Google Colab Link: https://drive.google.com/file/d/1EVJHM7NFvSosJJY3_G-M2A5PmENKgHIF/view?usp=sharing

Lab Exercise 6 : Basic Vectorization

1. For the text in the file given LabE6.txt(1000 reviews from IMDB dataset):

a. Apply preprocessing.
b. Create one-hot encoded vectors for the each tokens in the vocabulary.

In [1]:
```
import sys
import nltk
from nltk import word_tokenize
import re
import string
import pandas as pd
import numpy as np
import spacy
from nltk.corpus import stopwords
np.random.seed(50)
```

In [2]:
```
data_path = "C:\spark\MCA\Semester1\E3_NLP\input\lab_e6\LabE6.txt"
text = ""
with open (data_path, "r") as f: #
    text = f.read()
```

In [3]:
```
def pre_processing(txt):
    txt = txt.encode('unicode_escape').decode('utf-8') #many characters(like Ã©, Ã-, Ã³,...) are wrongly read in python so converting them back to original character
    txt = str(txt.lower())
    txt = re.sub(r'http\S+', '', txt) #remove URLs
    txt = txt.replace(r"''", "\"") #replacing 2 consequtive single quotes to double quote
```

```
    txt = txt.replace(r".", " ") #removing all period "."
    txt = txt.replace(r"<br />", " ")
    txt= txt.translate(str.maketrans('','',string.punctuation.replace("'",""))) #removing punctuations except '
    txt = txt.translate(str.maketrans('','','\t\n'))
    txt_list = txt.split(' ')
    txt_list = [word for word in txt_list if not word in set(stopwords.words('english')) ]
    txt = ' '.join(txt_list)
    return txt
```

In [4]:
```
text = pre_processing(text)
text.split()[:5]
```

Out[4]:
```
['one', 'reviewers', 'mentioned', 'watching', '1']
```

In [5]:
```
vocabulary = text.split(" ")
vocabulary = [re.sub('[^a-zA-Z]', '', txt ) for txt in vocabulary]
print(vocabulary[:15])
```

```
['one', 'reviewers', 'mentioned', 'watching', '', 'oz', 'episode', 'hooked', '', 'right', 'exactly', 'happened', '', '', 'first']
```

In [6]:
```
vocabulary.sort()
vocabulary = list(filter(lambda x: x != r"'", vocabulary))
vocabulary = list(filter(None, vocabulary)) #removing nulls
print(vocabulary[:50])
```

```
['aaargh', 'aaliyah', 'aamir', 'aaron', 'aaron', 'ab', 'abandon', 'abandon', 'abandoned', 'abandoned', 'abandoned', 'abandoned', 'abandoned',
'abandoned', 'abandons', 'abba', 'abbey', 'abbey', 'abbeys', 'abbot', 'abbot', 'abbot', 'abbot', 'abbot', 'abbott', 'abbott', 'abbott', 'abbo
tt', 'abbott', 'abbreviated', 'abbreviated', 'abbreviated', 'abductee', 'abedded', 'abetted', 'abetted', 'abetted', 'abhorrent', 'abhorrent',
'abide', 'abiding', 'abiding', 'abilities', 'abilities', 'abilities', 'ability', 'ability', 'ability', 'ability', 'ability']
```

In [7]:
```
df_one_hot_enc = pd.get_dummies(vocabulary)
df_one_hot_enc
```

Out[7]:

	aaargh	aaliyah	aamir	aaron	ab	abandon	abandoned	abandons	abba	abbey	...	zoo	zoology	zoom	zooming	zooms	zp	zu	zucker	zulu	zwick
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	aaargh	aaliyah	aamir	aaron	ab	abandon	abandoned	abandons	abba	abbey	...	zoo	zoology	zoom	zooming	zooms	zp	zu	zucker	zulu	zwick
1	0	1	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
118703	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	0	0
118704	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
118705	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
118706	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
118707	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1

118708 rows × 19612 columns

2. Apply newline tokenization to the text (use split("\n"). Consider each element in the list as a document.

a. Apply preprocessing.

b. Create BoWs vectors for each of the documents

In [8]:

```
def pre_processing2(txt):
    txt = txt.encode('unicode_escape').decode('utf-8') #many characters(like Ã©, Ã-, Ã³,...) are wrongly read in python so converting them back to original character
    txt = str(txt.lower())
    txt = re.sub(r'http\S+', '', txt) #remove URLs
    txt = txt.replace(r"''", "\"") #replacing 2 consequtive single quotes to double quote
    txt = txt.replace(r"<br />", " ")
    txt = txt.replace("\\n", "\n")
    #txt = re.sub('[^a-zA-Z-]', ' ', txt ) #removing punctuations numbers
    txt = txt.translate(str.maketrans('','','\t'))
    txt_list = txt.split(' ')
```

```
    txt_list = [word for word in txt_list if not word in set(stopwords.words('english')) ]
    txt = ' '.join(txt_list)
    return txt
```

In [9]:

```
text = ""
with open (data_path, "r") as f: #
    text = f.read()
text = pre_processing2(text)
doc_list = text.split("\n")
doc_list = [re.sub('[^a-zA-Z-]', ' ', txt ) for txt in doc_list]
doc_list = list(filter(None, doc_list)) #removing null documents
doc_list = [doc.strip() for doc in doc_list] #removing trailing and leading spaces in each document
print(doc_list[0])
```

```
one reviewers mentioned watching  oz episode hooked  right  exactly happened me  first thing struck oz brutality unflinching scenes violenc
e  set right word go  trust me  show faint hearted timid  show pulls punches regards drugs  sex violence  hardcore  classic use word  called
oz nickname given oswald maximum security state penitentary  focuses mainly emerald city  experimental section prison cells glass fronts face
inwards  privacy high agenda  em city home many  aryans  muslims  gangstas  latinos  christians  italians  irish more  so scuffles  death s
tares  dodgy dealings shady agreements never far away  would say main appeal show due fact goes shows dare  forget pretty pictures painted m
ainstream audiences  forget charm  forget romance  oz mess around  first episode ever saw struck nasty surreal  say ready it  watched more
developed taste oz  got accustomed high levels graphic violence  violence  injustice  crooked guards who ll sold nickel  inmates who ll kill
order get away it  well mannered  middle class inmates turned prison bitches due lack street skills prison experience  watching oz  may becom
e comfortable uncomfortable viewing  thats get touch darker side
```

In [10]:

```
len(doc_list)
```

Out[10]:

```
999
```

In [11]:

```
doc_list[1]
```

Out[11]:

```
'a wonderful little production  filming technique unassuming- old-time-bbc fashion gives comforting  sometimes discomforting  sense realism
entire piece  actors extremely well chosen- michael sheen  has got polari  voices pat too  truly see seamless editing guided references wil
liams  diary entries  well worth watching terrificly written performed piece  masterful production one great master s comedy life  realism
really comes home little things  fantasy guard which  rather use traditional  dream  techniques remains solid disappears  plays knowledge sen
ses  particularly scenes concerning orton halliwell sets  particularly flat halliwell s murals decorating every surface  terribly well done'
```

In [12]:

```
def bag_of_words(doc):
    dict_BoW = {}
```

```
    words = nltk.word_tokenize(doc)
    for word in words:
        if( word not in dict_BoW.keys() ):
            dict_BoW[word] = 1
        else:
            dict_BoW[word] += 1

    return dict_BoW
```

In [13]:
```
df_BoW_list = []
for doc in doc_list:
    dict_BoW = bag_of_words(doc)
    df_BoW_list.append(pd.DataFrame([dict_BoW]))
```

In [14]:
```
df_BoW_list[1]
```

Out[14]:

	a	wonderful	little	production	filming	technique	unassuming-	old-time-bbc	fashion	gives	...	orton	halliwell	sets	flat	murals	decorating	every	surface	terrib
0	1	1	2	2	1	1	1	1	1	1	...	1	2	1	1	1	1	1	1	

1 rows × 81 columns

3. Read a search text from the user

a. Using cosine similarity : List the top five similar documents based on the search text

In [15]:
```
def cosine_similarity(A,B):
    res = np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))
    return res
```

In [16]:
```
def eucl_similarity(v1,v2):
```

```
    res = np.sqrt(np.sum((v1 - v2) ** 2))
    res = 1/(1+res)
    return res
```

Taking 15 random adjectives from input text as a user input

In [17]:
```
def extract_adjectives(doc):
    list_token = []
    col_list_token = ["Text", "POS", "Explanation", "Tag"]
    for token in doc:
        list_token.append([token.text, token.pos_, spacy.explain(token.pos_), token.tag_])
    df_spacy_pos = pd.DataFrame(list_token, columns = col_list_token)
    adj_list = list(df_spacy_pos[df_spacy_pos.Tag == "JJ"]['Text'])
    return adj_list
```

In [18]:
```
nlp = spacy.load("en_core_web_sm")
text_rnd = " ".join(list(np.random.choice(text.split(" "), 5000))) #taking 5000 random words from reviews
doc = nlp(text_rnd)
adj_list = extract_adjectives(doc)
adj_list = list(set(adj_list))
len(adj_list)
```

Out[18]:
```
464
```

In [19]:
```
np.random.seed(500)
user_input = np.random.choice(adj_list, 20)
user_input = " ".join(list(user_input))
user_input
```

Out[19]:
```
'young thin faithful exotic such aussie psychological ex horrible new exploitive exotic 19th cruel bored phenomenal actual set evil incredibl
e'
```

In [20]:
```
def text_to_vector(txt, vocabulary):
```

```
    vec = np.zeros(len(vocabulary), dtype = np.int16)
    for w in word_tokenize(txt):
        if w.lower() in vocabulary:
            index = vocabulary.index(w.lower())
            vec[index] += 1
    return vec
```

In [21]:

```
list_cos_sim = []
ind = 0
for doc in doc_list:
    vocabulary = word_tokenize(doc)
    A = text_to_vector(doc, vocabulary)
    B = text_to_vector(user_input, vocabulary)
    cos_sim = cosine_similarity(A,B)
    list_cos_sim.append([ind, cos_sim])
    ind = ind + 1
```

```
<ipython-input-15-46bc0e111988>:2: RuntimeWarning: invalid value encountered in true_divide
  res = np.dot(A,B)/(np.linalg.norm(A)*np.linalg.norm(B))
```

In [22]:

```
list_cos_sim[:20]
```

Out[22]:

```
[[0, 0.06097107608496923],
 [1, nan],
 [2, 0.09853292781642932],
 [3, nan],
 [4, 0.06772854614785964],
 [5, nan],
 [6, 0.08333333333333333],
 [7, nan],
 [8, nan],
 [9, 0.20851441405707477],
 [10, 0.13018891098082389],
 [11, 0.10206207261596577],
 [12, 0.06741998624632421],
 [13, nan],
 [14, nan],
 [15, nan],
 [16, nan],
 [17, 0.07332355751067665],
```

```
[18, nan],
[19, nan]]
```

Cosine Similarity

In [23]:

```
df_cos_sim = pd.DataFrame(list_cos_sim, columns=['doc_ind','cos_sim'])
df_cos_sim.sort_values(by=['cos_sim'], inplace=True, ascending=False)
print("Searched Text = ", user_input)
print("\nTop five similar documents based on the searched text:")
df_cos_sim = df_cos_sim[:5]
for i in range(len(df_cos_sim)):
    print("\n\n\nDoc =\t", doc_list[df_cos_sim.iloc[i, 0]], "\n\nSimilarity =", df_cos_sim.iloc[i, 1])
```

```
Searched Text =  young thin faithful exotic such aussie psychological ex horrible new exploitive exotic 19th cruel bored phenomenal actual se
t evil incredible

Top five similar documents based on the searched text:



Doc =    the golden door story sicilian family s journey old world  italy  new world  america   salvatore  middle-aged man hopes fruitful lif
e  persuades family leave homeland behind sicily  take arduous journey across raging seas  inhabit land whose rivers supposedly flow milk  sh
ort  believe risking everything new world dreams prosperity fulfilled  imagery new world optimistic  clever highly imaginative  silver coins
rain heaven upon salvatore anticipates prosperous he ll new world  carrots onions twice size human beings shown harvested suggest wealth heal
th  rivers milk swam flow minds anticipate new world yield  imagery surrealistically interwoven characters helps nicely compliment gritty rea
lism story unfolds audience  contrast imagery versus dark reality sicilian people helps provide hope they re aboard ship new world  voyage n
ew world shot almost complete darkness  especially seas tempests roar nearly kill people within  dark reality referred old world journey new
world  old world depicted somewhat destitute primitive  shown salvatore scrambles together sell possessions left  donkeys  goats rabbits  ord
er obtain appropriate clothing needs enter new world  thought rather interesting people believed conform certain dress code order accepted ne
w world  almost suggesting people fit particular stereotype mold order recognized morally fit  powerful image film ship leaving homeland sett
ing sail new world  shot shows overhead view crowd people slowly seem separate one another  depicting separation old new worlds  shot also su
ggested people torn away familiar  wanted divorce previous dark living conditions desirous enter world held promise  later contrasted new wo
rld visually looks  old world seems dark bleak compared bright yet foggy new world  thought particularly interesting statue liberty never sho
wn fog ellis island  remained hidden  think intentional directing choice seemed negate purpose statue liberty stands for  give poor  tired
hungry  seemed like joke regards people go arriving new world  arrived americas  go rather humiliating tests  i e  delousing  mathematics  pu
zzles  etc  order prove fit new world  tests completely changed perspectives sicilian people  particular  salvatore s mother difficult time
subjecting rules laws new world  feeling violated treated respect  dreams provided hope optimism new world would provide  reality new world r
equired disparaging rude  salvatore change much attitude towards felt new world would like versus new world actually seemed disappointing him
attitude shared mostly everyone voyaged him  character arcs deal cherished dream greatly upset dark reality accepted  film seems make strong
commentary preparing oneself enter heavenly civilized society  cleanliness  marriage intelligence prerequisites  adhering rules prevent disea
se  immoral behavior stupidity dominating  perhaps commentary america learned failings nations purposefully established secure plagues infest
destruct  though rules seemed rigid  protect help people flourish

Similarity = 0.5076849508812099
```

```
Doc =    one starewicz s longest strangest short films follows toy dog search orange becoming animated tear mother girl longs orange  dog com
es upon orange falling back car way sold  night must protect orange comes enters devilish nightclub featuring many bizarre scary characters
help stuffed cat  dog gets orange back little girl saved terrible scurvy death  mascot features new techniques yet seen starewicz s films  ad
dition sync sound mixture live action stop-motion animation makes new twist starewicz s old style puppetry  live scenes moving cars people s
feet walking puppet sits concrete sidewalk impressive fresh  honking cars cries street vendors noteworthy due fact small studio shifts sound
costly starewicz s utilization new technology seems like old hat  new puppet characters film frightening contributions devil s club scene  tw
igss newspaper shreds come life  skeletons dead birds lay eggs hatch skeleton chicks  characters come flying pats pans rocking horses  new ed
iting technique uses quick zooms accomplished editing speed pace might slow scene  overall  starewicz able update style film-making meet dema
nds new audience making film one best examples work

Similarity = 0.3375263702778072



Doc =    lost          carnivale    desperate housewifes    the list goes on  these  bunch high-quality  shows proves we re middle golden age
television history   lost  pure genius  incredible layers personal  psychologically viable  stories  underscored sublime cinematography  incr
edible use word  describing tv-show   killer score  great performances editing  anyone hooked this  missing one important creative expression
s television ever  may problems  watching one episode week  dvd format actually incredible way watch this  hope keep  as i m sure do

Similarity = 0.3234983196103152



Doc =    this funny film like lot  cary elwes plays robin hood tee  is  course  usual good vs evil robin evil sheriff nottingham  humor sort
face stuff part  still works well  comedy night want think much  well worth rent

Similarity = 0.30499714066520933



Doc =    the one remarkable sci-fi movies millennium  movie incredible future vision  movie establishes new standard s f movies  hail kill

Similarity = 0.29488391230979427
```

Euclidean Similarity

In [25]:

```
list_euc_sim = []
ind = 0
for doc in doc_list:
    vocabulary = word_tokenize(doc)
    A = text_to_vector(doc, vocabulary)
    B = text_to_vector(user_input, vocabulary)
    euc_sim = eucl_similarity(A,B)
    list_euc_sim.append([ind, euc_sim])
```

```
    ind = ind + 1
df_euc_sim = pd.DataFrame(list_euc_sim, columns=['doc_ind','euc_sim'])
df_euc_sim.sort_values(by=['euc_sim'], inplace=True, ascending=False)
print("Searched Text = ", user_input)
print("\nTop five similar documents based on the searched text:")
df_euc_sim = df_euc_sim[:5]
for i in range(len(df_euc_sim)):
    print("\n\nDoc =\t", doc_list[df_euc_sim.iloc[i, 0]], "\n\nSimilarity =", df_euc_sim.iloc[i, 1])
```

```
Searched Text =  young thin faithful exotic such aussie psychological ex horrible new exploitive exotic 19th cruel bored phenomenal actual se
t evil incredible

Top five similar documents based on the searched text:


Doc =    a rating     begin express dull  depressing relentlessly bad movie is

Similarity = 0.2402530733520421


Doc =    what waste talent  poor  semi-coherent  script cripples film  rather unimaginative direction  too  faint echoes  fargo  here  come o
ff

Similarity = 0.1907435698305462


Doc =    this great film  touching strong  direction without question breathless  good work team  feel sorry marlene  grace god go i

Similarity = 0.18660549686337075


Doc =    highly regarded release  since rather neglected  immense importance history performing arts  classic use embedded plots  one favouri
te films  soundtrack re-released

Similarity = 0.1827439976315568


Doc =    the one remarkable sci-fi movies millennium  movie incredible future vision  movie establishes new standard s f movies  hail kill

Similarity = 0.179128784747792
```