```
Bia Chaudhry

BESE -7B

184055

Natural Language Processing Project

Submitted to

Ma'am Seemab Latif
```

COMPARISON BETWEEN COSINE SIMILARITY & SOFT COSINE SIMILARITY

Installing packages and Importing Libraries

```
!pip install beautifulsoup4
```

```
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.6/dist-packages (4.6.3)
```

```
!pip install lxml
```

```
Requirement already satisfied: lxml in /usr/local/lib/python3.6/dist-packages (4.2.6)
```

```
import bs4 as bs
import urllib.request
import re
import nltk, string, numpy as np
```

```
nltk.download('wordnet') # first-time use only
nltk.download('punkt')
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
import math
import pandas as pd
import networkx as nx
from networkx.generators.small import krackhardt_kite_graph
from string import ascii_lowercase
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Unzipping corpora/wordnet.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Extract Abstract from a webpage

```
def abstractExtracter(url_to_go):
  scraped_data = urllib.request.urlopen(url_to_go)
  article = scraped_data.read()

  parsed_article = bs.BeautifulSoup(article,'lxml')

  paragraphs = parsed_article.find_all(id='p-2')

  article_text = ""

  for p in paragraphs:
      article_text += p.text
  return article_text

# COVID-19 related
abstract1 = abstractExtracter('https://www.medrxiv.org/content/10.1101/2020.02.24.20027052v1')
abstract2 = abstractExtracter('https://www.medrxiv.org/content/10.1101/2020.03.07.20031575v1')
abstract3 = abstractExtracter('https://www.medrxiv.org/content/10.1101/2020.03.19.20034124v1')

# Movie Characters
```

```
# Movie characters
abstract4 = "Harry Potter is a series of fantasy novels written by British author J. K. Rowling. The novels chronicle the lives of a youn
abstract5 = "Cruise began acting in the early 1980s and made his breakthrough with leading roles in the comedy film Risky Business (1983)
abstract6 = "Spider-Man is a fictional superhero created by writer-editor Stan Lee and writer-artist Steve Ditko. He first appeared in th

# Sports related
abstract7 = "Cricket is a bat-and-ball game played between two teams of eleven players on a field at the centre of which is a 20-metre (2
abstract8 = "Roller hockey, also known as quad hockey, international-style ball hockey, rink hockey and Hoquei em Patins, is an overarchi
```

```
print('ABSTRACT # 1: ', abstract1)

print('\n-----------\n')

print('ABSTRACT # 2: ',abstract2)

print('\n-----------\n')

print('ABSTRACT # 3: ',abstract3)

print('\n-----------\n')

print('ABSTRACT # 4: ', abstract4)

print('\n-----------\n')

print('ABSTRACT # 5: ', abstract5)

print('\n-----------\n')

print('ABSTRACT # 6: ', abstract6)

print('\n-----------\n')

print('ABSTRACT # 7: ', abstract7)

print('\n-----------\n')
```

```
print('ABSTRACT # 8: ', abstract8)
```

```
ABSTRACT # 1:  Objective: To investigate the correlation between clinical characteristics and cardiac injury of COVID-2019 pneumonia
```

Creating Documents

```
def fileWriter(filename, content):
  abs_file = open(filename, mode='w+')
  abs_file.write(content)

fileWriter('doc3.txt', abstract3)
```

```
ABSTRACT # 3:  Background
```

```
d1 = abstract1
d2 = abstract2
d3 = abstract3
d4 = abstract4
d5 = abstract5
d6 = abstract6
d7 = abstract7
d8 = abstract8



documents = [d1, d2, d3, d4, d5, d6, d7, d8]
```

Data Preprocessing

```
lemmer = nltk.stem.WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

```
ABSTRACT # 8:  Roller hockey, also known as quad hockey, international-style ball hockey, rink hockey and Hoquei em Patins, is an ov
```

```
LemVectorizer = CountVectorizer(tokenizer=LemNormalize, stop_words='english')
LemVectorizer.fit_transform(documents)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_extraction/text.py:385: UserWarning: Your stop_words may be inconsistent with
  'stop_words.' % sorted(inconsistent))
<8x544 sparse matrix of type '<class 'numpy.int64'>'
	with 624 stored elements in Compressed Sparse Row format>
```

```
LemVectorizer.vocabulary_
```

{'0025': 0,
 '0158': 1,
 '03': 2,
 '0987': 3,
 '10': 4,
 '102': 5,
 '1187': 6,
 '1206': 7,
 '13': 8,
 '15': 9,
 '1548': 10,
 '160': 11,
 '1912': 12,
 '1962': 13,
 '1980s': 14,
 '1983': 15,
 '1986': 16,
 '1988': 17,
 '1989': 18,
 '1990s': 19,
 '1992': 20,
 '1993': 21,
 '1994': 22,
 '1996': 23,
 '2': 24,
 '20': 25,
 '2019': 26,
 '2020': 27,
 '20metre': 28,
 '21': 29,
 '22yard': 30,
 '2350': 31,
 '244': 32,
 '3': 33,
 '3079': 34,
 '32': 35,
 '4': 36,
 '40fold': 37,
 '41': 38,
 '425625': 39,
 '4580': 40,
 '48': 41,
 '488': 42,

'50316': 43,
'525': 44,
'54': 45,
'64': 46,
'6th': 47,
'732': 48,
'7805': 49,
'84': 50,
'8864': 51,
'95': 52,
'975': 53,
'ability': 54,
'abnormal': 55,
'abnormality': 56,
'academy': 57,
'acclaim': 58,
'accompanied': 59,
'according': 60,
'acquiring': 61,
'act': 62,
'acting': 63,
'action': 64,
'actor': 65,
'acute': 66,
'adaptation': 67,
'addition': 68,
'adipose': 69,
'adjudicated': 70,
'adjusting': 71,
'adolescence': 72,
'adult': 73,
'age': 74,
'aided': 75,
'alias': 76,
'altitude': 77,
'amazing': 78,
'american': 79,
'analysis': 80,
'analyzed': 81,
'anthology': 82,
'appeared': 83,
'appears': 84,
'applying': 85,
'arc': 86

```
'arc': 86,
'atrial': 87,
'august': 88,
'aunt': 89,
'author': 90,
'award': 91,
'b': 92,
'background': 93,
'bail': 94,
'balanced': 95,
'ball': 96,
'barcelona': 97,
'bat': 98,
'batandball': 99,
'batting': 100,
'began': 101,
'beijing': 102,
'ben': 103,
'best': 104,
'bite': 105,
'blood': 106,
'body': 107,
'book': 108,
'born': 109,
'bowled': 110,
'bowling': 111,
'bpm': 112,
'breakthrough': 113,
'british': 114,
'business': 115,
'cad': 116,
'came': 117,
'cardiac': 118,
'case': 119,
'catching': 120,
'caused': 121,
'centre': 122,
'character': 123,
'characteristic': 124,
'chest': 125,
'china': 126,
'chronicle': 127,
'ci': 128,
'city': 129,
```

```
'clinging': 130,
'clinical': 131,
'clinically': 132,
'closely': 133,
'cofounder': 134,
'collected': 135,
'color': 136,
'comedy': 137,
'comic': 138,
'commercially': 139,
'commission': 140,
'common': 141,
'communicate': 142,
'complication': 143,
'comprising': 144,
'computed': 145,
'concern': 146,
'conclusion': 147,
'condition': 148,
'consecutive': 149,
'consolidation': 150,
'continuous': 151,
'corona': 152,
'coronavirus': 153,
'correlated': 154,
'correlation': 155,
'country': 156,
'covid19': 157,
'covid2019': 158,
'crash': 159,
'created': 160,
'cricket': 161,
'critical': 162,
'cruise': 163,
'ct': 164,
'cycle': 165,
'danger': 166,
'dark': 167,
'data': 168,
'deal': 169,
'death': 170,
'declared': 171,
'demographic': 172,
'demonstrate': 173,
```

'demonstrate': 173,
'demonstration': 174,
'density': 175,
'despite': 176,
'detecting': 177,
'device': 178,
'diabetes': 179,
'diagnosed': 180,
'different': 181,
'disease': 182,
'dislodges': 183,
'dismiss': 184,
'dismissal': 185,
'dismissed': 186,
'distinguished': 187,
'ditko': 188,
'doctor': 189,
'dominate': 190,
'drama': 191,
'duration': 192,
'early': 193,
'eat': 194,
'ecg': 195,
'electrocardiogram': 196,
'em': 197,
'emergency': 198,
'end': 199,
'enrolled': 200,
'enzyme': 201,
'epicardial': 202,
'epidemic': 203,
'epidemiologic': 204,
'existed': 205,
'existing': 206,
'extrinsic': 207,
'factor': 208,
'fantasy': 209,
'fatality': 210,
'feb': 211,
'fibrillation': 212,
'fictional': 213,
'field': 214,
'fielding': 215,
'film': 216,

'financial': 217,
'finding': 218,
'firm': 219,
'foe': 220,
'following': 221,
'fourth': 222,
'friend': 223,
'function': 224,
'game': 225,
'gene': 226,
'glass': 227,
'globe': 228,
'goblin': 229,
'golden': 230,
'good': 231,
'governing': 232,
'granger': 233,
'green': 234,
'ground': 235,
'group': 236,
'guidance': 237,
'gun': 238,
'gwen': 239,
'ha': 240,
'hallmark': 241,
'harry': 242,
'harrys': 243,
'health': 244,
'heart': 245,
'hermione': 246,
'high': 247,
'higher': 248,
'highlighted': 249,
'hit': 250,
'hockey': 251,
'hogwarts': 252,
'hollywood': 253,
'hoquei': 254,
'horror': 255,
'hospital': 256,
'hospitalized': 257,
'hypertension': 258,
'identified': 259,
'identify': 260

'identity': 260,
'imaging': 261,
'immortal': 262,
'incidence': 263,
'include': 264,
'included': 265,
'including': 266,
'independent': 267,
'index': 268,
'indicated': 269,
'individual': 270,
'ineligible': 271,
'inflammation': 272,
'information': 273,
'inhospital': 274,
'initially': 275,
'injury': 276,
'inline': 277,
'inning': 278,
'intends': 279,
'interlobular': 280,
'international': 281,
'internationalstyle': 282,
'interpretation': 283,
'intervention': 284,
'interview': 285,
'intrinsic': 286,
'invented': 287,
'investigate': 288,
'involved': 289,
'issue': 290,
'issued': 291,
'j': 292,
'jameson': 293,
'jan': 294,
'jane': 295,
'january': 296,
'jerry': 297,
'jonah': 298,
'july': 299,
'k': 300,
'key': 301,
'keyindependent': 302,
'killed': 303,

'known': 304,
'kovic': 305,
'laboratory': 306,
'leading': 307,
'lee': 308,
'level': 309,
'life': 310,
'light': 311,
'likely': 312,
'linked': 313,
'load': 314,
'logistic': 315,
'long': 316,
'lord': 317,
'low': 318,
'magic': 319,
'maguire': 320,
'main': 321,
'man': 322,
'march': 323,
'marvel': 324,
'mary': 325,
'match': 326,
'max': 327,
'mean': 328,
'medical': 329,
'mellites': 330,
'men': 331,
'method': 332,
'mild': 333,
'ministry': 334,
'mode': 335,
'modell': 336,
'money': 337,
'monitoring': 338,
'movie': 339,
'muggles': 340,
'multivariable': 341,
'multivariate': 342,
'myocardial': 343,
'myocarditis': 344,
'mysterious': 345,
'n': 346,

'national': 347,
'natriuretic': 348,
'neglected': 349,
'new': 350,
'nomination': 351,
'nonmagical': 352,
'nonsevere': 353,
'normal': 354,
'noted': 355,
'notwithstanding': 356,
'novel': 357,
'ntprobnp': 358,
'number': 359,
'objective': 360,
'octopus': 361,
'offfield': 362,
'olympics': 363,
'onset': 364,
'opacity': 365,
'orf1ab': 366,
'origin': 367,
'orphan': 368,
'osborn': 369,
'outbreak': 370,
'outcome': 371,
'overarching': 372,
'overthrow': 373,
'oxygen': 374,
'p0001': 375,
'p0008': 376,
'p0012': 377,
'p0048': 378,
'p005': 379,
'p005typical': 380,
'parent': 381,
'parker': 382,
'patchy': 383,
'patient': 384,
'patins': 385,
'pcr': 386,
'peak': 387,
'people': 388,
'peptide': 389,
'peter': 390,

'pgml': 391,
'pii': 392,
'pitch': 393,
'plane': 394,
'played': 395,
'player': 396,
'pneumonia': 397,
'portrayal': 398,
'potential': 399,
'potter': 400,
'predicted': 401,
'predictor': 402,
'presented': 403,
'pressure': 404,
'prevent': 405,
'pro': 406,
'process': 407,
'prognostic': 408,
'public': 409,
'published': 410,
'pulmonary': 411,
'quad': 412,
'radioactive': 413,
'radiologic': 414,
'rain': 415,
'raised': 416,
'rare': 417,
'rate': 418,
'reached': 419,
'realtime': 420,
'received': 421,
'record': 422,
'referee': 423,
'regardless': 424,
'region': 425,
'regress': 426,
'regression': 427,
'related': 428,
'relatively': 429,
'remain': 430,
'reported': 431,
'respiratory': 432,
'result': 433,

```
'results of': 434,
'retrospective': 435,
'review': 436,
'richard': 437,
'rink': 438,
'risk': 439,
'risky': 440,
'role': 441,
'roller': 442,
'romance': 443,
'romantic': 444,
'ron': 445,
'rowling': 446,
'rtpcr': 447,
'run': 448,
'sample': 449,
'sarscov2': 450,
'saturation': 451,
'scan': 452,
'school': 453,
'score': 454,
'scorer': 455,
'screening': 456,
'second': 457,
'separate': 458,
'septal': 459,
'series': 460,
'serum': 461,
'set': 462,
'severe': 463,
'severity': 464,
'shadow': 465,
'shooting': 466,
'showed': 467,
'significantly': 468,
'silver': 469,
'singlecenter': 470,
'skate': 471,
'spider': 472,
'spiderman': 473,
'spiderrelated': 474,
'spidersense': 475,
'spiderwebs': 476,
'sport': 477,
```

```
 'stacy': 478,
 'stage': 479,
 'stan': 480,
 'star': 481,
 'starred': 482,
 'statistical': 483,
 'steve': 484,
 'story': 485,
 'striking': 486,
 'struggle': 487,
 'student': 488,
 'study': 489,
 'stump': 490,
 'subjugate': 491,
 'successful': 492,
 'summer': 493,
 'superhero': 494,
 'supporting': 495,
 'surface': 496,
 'swap': 497,
 'syndrome': 498,
 'team': 499,
 'television': 500,
 'terminal': 501,
 'thickening': 502,
 'threshold': 503,
 'thriller': 504,
 'time': 505,
 'tissue': 506,
 'tni': 507,
 'tomographic': 508,
 'total': 509,
```

TF_IDF Calculations

```
tf_matrix = LemVectorizer.transform(documents).toarray()
print (tf_matrix)
```

```
[[0 0 1 ... 0 1 0]
 [0 0 0 ... 0 0 0]
 [1 1 0 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
```

```
tf_matrix.shape
```

```
(8, 544)
```

```
'weasley': 550,
```

```
tfidfTran = TfidfTransformer(norm="l2")
tfidfTran.fit(tf_matrix)
print (tfidfTran.idf_)
```

```
[2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 1.81093022 2.5040774  2.09861229 2.09861229 2.5040774  2.09861229
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.09861229 2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.09861229 2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.09861229 1.81093022
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.09861229
 2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.09861229 2.5040774  2.5040774  2.5040774  2.5040774
 1.81093022 2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.09861229 2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.09861229 2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  1.81093022 2.09861229
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.09861229 2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
 2.5040774  2.09861229 2.09861229 2.5040774  2.5040774  2.5040774
 1.58778666 2.5040774  2.09861229 2.5040774  1.81093022 2.5040774
 2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
```

```
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.09861229 2.5040774  1.81093022 2.09861229 2.5040774  2.5040774
  2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
  2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.09861229
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.09861229 2.5040774  2.09861229 2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.09861229 2.5040774
  2.5040774  2.5040774  2.5040774  2.09861229 2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.09861229 2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.09861229 2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.09861229 2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774  2.5040774
```

```
def idf(n,df):
    result = math.log((n+1.0)/(df+1.0)) + 1
    return result
print( "The idf for terms that appear in one document: " + str(idf(4,1)))
print( "The idf for terms that appear in two documents: " + str(idf(4,2)))
```

```
The idf for terms that appear in one document: 1.916290731874155
The idf for terms that appear in two documents: 1.5108256237659907
```

```
tfidf_matrix = tfidfTran.transform(tf_matrix)
print (tfidf_matrix.toarray())
```

```
[[0.         0.         0.05690053 ... 0.         0.05690053 0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.04549598 0.04549598 0.         ... 0.         0.         0.        ]
 ...
 [0.         0.         0.         ... 0.08517796 0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

Cosine Similarity Matrix

```
cos_similarity_matrix = (tfidf_matrix * tfidf_matrix.T).toarray()
print (cos_similarity_matrix)
```

```
[[1.         0.31494383 0.22431601 0.         0.01490693 0.00194864
  0.00384484 0.00839603]
 [0.31494383 1.         0.27191561 0.00848904 0.         0.00338107
  0.00667113 0.01942379]
 [0.22431601 0.27191561 1.         0.         0.01260042 0.00816564
  0.00614843 0.01118869]
 [0.         0.00848904 0.         1.         0.0071785  0.05126799
  0.         0.01051885]
 [0.01490693 0.         0.01260042 0.0071785  1.         0.
  0.01692371 0.00717347]
 [0.00194864 0.00338107 0.00816564 0.05126799 0.         1.
  0.01726673 0.00837903]
 [0.00384484 0.00667113 0.00614843 0.         0.01692371 0.01726673
  1.         0.03306504]
 [0.00839603 0.01942379 0.01118869 0.01051885 0.00717347 0.00837903
  0.03306504 1.        ]]
```

```
df = pd.DataFrame(cos_similarity_matrix, index= ['doc1','doc2','doc3', 'doc4', 'doc5', 'doc6', 'doc7', 'doc8'], columns=['doc1','doc2','d
df
```

	doc1	doc2	doc3	doc4	doc5	doc6	doc7	doc8
doc1	1.000000	0.314944	0.224316	0.000000	0.014907	0.001949	0.003845	0.008396

```
df.to_csv('sim_matrix.csv')
```

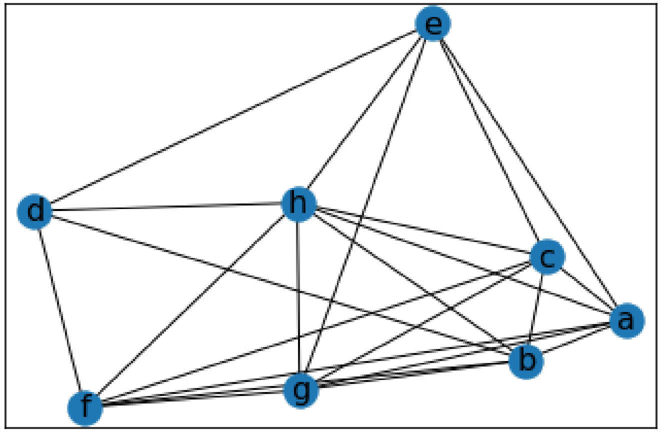
```
cos_similarity_matrix
```

```
array([[1.        , 0.31494383, 0.22431601, 0.        , 0.01490693,
        0.00194864, 0.00384484, 0.00839603],
       [0.31494383, 1.        , 0.27191561, 0.00848904, 0.        ,
        0.00338107, 0.00667113, 0.01942379],
       [0.22431601, 0.27191561, 1.        , 0.        , 0.01260042,
        0.00816564, 0.00614843, 0.01118869],
       [0.        , 0.00848904, 0.        , 1.        , 0.0071785 ,
        0.05126799, 0.        , 0.01051885],
       [0.01490693, 0.        , 0.01260042, 0.0071785 , 1.        ,
        0.        , 0.01692371, 0.00717347],
       [0.00194864, 0.00338107, 0.00816564, 0.05126799, 0.        ,
        1.        , 0.01726673, 0.00837903],
       [0.00384484, 0.00667113, 0.00614843, 0.        , 0.01692371,
        0.01726673, 1.        , 0.03306504],
       [0.00839603, 0.01942379, 0.01118869, 0.01051885, 0.00717347,
        0.00837903, 0.03306504, 1.        ]])
```

Similarity Network Graph

```
G = nx.from_numpy_matrix(np.array(cos_similarity_matrix))
pos=nx.spring_layout(G)
labels = {}
for idx, node in enumerate(G.nodes()):
    labels[node] = ascii_lowercase[idx]
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos, labels, font_size=16)
```

```
{0: Text(0.841745585842018, -0.1437313721044677, 'a'),
 1: Text(0.5297857290918704, -0.2615199885583736, 'b'),
 2: Text(0.5934057698571252, 0.044067664877l0522, 'c'),
 3: Text(-0.9999999999999999, 0.17826968106296096, 'd'),
 4: Text(0.2360313503960871, 0.7331077256377565, 'e'),
 5: Text(-0.8442270937553488, -0.40162029201579447, 'f'),
 6: Text(-0.17459786712837505, -0.3511607037451265, 'g'),
 7: Text(-0.18214347430337682, 0.20258728484593966, 'h')}
```



Soft Cosine Similarity

Importing Libraries and Downloading FastText pre-trained embeddings

```
#SOFT COSINE

import gensim
# upgrade gensim if you can't import softcossim
from gensim.matutils import softcossim
from gensim import corpora
import gensim.downloader as api
from gensim.utils import simple_preprocess
print(gensim.__version__)
#> '3.6.0'
```

```
# Download the FastText model
fasttext_model300 = api.load('fasttext-wiki-news-subwords-300')
```

Preparing Documents for Similarity Calculations

```
# Prepare a dictionary and a corpus.
dictionary = corpora.Dictionary([simple_preprocess(doc) for doc in documents])

# Prepare the similarity matrix
similarity_matrix = fasttext_model300.similarity_matrix(dictionary, tfidf=None, threshold=0.0, exponent=2.0, nonzero_limit=100)

# Convert the sentences into bag-of-words vectors.
sent_1 = dictionary.doc2bow(simple_preprocess(d1))
sent_2 = dictionary.doc2bow(simple_preprocess(d2))
sent_3 = dictionary.doc2bow(simple_preprocess(d3))
sent_4 = dictionary.doc2bow(simple_preprocess(d4))
sent_5 = dictionary.doc2bow(simple_preprocess(d5))
sent_6 = dictionary.doc2bow(simple_preprocess(d6))
sent_7 = dictionary.doc2bow(simple_preprocess(d7))
sent_8 = dictionary.doc2bow(simple_preprocess(d8))

sentences = [sent_1, sent_2, sent_3, sent_4, sent_5, sent_6, sent_7, sent_8]
```

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion of the second argument of issubdtype from `
  if np.issubdtype(vec.dtype, np.int):
```

Soft Cosine Similarity Matrix

```
import numpy as np
import pandas as pd

def create_soft_cossim_matrix(sentences):
    len_array = np.arange(len(sentences))
    xx, yy = np.meshgrid(len_array, len_array)
```

```
    xx, yy = np.meshgrid(len_array, len_array)
    cossim_mat = pd.DataFrame([[round(softcossim(sentences[i],sentences[j], similarity_matrix) ,2) for i, j in zip(x,y)] for y, x in zip(
    return cossim_mat

soft = create_soft_cossim_matrix(sentences)
```

```
soft
```

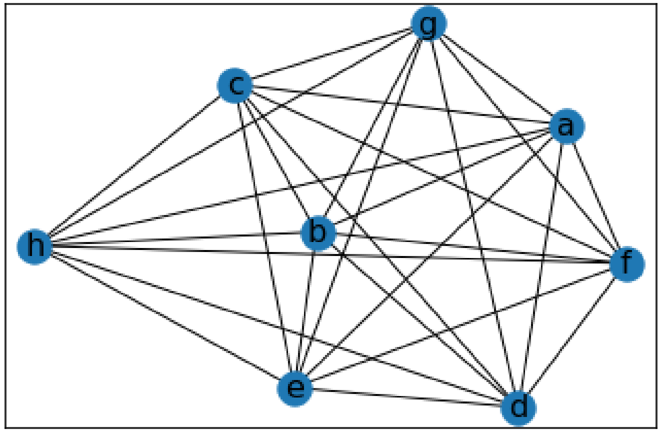
	0	1	2	3	4	5	6	7
0	1.00	0.86	0.86	0.65	0.75	0.73	0.69	0.40
1	0.86	1.00	0.78	0.57	0.68	0.65	0.65	0.40
2	0.86	0.78	1.00	0.65	0.71	0.72	0.69	0.42
3	0.65	0.57	0.65	1.00	0.65	0.70	0.64	0.34
4	0.75	0.68	0.71	0.65	1.00	0.78	0.75	0.38
5	0.73	0.65	0.72	0.70	0.78	1.00	0.74	0.43
6	0.69	0.65	0.69	0.64	0.75	0.74	1.00	0.46
7	0.40	0.40	0.42	0.34	0.38	0.43	0.46	1.00

Soft Cosine Similarity Network Graph

```
import networkx as nx
import numpy as np

from networkx.generators.small import krackhardt_kite_graph
from string import ascii_lowercase
G = nx.from_numpy_matrix(np.array(soft))
pos=nx.spring_layout(G)
labels = {}
for idx, node in enumerate(G.nodes()):
    labels[node] = ascii_lowercase[idx]
nx.draw_networkx_nodes(G, pos)
```

```
nx.draw_networkx_nodes(G, pos)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos, labels, font_size=16)
```

```
{0: Text(0.5493962762436561, 0.35848933862869, 'a'),
 1: Text(-0.17672259010990607, -0.04027416548680531, 'b'),
 2: Text(-0.41582659408412376, 0.5109643940084911, 'c'),
 3: Text(0.4121130346081521, -0.6954231904792567, 'd'),
 4: Text(-0.24189609010584057, -0.6216772584907388, 'e'),
 5: Text(0.7260350879452805, -0.15844942825209118, 'f'),
 6: Text(0.14690087550278308, 0.7391572228312548, 'g'),
 7: Text(-1.0, -0.09278691275954416, 'h')}
```



Evaluations and Results

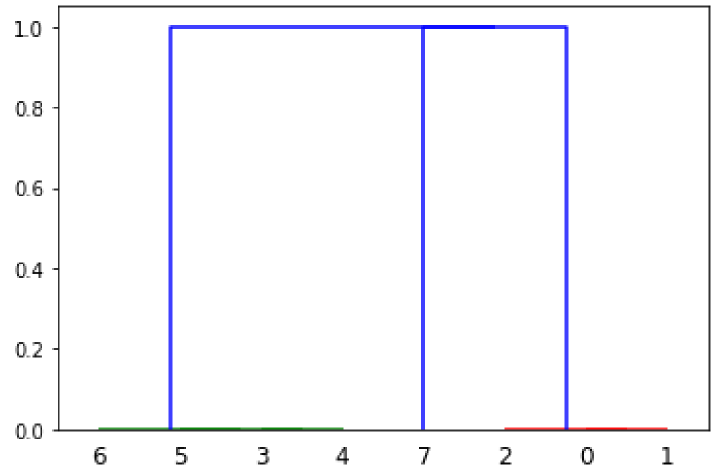
```
from sklearn.cluster import AgglomerativeClustering
import numpy as np
X = np.array(soft) # soft can be replaced with cos_similarity_matrix to generate its Dendogram and clusters
clustering = AgglomerativeClustering( n_clusters=3).fit(X)

AgglomerativeClustering(n_clusters=3)
clustering.labels_
```

```
array([2, 2, 2, 0, 0, 0, 0, 1])
```

```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt
X = [[i] for i in clustering.labels_]
X
```

```
[[2], [2], [2], [0], [0], [0], [0], [1]]
```

```
Z = linkage(X, 'single')
fig = plt.figure()
dn = dendrogram(Z)
```



Accuracy

```
Cos_Clusters_Predicted =  [[1], [1], [1], [2], [0], [2], [0], [1]]
Soft_Clusters_Predicted = [[2], [2], [2], [0], [0], [0], [0], [1]]

Cos_Clusters_Actual =  [[1], [1], [1], [2], [2], [2], [0], [0]]
Soft_Clusters_Actual = [[2], [2], [2], [0], [0], [0], [1], [1]]
```

Accuracy or Correct Rate

$$CR = \frac{C}{A}$$

CR – The correct rate;

C – The number of sample recognized correctly;

A – The number of all sample;

```
Soft_Accuracy = 7/8 * 100
Cos_Accuracy = 7/8 * 100
```

```
print("Soft Cosine Accuracy: ", Soft_Accuracy)

print("Cosine Accuracy: ", Cos_Accuracy)
```

```
Soft Cosine Accuracy:  87.5
Cosine Accuracy:  87.5
```

```
-- END ---
```