Practical No. 1

Perform tokenization (Whitespace, Punctuation-based, Treebank, Tweet, Multi-Word Expression - MWE) using NLTK library.

```
!pip install nltk
```

```
Looking in indexes:https://pypi.org/simple,https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.8/dist-packages (3.7)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.8/dist-packages (from nltk) (2022.6.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from nltk) (4.64.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from nltk) (7.1.2)
```

Whitespace based Tokenization

Syntax : tokenize.WhitespaceTokenizer()

Return : Return the tokens from a string

```
# import WhitespaceTokenizer() method from nltk
from nltk.tokenize import WhitespaceTokenizer
# Create a reference variable for Class WhitespaceTokenizer
wt = WhitespaceTokenizer()
# Create a string input
text = "Welcome to the I2IT-NLP Page. \n Good Morning \t"
print("\nOriginal string:")
print(text)
# Use tokenize method
tokenized_text = wt.tokenize(text)
print("\nSplitting using whitespece into separate tokens:")
print(tokenized_text)
```

```
Original string:
Welcome to the I2IT-NLP Page.
```

Good Morning

Splitting using whitespece into separate tokens:

['Welcome', 'to', 'the', 'I2IT-NLP', 'Page.', 'Good', 'Morning']

Punctuation-based tokenizer

```
from nltk.tokenize import WordPunctTokenizer
text = "Welcome to the I2IT-NLP Page. \n Good Morning \t"
print("\nOriginal string:")
print(text)
result = WordPunctTokenizer().tokenize(text)
print("\nSplit all punctuation into separate tokens:")
print(result)
```

Original string:

Welcome to the I2IT-NLP Page.

Good Morning

Split all punctuation into separate tokens:

['Welcome', 'to', 'the', 'I2IT', '-', 'NLP', 'Page', '.', 'Good', 'Morning']

Treebank Tokenizer

```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
tokenizer.tokenize(text)
```

['Welcome', 'to', 'the', 'I2IT-NLP', 'Page.', 'Good', 'Morning']

Tweet Tokenizer

When we want to apply tokenization in text data like tweets, the tokenizers mentioned above can’t produce practical tokens. Through this issue, NLTK has a rule based tokenizer special for tweets. We can split emojis into different words if we need them for tasks like sentiment analysis.

```
from nltk.tokenize import TweetTokenizer
tweet_tokenize = TweetTokenizer()
sample_tweet = "Who is your favourite cryptocurrency influencer? 🤔🏆 Tag them below! 👇"
```

```
print(tweet_tokenize.tokenize(sample_tweet))
```

['Who', 'is', 'your', 'favourite', 'cryptocurrency', 'influencer', '?', '🗣', '🏆', 'Tag', 'them', 'below', '!', '👇']

Multi-Word Expression Tokenizer

A MWETokenizer takes a string which has already been divided into tokens and retokenizes it, merging multi-word expressions into single tokens, using a lexicon of MWEs.

The multi-word expression tokenizer is a rule-based, “add-on” tokenizer offered by NLTK. Once the text has been tokenized by a tokenizer of choice, some tokens can be re-grouped into multi-word expressions.

For example, the name Steven Spielberg is combined into a single token instead of being broken into two tokens. This tokenizer is very flexible since it is agnostic of the base tokenizer that was used to generate the tokens.

```
# import MWETokenizer() method from nltk
from nltk.tokenize import MWETokenizer
#from nltk.tokenize import  word_tokenize
# Create a reference variable for Class MWETokenizer
tokenizer = MWETokenizer([('a', 'little'), ('a', 'little', 'bit'), ('a', 'lot')])
tokenizer.add_mwe(('in', 'spite', 'of'))
tokenizer.tokenize('In a little or a little bit or a lot in spite of'.split())
['In', 'a_little', 'or', 'a_little_bit', 'or', 'a_lot', 'in_spite_of']
from nltk.tokenize import MWETokenizer
tokenizer = MWETokenizer()
tokenizer.add_mwe(('Steven', 'Spielberg'))
tokenizer.tokenize('Steven Spielberg is an American writer producer director '.split())
['Steven_Spielberg', 'is', 'an', 'American', 'writer', 'producer', 'director']
```

Porter Stemmer

```
import nltk
from nltk.stem import PorterStemmer
ps = PorterStemmer()
```

```
example_words =["connector","connection","connects","connecting","connected"]
#Next, we can easily stem by doing something like:
for w in example_words:
  print(ps.stem(w))
connector
connect
connect
connect
connect
```

Snowball Stemmer

```
from nltk.stem import SnowballStemmer
snowball = SnowballStemmer(language='english')
words = ['generous','generate','generously','generation']
for word in words:
    print(word,"--->",snowball.stem(word))
generous ---> generous
generate ---> generat
generously ---> generous
generation ---> generat
```

Lancaster Stemmer

```
from nltk.stem import LancasterStemmer
lancaster = LancasterStemmer()
words = ['eating','eats','eaten','puts','putting']
for word in words:
    print(word,"--->",lancaster.stem(word))
eating ---> eat
eats ---> eat
eaten ---> eat
puts ---> put
```

putting ---> put

Regex Stemmer

```
from nltk.stem import RegexpStemmer
regexp = RegexpStemmer('ing|ï¿½|e|ï¿½ï¿½ï¿½ï¿½', min=4)
words = ['mass','was','bee','computer','advisable']
for word in words:
    print(word,"--->",regexp.stem(word))
```

mass ---> mas

was ---> was

bee ---> bee

computer ---> computer

advisable ---> advis

Porter Vs Snowball Vs Lancaster Vs Regex Stemmers

```
from nltk.stem import PorterStemmer, SnowballStemmer, LancasterStemmer, RegexpStemmer
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer(language='english')
regexp = RegexpStemmer('ing|ï¿½|e|ï¿½ï¿½ï¿½ï¿½', min=4)
word_list = ['generous','generate','generously','generation']
print("{0:20}{1:20}{2:20}{3:30}{4:40}".format("Word","Porter Stemmer","Snowball Stemmer","Lancaster Stemmer",'Regexp Stemmer'))
for word in word_list:
print("{0:20}{1:20}{2:20}{3:30}{4:40}".format(word,porter.stem(word),snowball.stem(word),lancaster.stem(word),regexp.stem(word)))
```

Word	Porter Stemmer	Snowball Stemmer	Lancaster Stemmer	Regexp Stemmer
generous	gener	generous	gen	generou
generate	gener	generat	gen	generat
generously	gener	generous	gen	generously
generation	gener	generat	gen	generation

Lemmatization

Using NLTK Library for Lemmatization

```
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
True
import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()
sentence = "He was running and eating at same time. He has bad habit of swimming after playing long hours in the Sun."
punctuations="?:!.,;"
sentence_words = nltk.word_tokenize(sentence)
for word in sentence_words:
    if word in punctuations:
        sentence_words.remove(word)
sentence_words
print("{0:20}{1:20}".format("Word","Lemma"))
for word in sentence_words:
    print ("{0:20}{1:20}".format(word,wordnet_lemmatizer.lemmatize(word)))
Word                 Lemma
He                   He
was                  wa
running              running
and                  and
```

eating eating
at at
same same
time time
He He
has ha
bad bad
habit habit
of of
swimming swimming
after after
playing playing
long long
hours hour
in in
the the
Sun Sun

Practical No. 2

Bag of Words Algorithm Implementation

```
''' vectorize() function takes list of words in a sentence as input
    and returns a vector of size of filtered_vocab.It puts 0 if the
    word is not present in tokens and count of token if present.'''
def vectorize(tokens):
    vector=[]
    for w in filtered_vocab:
        vector.append(tokens.count(w))
    return vector
'''unique() functions returns a list in which the order remains
    same and no item repeats.Using the set() function does not
    preserve the original ordering,so i didnt use that instead'''
def unique(sequence):
    seen = set()
    return [x for x in sequence if not (x in seen or seen.add(x))]
#create a list of stopwords.You can import stopwords from nltk too
stopwords=["to","was","a"]
#list of special characters.You can use regular expressions too
special_char=[",",":"," ",";",".","?"]
string1 = "Data science is fun and interesting"
string2 = "Data science is fun"
string3 = "science is interesting"
#converting strings to lower case
string1=string1.lower()
string2=string2.lower()
string3=string3.lower()
#split the sentences into tokens
tokens1=string1.split()
tokens2=string2.split()
```

```
tokens3=string3.split()
print(tokens1)
print(tokens2)
print(tokens3)
#create a vocabulary list
vocab=unique(tokens1+tokens2+tokens3)
print(vocab)
#filter the vocabulary list
filtered_vocab=[]
for w in vocab:
    if w not in stopwords and w not in special_char:
        filtered_vocab.append(w)
print("Final filtered vocabulary: ", filtered_vocab)
#convert sentences into vectords
vector1=vectorize(tokens1)
print("Sentence 1 vector :",vector1)
vector2=vectorize(tokens2)
print("Sentence 2 vector :",vector2)
vector3=vectorize(tokens3)
print("Sentence 3 vector :",vector2)
['data', 'science', 'is', 'fun', 'and', 'interesting']
['data', 'science', 'is', 'fun']
['science', 'is', 'interesting']
['data', 'science', 'is', 'fun', 'and', 'interesting']
Final filtered vocabulary:  ['data', 'science', 'is', 'fun', 'and', 'interesting']
Sentence 1 vector : [1, 1, 1, 1, 1, 1]
Sentence 2 vector : [1, 1, 1, 1, 0, 0]
Sentence 3 vector : [1, 1, 1, 1, 0, 0]
```

Creating Bag of Words using sklearn library

```
import pandas as pd
```

```
from sklearn.feature_extraction.text import CountVectorizer
string1 = "Data science is fun and interesting"
string2 = "Data science is fun"
string3 = "science is interesting"
doc = string1+string2+string3
CountVec = CountVectorizer(ngram_range=(1,1))
#transform
Count_data = CountVec.fit_transform([string1,string2,string3])
#create dataframe
cv_dataframe=pd.DataFrame(Count_data.toarray(),columns=CountVec.get_feature_names())
print(cv_dataframe)
```

	and	data	fun	interesting	is	science
0	1	1	1	1	1	1
1	0	1	1	0	1	1
2	0	0	0	1	1	1

```
/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function get_feature_names is deprecated; get_feature_names is deprecated in 1.0 and will be removed in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Count Occurrence

```
count_vec = CountVectorizer()
count_occurs = count_vec.fit_transform([doc])
count_occur_df = pd.DataFrame((count, word) for word, count in zip(count_occurs.toarray().tolist()[0], count_vec.get_feature_names_out()))
count_occur_df.columns = ['Word', 'Count']
count_occur_df.sort_values('Count', ascending=False, inplace=True)
count_occur_df.head()
```

	Word	Count
6	is	3
7	science	2
0	and	1
1	data	1
2	fun	1

Normalized Count Occurrence

```
from sklearn.feature_extraction.text import TfidfVectorizer
norm_count_vec = TfidfVectorizer(use_idf=False, norm='l2')
norm_count_occurs = norm_count_vec.fit_transform([doc])
norm_count_occur_df = pd.DataFrame((count, word) for word, count in zip(
norm_count_occurs.toarray().tolist()[0], norm_count_vec.get_feature_names_out()))
norm_count_occur_df.columns = ['Word', 'Count']
norm_count_occur_df.sort_values('Count', ascending=False, inplace=True)
norm_count_occur_df.head()
```

	Word	Count
6	is	0.688247
7	science	0.458831
0	and	0.229416
1	data	0.229416
2	fun	0.229416

Practical No. 3

TF-IDF Algorithm Implementation

```
import pandas as pd
import numpy as np
corpus = ['Natural language processing is fun and interesting', 'Natural language processing is fun',
'Hindi language is interesting' ]

#creating a word set for the corpus
words_set = set()
or doc in  corpus:
    words = doc.split(' ')
    words_set = words_set.union(set(words))
print('Number of words in the corpus:',len(words_set))
print('The words in the corpus: \n', words_set)
Number of words in the corpus: 8
The words in the corpus:
 {'fun', 'is', 'Natural', 'and', 'interesting', 'Hindi', 'language', 'processing'}
Computing Term Frequency
#creating a dataframe by the number of documents in the corpus and the word set, and use that
information to compute the term frequency (TF)
n_docs = len(corpus)         #·Number of documents in the corpus
n_words_set = len(words_set) #·Number of unique words in the
df_tf = pd.DataFrame(np.zeros((n_docs, n_words_set)), columns=words_set)
# Compute Term Frequency (TF)
for i in range(n_docs):
    words = corpus[i].split(' ') # Words in the document
    for w in words:
        df_tf[w][i] = df_tf[w][i] + (1 / len(words))
df_tf
```

	fun	is	Natural	and	interesting	Hindi	language	processing
0	0.142857	0.142857	0.142857	0.142857	0.142857	0.00	0.142857	0.142857
1	0.200000	0.200000	0.200000	0.000000	0.000000	0.00	0.200000	0.200000
2	0.000000	0.250000	0.000000	0.000000	0.250000	0.25	0.250000	0.000000

Computing Inverse Document Frequency

```
print("IDF of: ")
idf = {}
for w in words_set:
    k = 0    # number of documents in the corpus that contain this word
    for i in range(n_docs):
        if w in corpus[i].split():
            k += 1
    idf[w] =  np.log10(n_docs / k)
    print(f'{w:>15}: {idf[w]:>10}' )
```

```
IDF of:
            fun: 0.17609125905568124
             is:        0.0
        Natural: 0.17609125905568124
            and: 0.47712125471966244
    interesting: 0.17609125905568124
          Hindi: 0.47712125471966244
       language:        0.0
     processing: 0.17609125905568124
```

Note the IDF value of the word "is"

Computing TF-IDF

```
df_tf_idf = df_tf.copy()
for w in words_set:
    for i in range(n_docs):
        df_tf_idf[w][i] = df_tf[w][i] * idf[w]
df_tf_idf
```

	fun	is	Natural	and	interesting	Hindi	language	processing
0	0.025156	0.0	0.025156	0.06816	0.025156	0.00000	0.0	0.025156
1	0.035218	0.0	0.035218	0.00000	0.000000	0.00000	0.0	0.035218
2	0.000000	0.0	0.000000	0.00000	0.044023	0.11928	0.0	0.000000

F-IDF using sklearn library

```
from sklearn.feature_extraction.text import TfidfVectorizer
tr_idf_model  = TfidfVectorizer()
tf_idf_vector = tr_idf_model.fit_transform(corpus)
tf_idf_array = tf_idf_vector.toarray()
print(tf_idf_array)
[[0.49929819 0.37972915 0.         0.37972915 0.29489356 0.29489356
  0.37972915 0.37972915]
 [0.         0.48759135 0.         0.         0.37865818 0.37865818
  0.48759135 0.48759135]
 [0.         0.         0.66283998 0.50410689 0.39148397 0.39148397
  0.         0.        ]]
words_set = tr_idf_model.get_feature_names_out()
print(words_set)
['and' 'fun' 'hindi' 'interesting' 'is' 'language' 'natural' 'processing']
```

df_tf_idf = pd.DataFrame(tf_idf_array, columns = words_set)

df_tf_idf

	and	fun	hindi	interesting	is	language	natural	processing
0	0.499298	0.379729	0.00000	0.379729	0.294894	0.294894	0.379729	0.379729
1	0.000000	0.487591	0.00000	0.000000	0.378658	0.378658	0.487591	0.487591
2	0.000000	0.000000	0.66284	0.504107	0.391484	0.391484	0.000000	0.000000

Practical No. 4

word2vec Implementation using Gensim Library

```
#Importing dependancies
from gensim.models import Word2Vec, FastText
import pandas as pd
import re
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt
import plotly.graph_objects as go
import numpy as np
import warnings
warnings.filterwarnings('ignore')
sentences = [['Natural', 'language', 'processing', 'is', 'fun', 'and', 'interesting'],
        ['Natural', 'language', 'processing', 'is', 'fun'],
        ['Hindi', 'language', 'is', 'interesting']]
# train word2vec model
#w2v =Word2Vec(sentences, size=2, alpha=0.025)
w2v = Word2Vec(sentences, min_count=1, size = 2)
print(w2v)
```

WARNING:gensim.models.base_any2vec:consider setting layer size to a multiple of 4 for greater performance

WARNING:gensim.models.base_any2vec:under 10 jobs per worker: consider setting a smaller `batch_words' for smoother alpha decay

```
Word2Vec(vocab=8, size=2, alpha=0.025)
```

When printed, the model displays the count of unique vocab words, array size and learning rate (default .025)

```
# access vector for one word
print(w2v['language'])
#list the vocabulary words
words = list(w2v.wv.vocab)
```

```
print(words)

#or show the dictionary of vocab words
w2v.wv.vocab
[-0.06693748 -0.03051319]
['Natural', 'language', 'processing', 'is', 'fun', 'and', 'interesting', 'Hindi']
'Natural': <gensim.models.keyedvectors.Vocab at 0x7f6644f314f0>,
 'language': <gensim.models.keyedvectors.Vocab at 0x7f6644f31790>,
 'processing': <gensim.models.keyedvectors.Vocab at 0x7f6644f31490>,
 'is': <gensim.models.keyedvectors.Vocab at 0x7f6644f316a0>,
 'fun': <gensim.models.keyedvectors.Vocab at 0x7f6644f31820>,
 'and': <gensim.models.keyedvectors.Vocab at 0x7f6644f315e0>,
 'interesting': <gensim.models.keyedvectors.Vocab at 0x7f6644f31e20>,
 'Hindi': <gensim.models.keyedvectors.Vocab at 0x7f6644f31910>}
```

VISUALIZE EMBEDDINGS

```
X = w2v[w2v.wv.vocab]
pca = PCA(n_components=2)
result = pca.fit_transform(X)
# create a scatter plot of the projection
plt.scatter(result[:, 0], result[:, 1])
words = list(w2v.wv.vocab)
for i, word in enumerate(words):
  plt.annotate(word, xy=(result[i, 0], result[i, 1]))
plt.show()
```

Our corpus is tiny so it is easy to visualize; however, it's hard to decipher any meaning from the plotted points since the model had so little information from which to learn.

\

Visualizing Email Dataset Word Embeddings

df = pd.read_csv('/content/drive/MyDrive/ACADEMIC YEAR 2022-23 SEMESTER II/LP-VI Lab Assignments/NLP/emails.csv')

df.head()

	text	spam
0	Subject: naturally irresistible your corporate...	1
1	Subject: the stock trading gunslinger fanny i...	1
2	Subject: unbelievable new homes made easy im ...	1
3	Subject: 4 color printing special request add...	1
4	Subject: do not have money , get software cds ...	1

Data Cleaning

```
clean_txt = []
for w in range(len(df.text)):
  desc = df['text'][w].lower()
  #remove punctuation
  desc = re.sub('[^a-zA-Z]', ' ', desc)
  #remove tags
  desc=re.sub("</?.*?>"," <> ",desc)
  #remove digits and special chars
  desc=re.sub("(\d|\W)+"," ",desc)
  clean_txt.append(desc)
df['clean'] = clean_txt
df.head()
```

	text	spam	clean
0	Subject: naturally irresistible your corporate...	1	subject naturally irresistible your corporate ...
1	Subject: the stock trading gunslinger fanny i...	1	subject the stock trading gunslinger fanny is ...
2	Subject: unbelievable new homes made easy im ...	1	subject unbelievable new homes made easy im wa...
3	Subject: 4 color printing special request add...	1	subject color printing special request additio...
4	Subject: do not have money , get software cds ...	1	subject do not have money get software cds fro...

Creating Corpus and Vector

```
corpus = []
for col in df.clean:
  word_list = col.split(" ")
  corpus.append(word_list)
#show first value
print(corpus[0:1])
print(len(corpus))
#generate vectors from corpus
model = Word2Vec(corpus, min_count=1, size = 56)
```

[['subject', 'naturally', 'irresistible', 'your', 'corporate', 'identity', 'lt', 'is', 'really', 'hard', 'to', 'recollect', 'a', 'company', 'the', 'market', 'is', 'full', 'of', 'suqgestions', 'and', 'the', 'information', 'isoverwhelminq', 'but', 'a', 'good', 'catchy', 'logo', 'stylish', 'statlonery', 'and', 'outstanding', 'website', 'will', 'make', 'the', 'task', 'much', 'easier', 'we', 'do', 'not', 'promise', 'that', 'havinq', 'ordered', 'a', 'iogo', 'your', 'company',

'will', 'automaticaily', 'become', 'a', 'world', 'ieader', 'it', 'isguite', 'ciear', 'that', 'without', 'good', 'products', 'effective', 'business', 'organization', 'and', 'practicable', 'aim', 'it', 'will', 'be', 'hotat', 'nowadays', 'market', 'but', 'we', 'do', 'promise', 'that', 'your', 'marketing', 'efforts', 'will', 'become', 'much', 'more', 'effective', 'here', 'is', 'the', 'list', 'of', 'clear', 'benefits', 'creativeness', 'hand', 'made', 'original', 'logos', 'specially', 'done', 'to', 'reflect', 'your', 'distinctive', 'company', 'image', 'convenience', 'logo', 'and', 'stationery', 'are', 'provided', 'in', 'all', 'formats', 'easy', 'to', 'use', 'content', 'management', 'system', 'letsyou', 'change', 'your', 'website', 'content', 'and', 'even', 'its', 'structure', 'promptness', 'you', 'will', 'see', 'logo', 'drafts', 'within', 'three', 'business', 'days', 'affordability', 'your', 'marketing', 'break', 'through', 'shouldn', 't', 'make', 'gaps', 'in', 'your', 'budget', 'satisfaction', 'guaranteed', 'we', 'provide', 'unlimited', 'amount', 'of', 'changes', 'with', 'no', 'extra', 'fees', 'for', 'you', 'to', 'be', 'surethat', 'you', 'will', 'love', 'the', 'result', 'of', 'this', 'collaboration', 'have', 'a', 'look', 'at', 'our', 'portfolio', 'not', 'interested', '']]

5728

The data has been tokenized and is ready to be vectorized!

Visualizing Email Dataset Word Vectors

```
#pass the embeddings to PCA
X = model[model.wv.vocab]
words_list = list(model.wv.vocab)
print(len(words_list))
pca = PCA(n_components=2)
result = pca.fit_transform(X)
#create df from the pca results
pca_df = pd.DataFrame(result, columns = ['x','y'])
#add the words for the hover effect
pca_df['word'] = words_list
pca_df.head()
pca_df
```

	x	y	word
0	0.457135	2.498262	subject
1	0.365060	0.109267	naturally
2	-0.222138	-0.221582	irresistible
3	6.177703	-5.798908	your
4	2.348782	2.179319	corporate
...	...	...	...
33710	-0.348704	-0.236174	selectable
33711	-0.311514	-0.279343	hypercube
33712	-0.299495	-0.292167	toughest
33713	-0.311940	-0.271894	critic
33714	-0.290639	-0.243245	desiring

33715

33715 rows × 3 columns

```
N = 1000000
words = list(model.wv.vocab)
fig = go.Figure(data=go.Scattergl(
   x = pca_df['x'],
   y = pca_df['y'],
   mode='markers',
   marker=dict(
      color=np.random.randn(N),
      colorscale='Viridis',
```

```
        line_width=1
    ),
    text=pca_df['word'],
    textposition="bottom center"
))
fig.show()
```

Output hidden; open in https://colab.research.google.com to view.

```
model.wv.most_similar('joe')
```

```
[('scott', 0.9339468479156494),
 ('nelson', 0.928282618522644),
 ('eric', 0.9210140109062195),
 ('kara', 0.9144555330276489),
 ('mrha', 0.9133355617523193),
 ('bennett', 0.9132177233695984),
 ('palmer', 0.9118150472640991),
 ('sally', 0.9114452004432678),
 ('lee', 0.9108055830001831),
 ('alison', 0.910140335559845)]
```

```
model.wv.most_similar_cosmul(positive = ['phone', 'number'], negative = ['call'])
```

```
[('cognizant', 1.060425877571106),
 ('address', 1.039358139038086),
 ('buiness', 1.038673758506775),
 ('patolia', 1.0339504480361938),
 ('courier', 1.0248805284500122),
 ('name', 1.0128659009933472),
 ('nondisclosure', 1.011989951133728),
 ('inboxes', 1.00129234790802),
 ('jisao', 0.9969928860664368),
 ('zip', 0.993704617023468)]
```