'input', 'data', 'information', 'next', 'layer', 'use', 'certain', 'predictive', 'task', 'thanks', 'structure', 'machine', 'learn', 'data', 'processing'], ['training', 'deep', 'learning', 'models', 'often', 'requires', 'large', 'amounts', 'training', 'data', 'high', 'end', 'compute', 'resources', 'gpu', 'tpu', 'longer', 'training', 'time', 'scenarios', 'dont', 'available', 'shortcut', 'training', 'process', 'using', 'technique', 'known', 'transfer', 'learning'], ['machine', 'learning', 'subset', 'artificial', 'intelligence', 'uses', 'techniques', 'deep', 'learning', 'enable', 'machines', 'use', 'experience', 'improve', 'tasks', 'learning', 'process', 'based', 'following', 'steps', '1', 'feed', 'data', 'algorithm', 'use', 'data', 'train', 'model', 'test', 'deploy', 'model', 'consume', 'deployed', 'model', 'automated', 'predictive', 'task', 'words', 'call', 'use', 'deployed', 'model', 'receive', 'predictions', 'returned', 'model'], ['artificial', 'intelligence', 'ai', 'technique', 'enables', 'computers', 'mimic', 'human', 'intelligence', 'includes', 'machine', 'learning'], ['text', 'analytics', 'based', 'deep', 'learning', 'methods', 'involves', 'analyzing', 'large', 'quantities', 'text', 'data', 'example', 'medical', 'documents', 'expenses', 'receipts', 'recognizing', 'patterns', 'creating', 'organized', 'concise', 'information'], ['artificial', 'neural', 'networks', 'formed', 'layers', 'connected', 'nodes', 'deep', 'learning', 'models', 'use', 'neural', 'networks', 'large', 'number', 'layers'], ['feedforward', 'neural', 'network', 'simple', 'type', 'artificial', 'neural', 'network', 'feedforward', 'network', 'information', 'moves', 'one', 'direction', 'input', 'layer', 'output', 'layer', 'feedforward', 'neural', 'networks', 'transform', 'input', 'putting', 'series', 'hidden', 'layers', 'every', 'layer', 'made', 'set', 'neurons', 'layer', 'fully', 'connected', 'neurons', 'layer', 'last', 'fully', 'connected', 'layer', 'output', 'layer', 'represents', 'generated', 'predictions'], ['recurrent', 'neural', 'networks', 'widely', 'used', 'artificial', 'neural', 'network', 'networks', 'save', 'output', 'layer', 'feed', 'back', 'input', 'layer', 'help', 'predict', 'layers', 'outcome', 'recurrent', 'neural', 'networks', 'great', 'learning', 'abilities', 'theyre', 'widely', 'used', 'complex', 'tasks', 'time', 'series', 'forecasting', 'learning', 'handwriting', 'recognizing', 'language'], ['convolutional', 'neural', 'network', 'particularly', 'effective', 'artificial', 'neural', 'network', 'presents', 'unique', 'architecture', 'layers', 'organized', 'three', 'dimensions', 'width', 'height', 'depth', 'neurons', 'one', 'layer', 'connect', 'neurons', 'next', 'layer', 'small', 'region', 'layers', 'neurons', 'final', 'output', 'reduced', 'single', 'vector', 'probability', 'scores', 'organized', 'along', 'depth', 'dimension'], ['generative', 'adversarial', 'networks', 'generative', 'models', 'trained', 'create', 'realistic', 'content', 'images', 'made', 'two', 'networks', 'known', 'generator', 'discriminator', 'networks', 'trained', 'simultaneously', 'training', 'generator', 'uses', 'random', 'noise', 'create', 'new', 'synthetic', 'data', 'closely', 'resembles', 'real', 'data', 'discriminator', 'takes', 'output', 'generator', 'input', 'uses', 'real', 'data', 'determine', 'whether', 'generated', 'content', 'real', 'synthetic', 'network', 'competing', 'generator', 'trying', 'generate', 'synthetic', 'content', 'indistinguishable', 'real', 'content', 'discriminator', 'trying', 'correctly', 'classify', 'inputs', 'real', 'synthetic', 'output', 'used', 'update', 'weights', 'networks', 'help', 'better', 'achieve', 'respective', 'goals'], ['transformers', 'model', 'architecture', 'suited', 'solving', 'problems', 'containing', 'sequences', 'text', 'time', 'series', 'data', 'consist', 'encoder', 'decoder', 'layers', 'encoder', 'takes', 'input', 'maps', 'numerical', 'representation', 'containing', 'information', 'context', 'decoder', 'uses', 'information', 'encoder', 'produce', 'output', 'translated', 'text', 'makes', 'transformers', 'different', 'architectures', 'containing', 'encoders', 'decoders', 'attention', 'sub', 'layers', 'attention', 'idea', 'focusing', 'specific', 'parts', 'input', 'based', 'importance', 'context', 'relation', 'inputs', 'sequence', 'example', 'summarizing', 'news', 'article', 'sentences', 'relevant', 'describe', 'main', 'idea', 'focusing', 'key', 'words', 'throughout', 'article', 'summarization', 'done', 'single', 'sentence', 'headline'], ['automated', 'machine', 'learning', 'also', 'referred', 'automated', 'ml', 'automl', 'process', 'automating', 'time', 'consuming', 'iterative', 'tasks', 'machine', 'learning', 'model', 'development', 'allows', 'data', 'scientists', 'analysts', 'developers', 'build', 'ml', 'models', 'high', 'scale', 'efficiency', 'productivity', 'sustaining', 'model', 'quality']]
```

Glove

In [4]:

```python
import numpy as np
path = "G:\spark_big_files\\"
fn = "glove.6B\glove.6B.50d.txt"
g_file = open(path+fn, encoding="utf-8")
```

```python
model_glove={}
for line in g_file:
    parts = line.split()
    word = parts[0]
    embedding = np.array([float(val) for val in parts[1:]])
    model_glove[word] = embedding
```

In [5]:

```python
model_glove['logistic']
```

Out[5]:

```
array([ 1.3974   , -0.14533  ,  0.0031009,  0.20223  ,  0.10888  ,
       -0.37115  ,  0.81565  , -0.029923 ,  1.1535   , -1.4118   ,
        0.76681  ,  0.21901  , -0.16761  , -0.61186  , -1.5716   ,
       -0.49697  , -0.42153  ,  0.30808  ,  0.59776  , -0.32651  ,
        0.035977 , -0.47536  , -0.62235  , -0.22975  , -0.54818  ,
        0.62333  , -0.41482  , -0.59224  ,  0.68942  ,  0.97883  ,
        1.4524   ,  0.88615  , -0.15822  , -0.36141  ,  0.4336   ,
        1.2251   , -0.15228  ,  0.22974  , -0.32081  ,  0.85588  ,
        0.8408   , -0.26906  ,  0.22466  , -0.4583   , -0.42407  ,
       -0.28703  ,  0.39841  ,  1.2724   ,  0.32464  ,  0.38978  ])
```

In [6]:

```python
with open(path+fn, encoding="utf-8") as f:
    for line in f:
        word, *vector = line.split()
        print(line)
        print(word)
        print(vector)
        break
```

```
the 0.418 0.24968 -0.41242 0.1217 0.34527 -0.044457 -0.49688 -0.17862 -0.00066023 -0.6566 0.27843 -0.14767 -0.55677 0.14658 -0.0095095 0.0116
58 0.10204 -0.12792 -0.8443 -0.12181 -0.016801 -0.33279 -0.1552 -0.23131 -0.19181 -1.8823 -0.76746 0.099051 -0.42125 -0.19526 4.0071 -0.18594
-0.52287 -0.31681 0.00059213 0.0074449 0.17778 -0.15897 0.012041 -0.054223 -0.29871 -0.15749 -0.34758 -0.045637 -0.44251 0.18785 0.0027849 -
0.18411 -0.11514 -0.78581

the
['0.418', '0.24968', '-0.41242', '0.1217', '0.34527', '-0.044457', '-0.49688', '-0.17862', '-0.00066023', '-0.6566', '0.27843', '-0.14767',
'-0.55677', '0.14658', '-0.0095095', '0.011658', '0.10204', '-0.12792', '-0.8443', '-0.12181', '-0.016801', '-0.33279', '-0.1552', '-0.2313
1', '-0.19181', '-1.8823', '-0.76746', '0.099051', '-0.42125', '-0.19526', '4.0071', '-0.18594', '-0.52287', '-0.31681', '0.00059213', '0.007
4449', '0.17778', '-0.15897', '0.012041', '-0.054223', '-0.29871', '-0.15749', '-0.34758', '-0.045637', '-0.44251', '0.18785', '0.0027849',
'-0.18411', '-0.11514', '-0.78581']
```

In [7]:

```python
doc_wv_glove = []
```

```python
for d in document_list_tokens:
    list_wv_glove = []
    for w in d:
        try:
            wv = model_glove[w]
        except Exception as e:
            print(w, " not available in pre-trained model")
            continue
        list_wv_glove.append(list(wv))
    doc_wv_glove.append(list_wv_glove)
print(doc_wv_glove[0][0])
```

```
automl  not available in pre-trained model
[0.38869, -0.6629, -0.26774, -0.73281, -0.34786, -0.10755, -0.41984, 0.068642, 0.27751, -0.16855, 0.3003, -0.32924, 0.018467, 0.21328, 0.3851
1, -0.13537, 0.4288, 0.24974, 0.95216, 0.14078, 0.74774, 0.45853, 0.08173, -0.30055, -0.65179, -0.21313, 0.3142, -0.6973, -0.57879, 0.32603,
1.5068, -0.91617, -1.0854, -0.9522, 0.38002, 0.57362, 0.73149, 0.93065, -0.21461, 0.2159, -0.51239, -0.43315, 0.14123, -0.064009, -0.83072, -
1.1617, -0.27445, 0.25408, -0.68674, 0.48642]
```

In [8]:

```python
sum_doc_wv_glove = []
for doc in doc_wv_glove:
    l = [0]*50
    l = list(map(sum, zip(*doc)))
    sum_doc_wv_glove.append(l)
```

In [9]:

```python
df_glove = pd.DataFrame(sum_doc_wv_glove)
df_glove
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
0	12.101104	4.374460	11.993935	-5.158415	14.766697	11.392416	-20.005113	-29.203114	5.262224	13.857550	...	8.184116	-5.746854	8.520857	18.389585
1	13.159370	3.038464	6.036460	-0.537000	9.670194	3.561177	-7.233797	-12.321832	2.061678	6.368808	...	1.961826	-9.315479	-1.189058	18.020984
2	16.422890	0.116132	11.596643	0.896830	16.635664	3.371303	-0.918356	-14.077522	8.042710	9.784653	...	6.085037	-5.879490	-7.870552	15.872896
3	19.553503	4.763946	14.721760	11.510947	5.782216	15.857914	6.332519	-21.575368	14.119812	6.427718	...	9.855202	-13.177338	-3.793389	21.025968
4	12.320516	2.111690	1.804296	-3.991555	-0.584747	-0.802747	-4.935597	-9.984762	7.997035	2.584212	...	2.946719	-6.756342	2.085694	14.076605

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
5	22.807088	-3.900397	16.441416	-2.635218	1.599845	4.707943	-5.181038	-24.854554	12.571613	0.299524	...	11.309626	-9.596195	4.312019	13.691593
6	4.764424	-1.891360	3.829040	1.878524	1.383179	1.577710	1.407910	-8.010710	3.982017	3.588680	...	3.639742	-1.615224	1.758908	6.101352
7	9.840030	1.926859	0.023040	-0.598059	5.568762	5.264467	-4.900165	-12.576323	9.636665	3.300239	...	8.492799	-4.738453	3.646483	10.930028
8	11.239430	4.968602	7.406865	6.011796	-3.260736	13.061140	0.722734	-9.380602	2.226385	-0.346118	...	1.745879	-4.783351	-1.827847	11.646055
9	19.852918	6.724471	24.297293	21.338611	0.225416	23.502331	14.284390	-17.924339	12.782282	6.788219	...	8.606146	-9.058993	-11.265004	17.786900
10	17.642795	0.854866	9.151886	8.017561	-4.534959	16.691555	0.713951	-14.555997	7.361738	9.843209	...	-0.900690	-6.122775	-8.348763	16.703587
11	18.284240	13.456778	11.535860	11.850854	6.958950	18.814811	14.269279	-16.078769	8.311665	-1.238126	...	-0.804451	-8.650185	-2.633244	10.964907
12	26.904500	-7.925407	28.415070	12.690781	6.427418	13.193917	1.179658	-34.411169	8.271911	39.370404	...	22.780720	-9.241697	-3.214502	22.440325
13	13.264192	7.959192	13.946187	11.420411	16.142531	18.327509	16.158288	-39.025083	6.957113	14.274446	...	27.902520	-1.281909	-7.319446	16.151506
14	12.500854	-4.376278	13.218348	-0.944837	-2.189072	4.623822	-1.292360	-21.266274	7.628481	7.603994	...	0.844215	-9.947371	-1.067155	11.956501

15 rows × 50 columns

In [10]:

```
#df_glove = pd.DataFrame(list_wv_glove)
#df_glove = df_glove.T
#df_glove.columns = filtered_new_toekns
#df_glove['dov_Vec'] = df_glove.sum(axis=1)
#first_column = df_glove.pop('dov_Vec')
#df_glove.insert(0, 'dov_Vec', first_column)
#df_glove.head()
```

Word2Vec

In [11]:

```
%%time
#Only once
word2vec_glove_file = path+'glove.6B\glove.6B.50d.word2vec.txt'
model_word2vec = KeyedVectors.load_word2vec_format(word2vec_glove_file)
model_word2vec.save('glove50_word2vec.model')
```

```
Wall time: 16.9 s
```

In [12]:

```python
model_word2vec = KeyedVectors.load('glove50_word2vec.model')
```

In [13]:

```python
model_word2vec.get_vector('logistic')
```

Out[13]:

```
array([ 1.3974   , -0.14533  ,  0.0031009,  0.20223  ,  0.10888  ,
       -0.37115  ,  0.81565  , -0.029923 ,  1.1535   , -1.4118   ,
        0.76681  ,  0.21901  , -0.16761  , -0.61186  , -1.5716   ,
       -0.49697  , -0.42153  ,  0.30808  ,  0.59776  , -0.32651  ,
        0.035977 , -0.47536  , -0.62235  , -0.22975  , -0.54818  ,
        0.62333  , -0.41482  , -0.59224  ,  0.68942  ,  0.97883  ,
        1.4524   ,  0.88615  , -0.15822  , -0.36141  ,  0.4336   ,
        1.2251   , -0.15228  ,  0.22974  , -0.32081  ,  0.85588  ,
        0.8408   , -0.26906  ,  0.22466  , -0.4583   , -0.42407  ,
       -0.28703  ,  0.39841  ,  1.2724   ,  0.32464  ,  0.38978  ],
      dtype=float32)
```

In [14]:

```python
doc_w2v_glove = []
for d in document_list_tokens:
    list_w2v_glove = []
    for w in d:
        try:
            w2v = model_word2vec.get_vector(w)
        except Exception as e:
            print(w, " not available in pre-trained model")
            continue
        list_w2v_glove.append(list(w2v))
    doc_w2v_glove.append(list_w2v_glove)
print(doc_w2v_glove[0][0])
```

```
automl  not available in pre-trained model
[0.38869, -0.6629, -0.26774, -0.73281, -0.34786, -0.10755, -0.41984, 0.068642, 0.27751, -0.16855, 0.3003, -0.32924, 0.018467, 0.21328, 0.3851
1, -0.13537, 0.4288, 0.24974, 0.95216, 0.14078, 0.74774, 0.45853, 0.08173, -0.30055, -0.65179, -0.21313, 0.3142, -0.6973, -0.57879, 0.32603,
1.5068, -0.91617, -1.0854, -0.9522, 0.38002, 0.57362, 0.73149, 0.93065, -0.21461, 0.2159, -0.51239, -0.43315, 0.14123, -0.064009, -0.83072, -
1.1617, -0.27445, 0.25408, -0.68674, 0.48642]
```

In [15]:

```python
#list_wv_word2vec = []
#for w in filtered_new_toekns:
```

```python
#    wv = model_word2vec.get_vector(w)
#    list_wv_word2vec.append(list(wv))
#df_word2vec = pd.DataFrame(list_wv_word2vec)
#df_word2vec = df_word2vec.T
#df_word2vec.columns = filtered_new_toekns
#df_word2vec['dov_Vec'] = df_word2vec.sum(axis=1)
#first_column = df_word2vec.pop('dov_Vec')
#df_word2vec.insert(0, 'dov_Vec', first_column)
#df_word2vec.head()
sum_doc_w2v_glove = []
for doc in doc_w2v_glove:
    l = [0]*50
    l = list(map(sum, zip(*doc)))
    sum_doc_w2v_glove.append(l)

df_word2vec = pd.DataFrame(sum_doc_w2v_glove)
df_word2vec
```

Out[15]:

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
0	12.101104	4.374460	11.993935	-5.158415	14.766697	11.392416	-20.005113	-29.203115	5.262224	13.857550	...	8.184116	-5.746854	8.520857	18.389585
1	13.159370	3.038464	6.036460	-0.537000	9.670194	3.561177	-7.233797	-12.321832	2.061678	6.368808	...	1.961826	-9.315479	-1.189058	18.020984
2	16.422890	0.116132	11.596643	0.896830	16.635664	3.371303	-0.918356	-14.077522	8.042710	9.784653	...	6.085037	-5.879490	-7.870552	15.872896
3	19.553503	4.763946	14.721760	11.510947	5.782216	15.857914	6.332519	-21.575368	14.119812	6.427718	...	9.855202	-13.177338	-3.793389	21.025968
4	12.320516	2.111690	1.804296	-3.991555	-0.584747	-0.802747	-4.935597	-9.984762	7.997035	2.584212	...	2.946719	-6.756342	2.085694	14.076605
5	22.807088	-3.900397	16.441416	-2.635218	1.599845	4.707943	-5.181038	-24.854554	12.571613	0.299524	...	11.309626	-9.596195	4.312019	13.691593
6	4.764424	-1.891360	3.829040	1.878524	1.383179	1.577710	1.407910	-8.010710	3.982017	3.588680	...	3.639742	-1.615224	1.758908	6.101352
7	9.840030	1.926859	0.023040	-0.598059	5.568762	5.264467	-4.900165	-12.576323	9.636665	3.300239	...	8.492799	-4.738453	3.646483	10.930028
8	11.239430	4.968602	7.406865	6.011796	-3.260736	13.061140	0.722734	-9.380602	2.226385	-0.346118	...	1.745879	-4.783351	-1.827847	11.646055
9	19.852918	6.724471	24.297293	21.338611	0.225416	23.502331	14.284390	-17.924339	12.782282	6.788219	...	8.606146	-9.058993	-11.265004	17.786900
10	17.642795	0.854866	9.151886	8.017561	-4.534959	16.691556	0.713951	-14.555997	7.361738	9.843209	...	-0.900690	-6.122775	-8.348763	16.703587

	0	1	2	3	4	5	6	7	8	9	...	40	41	42	43
11	18.284240	13.456778	11.535860	11.850854	6.958950	18.814811	14.269279	-16.078769	8.311665	-1.238126	...	-0.804451	-8.650185	-2.633244	10.964907
12	26.904500	-7.925407	28.415070	12.690781	6.427418	13.193917	1.179658	-34.411169	8.271911	39.370405	...	22.780720	-9.241697	-3.214502	22.440325
13	13.264192	7.959192	13.946187	11.420411	16.142531	18.327509	16.158288	-39.025083	6.957113	14.274446	...	27.902520	-1.281909	-7.319446	16.151506
14	12.500854	-4.376278	13.218348	-0.944837	-2.189072	4.623822	-1.292360	-21.266274	7.628481	7.603994	...	0.844215	-9.947371	-1.067155	11.956501

15 rows × 50 columns

In [16]:
```
df_word2vec.shape
```

Out[16]:
```
(15, 50)
```

FastText

In [17]:
```
from gensim.test.utils import datapath
from gensim.test.utils import get_tmpfile
```

In [18]:
```
corpus_file = datapath(path+'FastText NBs\sanskrit\cc.sa.300.vec')
```

In [19]:
```
model_fast_text = FastText(window=3, min_count=1)
model_fast_text.build_vocab(corpus_file=corpus_file)
```

In [20]:
```
fname = get_tmpfile("fasttext.model")
model_fast_text.save(fname)
```

In [21]:
```
model_fast_text = FastText.load(fname)
```

In [22]:
```
model_fast_text.wv['तिष्ठत']
```

Out[22]:
```
array([ 1.0589060e-03,  2.3311132e-03,  1.2368697e-03, -5.6811981e-04,
```

```
        1.9011724e-05,  7.9339748e-04, -3.9136652e-03, -1.5236739e-03,
       -6.7840185e-05, -2.5255696e-03,  1.0558382e-03,  6.0761982e-04,
       -1.3082168e-03, -7.8758696e-04,  1.5887890e-03,  9.1723900e-04,
       -2.0566678e-03, -3.2919311e-04, -1.0627214e-03, -2.1360822e-04,
        6.1727501e-04, -1.6938419e-04, -9.5880288e-04, -1.5526972e-03,
        2.2240897e-04, -1.0421999e-03, -1.2042557e-03,  1.1058887e-03,
        6.4493570e-04,  1.6196573e-04,  1.3850558e-03, -4.7308364e-04,
        1.1558679e-03,  6.2220579e-04,  1.7621431e-03,  2.8883219e-03,
        2.1022796e-03,  9.5493335e-05,  1.7296150e-03,  3.5035043e-04,
       -1.9055633e-03, -5.1811081e-04, -6.6972163e-04,  9.9916087e-06,
       -2.4443311e-03,  2.9190325e-03,  1.9951249e-04,  1.6525466e-03,
        1.3114263e-03, -1.9119360e-03, -6.2612497e-04,  2.0028788e-03,
       -6.7243283e-04, -1.7746652e-03,  1.7546925e-03, -5.4100860e-04,
        1.0678092e-03,  7.2115799e-04,  1.2923736e-03,  1.7911096e-03,
        9.8867575e-05,  2.7720253e-03, -4.5825762e-04, -7.7944729e-05,
        3.2207672e-04,  8.6252351e-04, -1.2073311e-03, -1.2600464e-04,
       -3.7638543e-04, -1.7092064e-03,  2.8064859e-04,  2.0554147e-03,
       -2.3641607e-03, -6.8713882e-04, -6.7771022e-04,  2.4890662e-03,
        9.3570584e-04,  6.2817929e-04, -5.6937855e-04,  3.8494830e-04,
       -4.0240097e-03,  4.8071845e-05, -1.0736110e-03,  3.3048175e-03,
       -2.1023052e-04,  1.0820535e-03, -1.8144501e-03, -1.1398678e-03,
        1.7038188e-03, -7.6009909e-04, -1.6537205e-03,  1.2892865e-03,
        6.4688997e-04, -1.1777059e-03, -7.9556368e-04,  8.9986867e-04,
        1.1268969e-03, -3.9036767e-04, -8.8770037e-05, -1.0880316e-04],
      dtype=float32)
```

In [23]:

```
words = ['तिष्ठति']
for i in range(len(words)):
    print(words[i], end="\t==>  ")
    similar = model_fast_text.wv.most_similar(words[i], topn = 100)
    for j in range(len(similar)):
        print(similar[j][0],end =", ")
    print("\n")
```

```
तिष्ठति	==>  उपतिष्ठति, तिष्ठत, प्रतिष्ठति, तिष्ठत्ः, अधितिष्ठति, नावतिष्ठति, तिष्ठत्, अवतिष्ठति, तिष्ठतो, तिष्ठता, यस्तिष्ठति, पर्यवतिष्ठति, तिष्ठतः, तिष्ठतु, प्रतितिष्ठति, तिष्ठतीति, समधितिष्ठति, तथैवतिष्ठति, योऽवतिष्ठति, अनुतिष्ठति, तिष्ठते, नानुतिष्ठति, त्वत्तस्तिष्ठति, पुरस्तिष्ठति, नोत्तिष्ठति, तिष्ठताम्, यदुत्तिष्ठति, यथोक्तमनुतिष्ठति, परितस्तिष्ठति, किमनुतिष्ठति, सज्जस्तिष्ठति, कर्मशून्यस्तिष्ठति, तिष्ठतेः, तिष्ष्ठति, उत्तिष्ठति, यावत्तिष्ठति, स्वविशुद्धस्वरूपेऽवतिष्ठति, प्रत्युत्तिष्ठति, उपतिष्ठतः, शयनादुत्तिष्ठति, अप्रतिष्ठतः, पर्याकुलस्तिष्ठति, आतिष्ठत्, राजोत्तिष्ठति, सम्बन्धस्तिष्ठति, प्रतिष्ठताम्, प्रतिष्ठत, समुपतिष्ठते, प्रतिष्ठता, तिष्ठामि, उपतिष्ठते, तिष्ठस्य, विनोर्ध्वमुत्तिष्ठति, उपातिष्ठत, अतिष्ठत्, उपातिष्ठत्, तिष्ठसि, तिष्ठामः, स्थिरस्तिष्ठति, तिष्ठ, प्रतिष्ठते, मृत्तिकामयूरस्तिष्ठति, अतिष्ठताम्, सभ्यगदित्यमुपतिष्ठते, तिष्ठत्यकर्मकृत्, ब्रह्माधितिष्ठत्, समतिष्ठत्, प्रतिष्ठितनेतारः, प्रतिष्ठिताः, नदीः, सुप्रतिष्ठितः, समाधिरुपतिष्ठते, तिष्यो, अस्वर्गम्, प्रवदन्त्यविपश्चितः, प्रतिष्ठितोऽस्ति, पर्यवतिष्ठते, प्रतिष्ठितानि, सुप्रतिष्ठिता, कर्तव्यनिष्ठिताः, प्रतिष्ठितोऽस्ति, श्रेाः, चावतिष्ठते, पोलियो, श्रेष्ठतमोऽस्ति, तिष्ठेदिति, प्रतिष्ठितः, कर्तव्यनिष्ठतायाः, सहसोदतिष्ठत्, नावतिष्ठते, नातिष्ठत्, परिमाणादसावधिकोऽस्ति, सन्तिष्ठते, head, तिष्ठेयम्, अनुतिष्ठतः, पादाःसम्मधिपादः, घनिष्ठतमः, रवेदार, जागतिकैः,
```

In [24]:

```
sim_score = model_fast_text.wv.similarity('तिष्ठत', 'अधितिष्ठति')
print('तिष्ठत and ', 'अधितिष्ठति', " = ", sim_score)
```

```
sim_score = model_fast_text.wv.similarity('तिष्ठत', 'नावतिष्ठति')
print('तिष्ठत and ', 'नावतिष्ठति', " = ", sim_score)
```

```
तिष्ठत and  अधितिष्ठति  =  0.48580906
तिष्ठत and  नावतिष्ठति  =  0.4587452
```

In [25]:

```
new_document_sanskrit_list = ["यथैतानि विशिष्टानि जात्यां जात्यां वृकोदर", "तन्तिपाल इति ख्यातो नाम्ना विदितमस्तु ते", "एवमेतन्महाबाहो यथा स भगवान्प्रभुः", "सदा क्षुतं च वातं च ष्ठीवनं चाचरेच्छनैः", "एवमुक्तस्ततो राज्ञा धौम्योऽथ द्विजसत्तमः",
                              "तथैतान्पातयिष्यामि यथा यास्यन्ति न क्षयम्", "ृतिज्ञां षण्ढकोऽस्मीति करिष्यामि महीपते", "यच्च भर्तानुयुञ्जीत तदेवाभ्यनुवर्तयेत्", "अन्यस्मिन्प्रेष्यमाणे तु पुरस्ताद्यः समुत्पतेत्", "क्रायुधानि समासज्य प्रवेक्ष्यामः पुरं वयम्",
                              "बद्धगोधाङ्गुलित्राणाः कालिन्दीमभितो ययुः", "अनुशिष्टाः स्म भद्रं ते नैतद्वक्तास्ति कश्चन", "श्रेयः सदात्मनो दृष्ट्वा परं राज्ञा न संवदेत्", "कस्तस्य मनसापीच्छेदनर्थं प्राज्ञसंमतः", "अनुकूलो भवेच्चास्य सर्वार्थेषु कथासु च"]
document_list_tokens_sanskrit = []
for d in new_document_sanskrit_list:
    filtered_new_toekns = d.split(' ')
    document_list_tokens_sanskrit.append(filtered_new_toekns)
print(document_list_tokens_sanskrit)
```

```
[['यथैतानि', 'विशिष्टानि', 'जात्यां', 'जात्यां', 'वृकोदर'], ['तन्तिपाल', 'इति', 'ख्यातो', 'नाम्ना', 'विदितमस्तु', 'ते'], ['एवमेतन्महाबाहो', 'यथा', 'स', 'भगवान्प्रभुः'], ['सदा', 'क्षुतं', 'च', 'वातं', 'च', 'ष्ठीवनं', 'चाचरेच्छनैः'], ['एवमुक्तस्ततो', 'राज्ञा', 'धौम्योऽथ', 'द्विजसत्तमः'], ['तथैतान्पातयिष्यामि', 'यथा', 'यास्यन्ति', 'न', 'क्षयम्'], ['ृतिज्ञां', 'षण्ढकोऽस्मीति', 'करिष्यामि', 'महीपते'], ['यच्च', 'भर्तानुयुञ्जीत', 'तदेवाभ्यनुवर्तयेत्'], ['अन्यस्मिन्प्रेष्यमाणे', 'तु', 'पुरस्ताद्यः', 'समुत्पतेत्'], ['क्रायुधानि', 'समासज्य', 'प्रवेक्ष्यामः', 'पुरं', 'वयम्'], ['बद्धगोधाङ्गुलित्राणाः', 'कालिन्दीमभितो', 'ययुः'], ['अनुशिष्टाः', 'स्म', 'भद्रं', 'ते', 'नैतद्वक्तास्ति', 'कश्चन'], ['श्रेयः', 'सदात्मनो', 'दृष्ट्वा', 'परं', 'राज्ञा', 'न', 'संवदेत्'], ['कस्तस्य', 'मनसापीच्छेदनर्थं', 'प्राज्ञसंमतः'], ['अनुकूलो', 'भवेच्चास्य', 'सर्वार्थेषु', 'कथासु', 'च']]
```

In [26]:

```
#list_wv_fastText = []
#for w in filtered_new_toekns_sanskrit:
#    wv = model_fast_text.wv[w]
#    list_wv_fastText.append(list(wv))

#df_fastText = pd.DataFrame(list_wv_fastText)
#df_fastText = df_fastText.T
#df_fastText.columns = filtered_new_toekns_sanskrit
#df_fastText['dov_Vec'] = df_fastText.sum(axis=1)
#first_column = df_fastText.pop('dov_Vec')
#df_fastText.insert(0, 'dov_Vec', first_column)
```

```
#df_fastText.head()


doc_fastText_sanskrit = []
for d in document_list_tokens:
    list_fastText_sanskrit = []
    for w in d:
        try:
            s_vec = model_fast_text.wv[w]
        except Exception as e:
            print(w, " not available in pre-trained model")
            continue
        list_fastText_sanskrit.append(list(s_vec))
    doc_fastText_sanskrit.append(list_fastText_sanskrit)
print(doc_fastText_sanskrit[0][0])
```

```
[-0.0009527995, -8.804027e-05, 0.0009892479, -0.0003704267, 0.0006932408, 0.0009536738, 0.0007528288, -0.0013706625, -0.00077991746, 0.000951
03896, -0.00044674528, -0.0004967448, 0.0019182435, -0.00085308694, -0.00017296587, 0.00042593974, -0.0004441517, -0.0010759244, -0.000662766
16, -0.00067316095, 0.00056647765, -0.0014014114, 0.000988022, 0.00025770074, 0.0023065438, 3.0562755e-05, -0.00025505826, -0.0015365044, -0.
0007660524, -0.0005755527, 0.0006458222, 0.00035174677, -0.0009029893, -0.0011454197, 0.0011482273, -0.00038082743, 0.00029289786, -0.0002704
8143, 0.0016051602, 0.00053378876, -0.0009835734, 0.00057035836, -0.0005911308, -0.0017213552, 0.00067084713, 0.0005278702, -4.115245e-05, 0.
0006972772, 0.000642024, -0.00037818606, -0.00062766945, -0.0006355875, -0.0009206987, 0.0009033149, 0.0005846131, 0.0011772407, 0.001595686
7, -0.0005367631, -0.002192059, -0.0003436258, 0.00019506218, 0.0012934171, 0.0008296852, -0.00036039238, 0.00042453996, -0.00044631417, 0.00
06095406, 0.0003739861, 0.00032086964, 0.00081291434, 0.001009208, -0.0005448937, -0.0024045631, -0.0021047168, -0.0001739436, -3.6310845e-0
5, -0.0013692296, 0.0021046132, 0.0010130139, 0.00039858106, -0.00031125668, -0.0008717673, 0.00067347003, -0.0019555, 0.00017445923, 0.00020
668749, 0.0011603308, 0.0014312329, 0.0014757626, -0.0012484716, -0.0009742587, -0.00064827903, 0.0022910319, 1.0857957e-05, 0.0007939271, -
0.00045857407, -0.00011124509, -0.0022032738, -0.0014112333, 0.00010914261]
```

In [27]:

```
sum_doc_fastText_sanskrit = []
for doc in doc_fastText_sanskrit:
    l = [0]*50
    l = list(map(sum, zip(*doc)))
    sum_doc_fastText_sanskrit.append(l)

df_fastText_sanskrit = pd.DataFrame(sum_doc_fastText_sanskrit)
df_fastText_sanskrit
```

Out[27]:

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94

	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93	94
0	-0.004782	0.025757	-0.013505	-0.001444	-0.008244	-0.020082	-0.001362	-0.010701	0.029740	-0.016900	...	-0.006636	-0.015261	-0.007624	0.001961	-0.020096
1	0.001806	-0.000180	-0.010639	-0.002363	0.001843	-0.005168	-0.001772	-0.008569	0.012185	-0.009944	...	0.026366	0.004711	-0.014789	-0.007607	-0.001361
2	0.005629	0.011824	0.005164	0.011099	-0.021485	-0.010498	0.007124	0.017100	0.029697	-0.007888	...	-0.027472	0.006225	-0.032664	-0.015042	-0.007745
3	-0.008209	0.016274	0.001326	0.008406	-0.015311	-0.000690	-0.007610	-0.007799	0.001651	-0.010490	...	0.023103	0.010331	-0.027182	0.015567	0.005497
4	-0.001768	0.002920	-0.011951	-0.000104	-0.011378	0.006874	-0.010834	-0.002150	-0.005779	-0.000514	...	-0.016875	-0.005024	0.004225	0.012178	-0.009797
5	-0.014022	-0.010216	-0.003657	-0.016005	-0.004598	0.011711	-0.006307	0.001205	0.014059	0.000473	...	0.003456	0.021676	-0.028886	0.023999	-0.017386
6	-0.004802	0.003829	0.004152	-0.007269	-0.000218	-0.002605	-0.004873	0.000380	0.001809	-0.000099	...	-0.007424	-0.003221	0.002436	-0.004582	0.002408
7	-0.003524	-0.001668	-0.011860	-0.002639	0.006061	-0.000403	0.016374	-0.000862	0.010125	-0.010359	...	0.003498	0.006846	-0.003140	0.000796	0.006829
8	-0.010301	0.011073	0.000279	-0.000136	-0.007597	0.007614	-0.004365	0.001407	-0.001337	-0.006448	...	-0.001829	0.002193	-0.004662	0.008821	0.006471
9	-0.029451	0.025667	0.008558	0.013790	-0.006232	-0.014753	-0.013256	-0.002057	0.016449	0.002467	...	0.007265	0.015723	-0.054601	0.014858	0.022098
10	-0.011766	-0.001325	0.009882	0.005051	-0.005197	-0.000145	0.007991	0.017023	-0.000228	0.009955	...	-0.009088	0.012368	-0.025528	0.006299	0.011829
11	-0.017896	0.011651	0.011171	0.005331	-0.000002	0.004772	-0.007014	-0.013918	0.000284	0.014109	...	-0.011071	0.010431	-0.028047	-0.005674	0.004549
12	-0.022391	0.018524	0.019718	-0.040489	0.020233	0.013534	0.001084	0.002147	0.039319	-0.019687	...	-0.021535	-0.008808	-0.032174	-0.001503	0.006515
13	-0.020882	0.033106	0.005284	0.026708	0.018098	0.014738	-0.000414	-0.037031	0.020591	-0.018923	...	0.007289	0.002370	-0.022399	0.031443	-0.006315
14	-0.012251	-0.010717	-0.009857	0.006058	-0.006042	0.007698	-0.004003	-0.004103	-0.000220	-0.003288	...	-0.018464	0.005216	0.007212	0.000753	-0.011329

15 rows × 100 columns