Niboon Boonprakob 61340500038

2 Sentiment Analysis

2.2 Movie Review Data

Let us first start by looking at the data provided with the exercise. We have positive and negative movie reviews labeled by human readers, all positive and negative reviews are in the 'pos' and 'neg' folders respectively. If you look in- side a sample file, you will see that these review messages have been 'tokenized', where all words are separated from punctuations. There are approximately 1000 files in each category with files names starting with cv000, cv001, cv002 and so on. You will split the dataset into training set and testing set.

1. Write some code to load the data from text files.

In [4]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [1]:

```
from os import listdir
from os.path import join
from collections import Counter
```

In [2]:

```
# function load file text

def load_data (filename):
    file = open(filename , "r")
    text = file.read() # read all
    file.close()
    return text
```

In [4]:

```
# load_data("review_polarity/txt_sentoken/neg/cv000_29416.txt")
```

In [5]:

```
text_all = []

directory = "review_polarity/txt_sentoken/neg"
for filename in listdir(directory):
    if not filename.endswith(".txt"):
        continue
    path = directory + "/" + filename
    file = open(path)
    text = file.read()
    file.close()
#     print(text)
    text_all.append(text)

directory = "review_polarity/txt_sentoken/pos"
for filename in listdir(directory):
    if not filename.endswith(".txt"):
        continue
    path = directory + "/" + filename
    file = open(path)
    text = file.read()
    file.close()
     #print(text)
    text_all.append(text)
```

In [6]:

```
# text_all
```

In [7]:

```
# !pip install -U nltk
```

```
Requirement already satisfied: nltk in /home/niboon_b/anaconda3/lib/python3.8
/site-packages (3.6.1)
Collecting nltk
  Downloading nltk-3.6.2-py3-none-any.whl (1.5 MB)
     |████████████████████████████████| 1.5 MB 1.5 MB/s eta 0:00:01
Requirement already satisfied: joblib in /home/niboon_b/anaconda3/lib/python
3.8/site-packages (from nltk) (1.0.1)
Requirement already satisfied: click in /home/niboon_b/anaconda3/lib/python3.
8/site-packages (from nltk) (7.1.2)
Requirement already satisfied: regex in /home/niboon_b/anaconda3/lib/python3.
8/site-packages (from nltk) (2021.4.4)
Requirement already satisfied: tqdm in /home/niboon_b/anaconda3/lib/python3.8
/site-packages (from nltk) (4.59.0)
Installing collected packages: nltk
  Attempting uninstall: nltk
    Found existing installation: nltk 3.6.1
    Uninstalling nltk-3.6.1:
      Successfully uninstalled nltk-3.6.1
Successfully installed nltk-3.6.2
```

In [8]:

```
import nltk
# nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.
xml
```

Out[8]: True

In [9]:

```
# Clean text data
from nltk.corpus import stopwords
import string
def clean_text(data):
    dataframe = data.split()
    table = str.maketrans("","",string.punctuation)
    dataframe = [w.translate(table) for w in dataframe]
    dataframe = [word for word in dataframe if word.isalpha()]

    stop_words = set(stopwords.words('english'))
    dataframe = [w for w in dataframe if not w in stop_words]

    dataframe = [word for word in dataframe if len(word) >1 ]
    return dataframe
```

In [10]:

```
def all_text (directory , vocab):
    load_line = []
    for filename in listdir(directory):
        if not filename.endswith(".txt"):
            continue
        path = directory + "/" + filename
        #add_text_to_vocab(path , vocab)
        load = add_text_to_vocab(path , vocab)
        load_line.append(load)
    return load_line
```

In [11]:

```
def add_text_to_vocab(filename , vocab):
    text = load_data(filename) # load test
    dataframe = clean_text(text)
    vocab.update(dataframe)
    # filter vocab
    dataframe = [w for w in dataframe if w in vocab]
    return ''.join(dataframe)
```

In [12]:

```
# define vocab (pos , neg)
vocab = Counter()
vocab_pos = all_text("review_polarity/txt_sentoken/pos",vocab)
vocab_neg = all_text("review_polarity/txt_sentoken/neg",vocab)
```

In [13]:

```
len(vocab)
```

Out[13]: 46557

In [14]:
```
vocab.most_common(20)
```

Out[14]:
```
[('film', 8860),
 ('one', 5521),
 ('movie', 5440),
 ('like', 3553),
 ('even', 2555),
 ('good', 2320),
 ('time', 2283),
 ('story', 2118),
 ('films', 2102),
 ('would', 2042),
 ('much', 2024),
 ('also', 1965),
 ('characters', 1947),
 ('get', 1921),
 ('character', 1906),
 ('two', 1825),
 ('first', 1768),
 ('see', 1730),
 ('well', 1694),
 ('way', 1668)]
```

In [15]:
```
# Save Prepared data
def save_text (path , filename):
    data = '\n'.join(path)
    file = open(filename,"w")
    file.write(data)
    file.close()
```

In [16]:
```
# หาอันที่มีโอกาสเกิดน้อยที่สุด
min_occurrence = 5
tokens = [i for i,j in vocab.items() if j >= min_occurrence]
```

In [18]:
```
# print(tokens , len(tokens))
```

In [19]:
```
save_text(tokens,"data_vocab.txt")
```

In [20]:
```
# load vocab
df_vocab = "data_vocab.txt"
df_vocab = load_data(df_vocab)
df_vocab = df_vocab.split()
df_vocab = set(df_vocab)
neg_data = all_text("review_polarity/txt_sentoken/neg/" , vocab)
pos_data = all_text("review_polarity/txt_sentoken/pos/" , vocab)

# save pos_data and neg_data
save_text(neg_data , "data_neg.txt")
save_text(pos_data , "data_pos.txt")
```

2.3 TF-IDF

From a raw text review, you want to create a vector, whose elements indicate the number of each word in each document. The frequency of all words within the documents are the ‘features’ of this machine learning problem.

A popular method for transforming a text to a vector is called tf-idf, short for term frequencyinverse document frequency.

1. Conduct a research about tf-idf and explain how it works.
2. Scikit-learn provides a module for calculating this, this is called TfidfVec- torizer. You can study how this function is used here:

```
 http://scikit-learn.org/stable/modules/generated
/sklearn.feature_extraction.text.TfidfVectorizer.html
```

Write code to transform your text to tf-idf vector.

Conduct a research about tf-idf and explain how it works.

หลังการทำงานของ Tf-idf คือ ใช้เปรียบเทียบความเหมือนกันของคำสอนคำ โดยวัดจาก tf และ idf การวัด cosine ตรงๆระหว่างคำสองคำไม่ค่อยเหมาะ เนื่องจากข้อมูลจะเบ้หนักมาก แล้วก็แยกแยะกันได้ยาก เช่น คำพวก the , it , they พวกนี้จะมาทำการแยกยากเพราะไม่ค่อยให้ข้อมูลอะไร เราเลยจะใช้ Tf-idf เข้ามาช่วย

Tf : Term frequency คือการนับความถี่ของคำ (t) ในเอกสาร (d) จำนวนคำ tf = count(t,d) หรือเราสามา take log ฐาน 10 เข้าไปเพื่อปรับสเกลไม่ให้เวอร์เกินไป tf = log10(count(t,d)+1) (+1 เพราะว่าไม่ให้เกิดเคส log0) idf : Inverse Document Frequency df : document Frequency คือจะให้ weight เยอะๆ สำหรับคำที่เจอไม่บ่อยในเอาสารมันมาจากแนวคิดว่าถ้าคำไหนเจอบ่อยๆใน เอกสารจะแปลว่ามันไม่สำคัญ df จะแตกต่างจาก collection frquency ตรงที่ถ้ากรอบ collection ของเราคือ document หลายๆชิ้นเวลานับ document frquency เราจะนับคำนั้นๆไปโผล่ใน document กี่ชิ้น ส่วน collection fequency จะนับว่าคำนั้นไปโผล่ใน collection (document ไหนก็ได้) กี่ครั้ง

In [ ]:

```
from IPython import display
display.Image("pic_tf-idf/0_nFyAU7l38Wo1dHhK.png")
```

Out[ ]:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

เราจะใช้ Inverse Document Frequency ดทนเพื่อให้ตีความง่านกว่า Df มีสูตร คือ N/df โดย N คือ จำนวน document ใน collection ส่วน df คือจำนวน document ที่มีคำนั้นๆ นอกจากนี้ก็มีการ take log ฐาน10 เขาไปเช่นกัน เพื่อปรับสเกล idf = log10(N/df)

In [ ]:
```
display.Image("pic_tf-idf/0_BM0SUOJBB4niQs0f.png")
```

Out[ ]:

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

ตามรูป N = 37 ทำให้ good กับ sweet ที่มี df = 37 คือ idf ออกมาได้ 0 ยิ่งค่า idf สูงๆแปลว่าคำๆนั้นสำคัญมาก สรุป tf-idf สามารถคำนวณตรงๆ ได้เลยคือ w = tf * idf

In [ ]:
```
display.Image("pic_tf-idf/0_xpFqparOAIPNTZw9.png")
```

Out[ ]:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Figure 6.8 A tf-idf weighted term-document matrix for four words in four Shakespeare plays, using the counts in Fig. 6.2. For example the 0.049 value for *wit* in *As You Like It* is the product of tf $= \log_{10}(20+1) = 1.322$ and idf $= .037$. Note that the idf weighting has eliminated the importance of the ubiquitous word *good* and vastly reduced the impact of the almost-ubiquitous word *fool*.

เราจะใช้ tf-idf เป็นค่ามาตรฐาน (baseline) สำหรับพิจรณา weighting ของ cooccurrence matrix

Refer: https://medium.com/@sirasith.petch/word-embedding-tf-idf-%E0%B9%81%E0%B8%A5%E0%B8%B0-word2vec-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3-%E0%B9%81%E0%B8%A5%E0%B9%89%E0%B8%A7%E0%B8%A1%E0%B8%B1%E0%B8%99%E0%B8%A1%E0%B8%B5%E0%B8%9B%E0%B8%A3%E0%B8%B0%E0%B9%82%E0%B8%A2%E0%B8%8A%E0%B8%99%E0%B9%8C%E0%B8%A2%E0%B8%B1%E0%B8%87%E0%B9%84%E0%B8%87-9a6c593cf507

Scikit-learn provides a module for calculating this, this is called TfidfVec- torizer. You can study how this function is used here

In [21]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

text_all = []
buff = 0
directory = "review_polarity/txt_sentoken/neg/"
for filename in listdir(directory):
    if not filename.endswith(".txt"):
        continue
    path = directory + "/" + filename
    file = open(path)
    text = file.read()
    file.close()
#     print(text)
    text_all.append(text)

directory = "review_polarity/txt_sentoken/pos/"
for filename in listdir(directory):
    if not filename.endswith(".txt"):
        continue
    path = directory + "/" + filename
    file = open(path)
    text = file.read()
    file.close()
     #print(text)
    text_all.append(text)
```

In [22]:

```
# print(text_all)
```

In [23]:

```
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(text_all)
```

In [24]:

```
X.shape
```

Out[24]:

```
(2000, 39659)
```

In [25]:

```
# Feature name
# vectorizer.get_feature_names()
```

Out[25]:

```
['00',
 '000',
 '0009f',
 '007',
 '00s',
 '03',
 '04',
 '05',
 '05425',
 '10',
 '100',
 '1000',
 '10000',
 '100m',
```

```
 '101',
 '102',
 '103',
 '104',
 '105',
 '106',
 '107',
 '108',
 '109',
 '10b',
 '10s',
 '10th',
 '11',
 '110',
 '111',
 '112',
 '113',
 '1138',
 '114',
 '115',
 '117',
 '118',
 '11th',
 '12',
 '121',
 '122',
 '123',
 '125',
 '126',
 '127',
 '1272',
 '128',
 '129',
 '1298',
 '12th',
 '13',
 '130',
 '1305',
 '131',
 '132',
 '133',
 '135',
 '137',
 '138',
 '139',
 '13th',
 '14',
 '140',
 '1400',
 '143',
 '144',
 '14th',
 '15',
 '150',
 '1500s',
 '150th',
 '151',
 '152',
 '1521',
 '153',
```

```
'155',
'1554',
'157',
'1583',
'1590',
'15th',
'16',
'160',
'1600',
'1600s',
'161',
'165',
'167',
'1692',
'16mm',
'16th',
'16x9',
'17',
'170',
'1700s',
'1709',
'172',
'175',
'1773',
'1791',
'1792',
'1793',
'1794',
'1799',
'17th',
'18',
'180',
'1800',
'1800s',
'1812',
'1830s',
'1839',
'1847',
'1862',
'1865',
'1869',
'1871',
'1885',
'1888',
'189',
'1896',
'1898',
'1899',
'18s',
'18th',
'19',
'1900',
'1900s',
'1903',
'1908',
'1912',
'1913',
'1914',
'1916',
'1919',
```

```
 '1920s',
 '1922',
 '1923',
 '1925',
 '1926',
 '1928',
 '1930',
 '1930s',
 '1932',
 '1933',
 '1934',
 '1935',
 '1937',
 '1938',
 '1939',
 '1940',
 '1940s',
 '1941',
 '1942',
 '1943',
 '1944',
 '1945',
 '1946',
 '1947',
 '1948',
 '1949',
 '1950',
 '1950s',
 '1951',
 '1952',
 '1953',
 '1954',
 '1955',
 '1956',
 '1957',
 '1958',
 '1959',
 '1960',
 '1960s',
 '1961',
 '1962',
 '1963',
 '1964',
 '1965',
 '1966',
 '1967',
 '1968',
 '1969',
 '1970',
 '1970s',
 '1971',
 '1972',
 '1973',
 '1974',
 '1975',
 '1976',
 '1977',
 '1978',
 '1979',
 '1980',
```

```
'1980s',
'1981',
'1982',
'1983',
'1984',
'1985',
'1986',
'1987',
'1988',
'1989',
'1990',
'1990s',
'1991',
'1992',
'1993',
'1994',
'1995',
'1996',
'1997',
'1998',
'1998s',
'1999',
'19th',
'1hr',
'1st',
'20',
'200',
'2000',
'2001',
'2002',
'2007',
'2010',
'2013',
'2015',
'2017',
'2018',
'2020',
'2023',
'2024',
'2029',
'2036',
'2040',
'2050',
'2056',
'2058',
'206',
'2065',
'209',
'2099',
'20s',
'20somethings',
'20th',
'20thcentury',
'21',
'216',
'2176',
'21a',
'21st',
'22',
'2259',
```

```
'2293',
'23',
'230',
'234',
'23rd',
'24',
'2400',
'2470',
'24th',
'25',
'250',
'254',
'25th',
'26',
'2654',
'26min',
'26th',
'27',
'28',
'280',
'289',
'28th',
'28up',
'29',
'2_',
'2am',
'2d',
'2hr',
'2nd',
'2th',
'30',
'300',
'3000',
'30ish',
'30m',
'30s',
'30th',
'31',
'310',
'31st',
'32',
'33',
'34',
'3411',
'3465',
'34th',
'35',
'357',
'35mm',
'35th',
'36',
'360',
'3654',
'36th',
'37',
'37th',
'38th',
'39',
'3d',
'3p0',
```

```
'3po',
'3rd',
'40',
'400',
'40mins',
'40s',
'41',
'42',
'426',
'43',
'44',
'449',
'45',
'460',
'47',
'48',
'48th',
'49',
'4960',
'4am',
'4th',
'50',
'500',
'5000',
'50000',
'500k',
'500th',
'50s',
'50th',
'51',
'51st',
'52',
'53',
'54',
'54th',
'55',
'555',
'56',
'5671',
'56k',
'57',
'571',
'57th',
'58',
'59',
'5th',
'60',
'600',
'6000',
'607',
'60s',
'61',
'62',
'63',
'64',
'640',
'65',
'65th',
'66',
'666',
```

```
'67',
'68',
'69',
'6th',
'70',
'700',
'7000',
'701',
'70ies',
'70m',
'70mm',
'70s',
'710',
'712',
'73',
'747s',
'75',
'750',
'76',
'77',
'777',
'779',
'78',
'79',
'7th',
'80',
'800',
'802',
'8034',
'80s',
'81',
'82',
'8216',
'83',
'84',
'85',
'85but',
'86',
'87',
'88',
'89',
'8a',
'8mm',
'8th',
'90',
'900',
'90210',
'90s',
'91',
'911',
'92',
'92ll',
'92re',
'92s',
'92t',
'92ve',
'93',
'939',
'94',
'95',
```

```
'95s',
'96',
'97',
'98',
'99',
'999',
'9mm',
'_2001_',
'_21_jump_street_',
'_48_hrs',
'_54_',
'___',
'____',
'_____',
'______',
'____________________________________________',
'__________________________________________________________',
'_a',
'_a_night_at_the_roxbury_',
'_air_force_one_',
'_all_',
'_am_',
'_amadeus_',
'_america',
'_american_beauty_',
'_american_psycho_',
'_and_',
'_andre_',
'_angel',
'_animal',
'_anything_',
'_are_',
'_armageddon_',
'_arrrgh_',
'_babe_',
'_bad_',
'_basquiat_',
'_before_',
'_beloved_',
'_blade',
'_blade_',
'_boogie',
'_boom_',
'_brazil_',
'_breakfast_',
'_breakfast_of_champions_',
'_but',
'_can',
'_can_',
'_casablanca_',
'_cliffhanger',
'_cliffhanger_',
'_clueless_',
'_come_',
'_dancing_',
'_dawson',
'_daylight_',
'_dead_',
'_dead_man_',
'_dead_man_on_campus_',
```

```
'_death',
'_dirty_work_',
'_disturbing_behavior_',
'_do_',
'_doctor_dolittle_',
'_does_',
'_dog',
'_don',
'_double_team_',
'_dragon',
'_dragon_',
'_dragonheart_',
'_election',
'_election_',
'_entertainment_weekly_',
'_escape',
'_eve',
'_everybody_',
'_exactly_',
'_experience_',
'_fantastic_',
'_fear_and_loathing_in_las_vegas_',
'_ferris',
'_fifty_',
'_film',
'_fisherman',
'_flirting',
'_four_',
'_full_house_',
'_gag',
'_gattaca_',
'_genius_',
'_ghost',
'_great_',
'_h20_',
'_halloween',
'_halloween_',
'_happen_',
'_hard_ware',
'_have_',
'_heathers_',
'_here_',
'_highly_',
'_his_',
'_holy_man_',
'_home',
'_home_alone_',
'_hope',
'_huge_',
'_hustler_',
'_i_know',
'_i_know_what_you_did_last_summer_',
'_in',
'_into_',
'_is_',
'_it',
'_itcom_',
'_jerry_maguire_',
'_john',
'_juliet_',
```

```
'_jumanji_',
'_kingpin_',
'_knock_off_',
'_la',
'_last',
'_last_',
'_least_',
'_leave',
'_life',
'_little',
'_loathe_',
'_lone',
'_long_',
'_looks_',
'_lot_',
'_mafia_',
'_many_',
'_matewan_',
'_matrix_',
'_melvin',
'_mind',
'_moby',
'_monster_movie_',
'_more_',
'_mortal',
'_murder_',
'_must_',
'_never',
'_no',
'_not_',
'_october',
'_offscreen_',
'_onegin_',
'_original_',
'_patlabor',
'_patlabor_',
'_pecker_',
'_people_',
'_pick_chucky_up_',
'_polish_wedding_',
'_pollock_',
'_poltergeist_',
'_porky',
'_practical',
'_quite_',
'_reach_the_rock_',
'_real_',
'_reality',
'_really_',
'_red',
'_remains',
'_require_',
'_roxbury_',
'_rushmore_',
'_saturday_night_live_',
'_saved_by_the_bell_',
'_saving',
'_scarface_',
'_schindler',
'_scream',
```

```
'_scream_',
'_scream_2_',
'_seven_nights_',
'_shaft',
'_shaft_',
'_shaft_in_africa_',
'_shine_',
'_should_',
'_six_days',
'_snl_',
'_so',
'_soldier_',
'_some_',
'_somewhere_',
'_star',
'_still_',
'_survives_',
'_that_',
'_the',
'_the_',
'_the_broadway_musical_',
'_the_fugitive_',
'_the_last_days_of_disco_',
'_the_lion_king',
'_the_quest_',
'_their_',
'_there_',
'_they',
'_this_',
'_titanic_',
'_titus_',
'_titus_andronicus_',
'_to',
'_today_',
'_too_',
'_twice_',
'_two_',
'_unbreakable_',
'_urban',
'_urban_legend_',
'_vampires_',
'_very_',
'_very_small_',
'_wag',
'_waiting_to_exhale_',
'_wartime',
'_wayyyyy_',
'_what',
'_whole_',
'_will_',
'_william_shakespeare',
'_wishmaster_',
'_would_',
'aa',
'aaa',
'aaaaaaaaah',
'aaaaaaaahhhh',
'aaaaaah',
'aaaahhhs',
'aahs',
```

```
'aaliyah',
'aalyah',
'aamir',
'aardman',
'aaron',
'aatish',
'ab',
'aback',
'abandon',
'abandoned',
'abandoning',
'abandonment',
'abandons',
'abating',
'abba',
'abbe',
'abberation',
'abberline',
'abbots',
'abbott',
'abbotts',
'abbreviated',
'abby',
'abc',
'abdomen',
'abducted',
'abductees',
'abduction',
'abductions',
'abdul',
'abe',
'abel',
'aberdeen',
'aberration',
'abetted',
'abetting',
'abeyance',
'abhorrence',
'abhorrent',
'abider',
'abides',
'abiding',
'abigail',
'abiility',
'abilities',
'ability',
'abject',
'ablaze',
'able',
'ably',
'abnormal',
'abnormally',
'abo',
'aboard',
'abode',
'abolish',
'abolitionist',
'abolitionists',
'abominable',
'abomination',
```

```
'aborbed',
'aborginal',
'aboriginal',
'aboriginals',
'aborigine',
'abort',
'aborted',
'abortion',
'abortionist',
'abortions',
'abortive',
'aboslutely',
'abound',
'abounded',
'abounding',
'abounds',
'about',
'abouts',
'above',
'abraded',
'abraham',
'abrahams',
'abrams',
'abrasive',
'abreast',
'abril',
'abroad',
'abrupt',
'abruptly',
'abs',
'absconded',
'absence',
'absences',
'absense',
'absent',
'absentee',
'absinthe',
'absoloute',
'absoltuely',
'absolut',
'absolute',
'absolutely',
'absolutes',
'absolution',
'absolutist',
'absolved',
'absorb',
'absorbant',
'absorbed',
'absorbing',
'absorbs',
'absorption',
'abstinence',
'abstract',
'abstraction',
'absurd',
'absurdism',
'absurdist',
'absurdities',
'absurdity',
```

```
 'absurdly',
 'abu',
 'abundance',
 'abundant',
 'abundantly',
 'abundence',
 'abuse',
 'abused',
 'abuser',
 'abusers',
 'abuses',
 'abusing',
 'abusive',
 'abuzz',
 'abysmal',
 'abysmally',
 'abyss',
 'abyssinian',
 'ac',
 'ac3',
 'academe',
 'academia',
 'academic',
 'academics',
 'academy',
 'accelerate',
 'accelerated',
 'accelerates',
 'accelerating',
 'acceleration',
 'accelerator',
 'accent',
 'accented',
 'accents',
 'accentuate',
 'accentuated',
 'accentuates',
 'accentuating',
 'accept',
 'acceptable',
 'acceptance',
 'accepted',
 'accepting',
 'acception',
 'accepts',
 'access',
 'accessibility',
 'accessible',
 'accessorize',
 'accessory',
 'accident',
 'accidental',
 'accidentally',
 'accidentlly',
 'accidently',
 'accidents',
 'acclaim',
 'acclaimed',
 'acclimatize',
 'accolade',
```

```
'accolades',
'accommodate',
'accommodates',
'accommodating',
'accommodations',
'accomodates',
'accompanied',
'accompanies',
'accompaniment',
'accompany',
'accompanying',
'accomplice',
'accomplices',
'accomplish',
'accomplished',
'accomplishes',
'accomplishing',
'accomplishment',
'accomplishments',
'accord',
'accordance',
'according',
'accordingly',
'accordion',
'accost',
'accosted',
'accosts',
'account',
'accountability',
'accountable',
'accountant',
'accountants',
'accounted',
'accounts',
'accumulate',
'accumulated',
'accumulation',
'accuracy',
'accurate',
'accurately',
'accursed',
'accusation',
'accusations',
'accuse',
'accused',
'accuser',
'accuses',
'accusing',
'accustomed',
'ace',
'acerbic',
'acerbity',
'aces',
'ache',
'acheivement',
'acheives',
'aches',
'achievable',
'achieve',
'achieved',
```

```
'achieveing',
'achievement',
'achievements',
'achiever',
'achieves',
'achieving',
'achilles',
'achin',
'achingly',
'achoo',
'acid',
'acidic',
'aciton',
'ack',
'acking',
'ackland',
'acknowledge',
'acknowledgeable',
'acknowledged',
'acknowledgement',
'acknowledges',
'acknowledging',
'acknowledgment',
'acme',
'acne',
'acore',
'acquaintance',
'acquaintances',
'acquainted',
'acquaints',
'acquiescence',
'acquire',
'acquired',
'acquires',
'acquisition',
'acquit',
'acquits',
'acquittal',
'acquitted',
'acre',
'acres',
'acrimonious',
'acrimony',
'acrobat',
'acrobatic',
'acrobatics',
'acrobats',
'acronym',
'across',
'acrylic',
'act',
'acted',
'acting',
'action',
'actioner',
'actioners',
'actionfest',
'actionless',
'actions',
'activated',
```

```
 'active',
 'actively',
 'activist',
 'activists',
 'activites',
 'activities',
 'activity',
 'actor',
 'actors',
 'actosta',
 'actress',
 'actresses',
 'acts',
 'actual',
 'actualisation',
 'actuality',
 'actualization',
 'actualizing',
 'actually',
 'actualy',
 'acuity',
 'acumen',
 'acupuncture',
 'acute',
 'acutely',
 'ad',
```

2.4 Classification

Use 4 different models to classify each movie into positive or negative category.

1. K-Nearestneighbormodel,using module `sklearn.neighbors.KNeighborsClassifier`
2. RandomForest, using module `sklearn.ensemble.RandomForestClassifier`
3. SVM, using module `sklearn.svm.SVC`
4. Neural network, using `sklearn.neural_network.MLPClassifier`

You may pick other models you would like to try. Just present results for at least 4 models. Please provide your code for model fitting and cross validation. Calculate your classification accuracy, precision, and recall.

ทำการแบ่งข้อมูลใหม่

In [56]:
```
Data_dir = "review_polarity/txt_sentoken/"
```

In [57]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_files
```

In [58]:
```
data = load_files(Data_dir , encoding = "utf-8" , decode_error="replace")
```

In [59]:
```
# data
```

In [60]:
```
# Calculate count of each category
labels , counts = np.unique(data.target , return_counts = True)
```

In [61]:
```
labels_str = np.array(data.target_names)[labels]
labels_str = dict(zip(labels_str , counts))
```

In [62]:
```
labels_str
```

Out[62]:
```
{'neg': 1000, 'pos': 1000}
```

In [63]:
```
data.data[0]
```

Out[63]:
```
"arnold schwarzenegger has been an icon for action enthusiasts , since the la
te 80's , but lately his films have been very sloppy and the one-liners are g
etting worse . \nit's hard seeing arnold as mr . freeze in batman and robin ,
especially when he says tons of ice jokes , but hey he got 15 million , what'
s it matter to him ? \nonce again arnold has signed to do another expensive b
lockbuster , that can't compare with the likes of the terminator series , tru
e lies and even eraser . \nin this so called dark thriller , the devil ( gabr
iel byrne ) has come upon earth , to impregnate a woman ( robin tunney ) whic
h happens every 1000 years , and basically destroy the world , but apparently
god has chosen one man , and that one man is jericho cane ( arnold himself )
. \nwith the help of a trusty sidekick ( kevin pollack ) , they will stop at
nothing to let the devil take over the world ! \nparts of this are actually s
o absurd , that they would fit right in with dogma . \nyes , the film is that
weak , but it's better than the other blockbuster right now ( sleepy hollow )
, but it makes the world is not enough look like a 4 star film . \nanyway , t
his definitely doesn't seem like an arnold movie . \nit just wasn't the type
of film you can see him doing . \nsure he gave us a few chuckles with his wel
l known one-liners , but he seemed confused as to where his character and the
film was going . \nit's understandable , especially when the ending had to be
changed according to some sources . \naside form that , he still walked throu
gh it , much like he has in the past few films . \ni'm sorry to say this arno
ld but maybe these are the end of your action days . \nspeaking of action , w
here was it in this film ? \nthere was hardly any explosions or fights . \nth
e devil made a few places explode , but arnold wasn't kicking some devil butt
. \nthe ending was changed to make it more spiritual , which undoubtedly ruin
ed the film . \ni was at least hoping for a cool ending if nothing else occur
red , but once again i was let down . \ni also don't know why the film took s
o long and cost so much . \nthere was really no super affects at all , unless
you consider an invisible devil , who was in it for 5 minutes tops , worth th
e overpriced budget . \nthe budget should have gone into a better script , wh
ere at least audiences could be somewhat entertained instead of facing boredo
m . \nit's pitiful to see how scripts like these get bought and made into a m
ovie . \ndo they even read these things anymore ? \nit sure doesn't seem like
it . \nthankfully gabriel's performance gave some light to this poor film . \
nwhen he walks down the street searching for robin tunney , you can't help bu
```

```
t feel that he looked like a devil . \nthe guy is creepy looking anyway ! \nw
hen it's all over , you're just glad it's the end of the movie . \ndon't both
er to see this , if you're expecting a solid action flick , because it's neit
her solid nor does it have action . \nit's just another movie that we are suc
kered in to seeing , due to a strategic marketing campaign . \nsave your mone
y and see the world is not enough for an entertaining experience . \n"
```

In [64]:
```
data.target[0]
```

Out[64]: 0

In [65]:
```
from sklearn.feature_extraction.text import TfidfVectorizer ,TfidfTransformer
vectorizer = TfidfVectorizer(stop_words="english"  , decode_error= "ignore").
tf_idf = TfidfTransformer()
vector = tf_idf.fit_transform(vectorizer.fit_transform(data.data)) # x
print(vector.shape)
```

```
(2000, 39354)
```

In [66]:
```
# Data preparation
from sklearn.model_selection import train_test_split
X_train , x_test , y_train , y_test = train_test_split(vector , data.target ,
```

In [67]:
```
X_train.shape
```

Out[67]: (1600, 39354)

In [68]:
```
y_train.shape
```

Out[68]: (1600,)

In [69]:
```
x_test.shape
```

Out[69]: (400, 39354)

Build model

1.K-Nearestneighbor

2.RandomForest

3.SVM

4.Neural network

Just present results for at least 4 models. Please provide your code for model fitting and cross validation.

Calculate your classification accuracy, precision, and recall

K-Nearestneighbor

In [41]:

```
from sklearn.model_selection import cross_val_predict , cross_val_score
from sklearn.metrics import  accuracy_score , precision_score , recall_score
from sklearn.neighbors import KNeighborsClassifier
np.random.seed(42)

neigh = KNeighborsClassifier(n_neighbors=5)
model = neigh.fit(X_train,y_train)

y_pred = model.predict(x_test)

# cross validation
print(f"model (KNN) accuracy  : {accuracy_score(y_test,y_pred):.2f}")
print(f"model (KNN) precision : {precision_score(y_test, y_pred):.2f}")
print(f"model (KNN) recall    : {recall_score(y_test, y_pred):.2f}")
```

```
model (KNN) accuracy  : 0.66
model (KNN) precision : 0.67
model (KNN) recall    : 0.68
```

RandomForest

In [42]:

```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)

clf = RandomForestClassifier(n_estimators=100, criterion="gini")
model = clf.fit(X_train ,y_train)

y_pred = model.predict(x_test)
print(f"model (RandomForest) accuracy  : {accuracy_score(y_test,y_pred):.2f}"
print(f"model (RandomForest) precision : {precision_score(y_test, y_pred):.2f
print(f"model (RandomForest) recall    : {recall_score(y_test, y_pred):.2f}")
```

```
model (RandomForest) accuracy  : 0.80
model (RandomForest) precision : 0.85
model (RandomForest) recall    : 0.76
```

SVM

In [43]:

```
from sklearn.svm import SVC
np.random.seed(42)
svm = SVC(C=1 , kernel="sigmoid" , degree = 3)
model = svm.fit(X_train , y_train)
y_pred = model.predict(x_test)

print(f"model (SVM) accuracy  : {accuracy_score(y_test,y_pred):.2f}")
print(f"model (SVM) precision : {precision_score(y_test , y_pred):.2f}")
print(f"model (SVM) recall    : {recall_score(y_test, y_pred):.2f}")
```

```
model (SVM) accuracy  : 0.80
model (SVM) precision : 0.82
model (SVM) recall    : 0.79
```

Neural network

In [ ]:

```
from sklearn.neural_network import MLPClassifier
np.random.seed(42)
Nn = MLPClassifier(hidden_layer_sizes=100 , activation='relu'
                   ,alpha= 0.0001 , learning_rate='constant')
# y_pred_Nn = cross_val_predict(Nn , vector , y , cv = 5)
model = Nn.fit(X_train , y_train)
y_pred = model.predict(x_test)
print(f"model (Neural network) accuracy  : {accuracy_score(y_test,y_pred):.2f
print(f"model (Neural network) precision : {precision_score(y_test , y_pred):
print(f"model (Neural network) recall    : {recall_score(y_test ,y_pred):.2f}
```

```
model (Neural network) accuracy  : 0.81
model (Neural network) precision : 0.82
model (Neural network) recall    : 0.80
```

In [ ]:

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
y = data.target
KNN = KNeighborsClassifier(n_neighbors=5)
RFC = RandomForestClassifier(n_estimators=100, criterion="gini")
SVM = SVC(C=1 , kernel="sigmoid" , degree = 3)
MLP = MLPClassifier(hidden_layer_sizes=100 , activation='relu'
                   ,alpha= 0.0001 , learning_rate='constant')
names_model = ["KNeighbors" , "RandomForest" , "SCV" , "Neural Network"]
list_model = [KNN,
              RFC,
              SVM,
              MLP]
for i , model in enumerate(list_model):
    y_pred = cross_val_predict(model , vector , y , cv =5)
    print(f"Name model : {names_model[i]}")
    print("Classification report")
    print(classification_report (y , y_pred , target_names = ['Neg' , 'Pos'])
```

```
Name model : KNeighbors
Classification report
              precision    recall  f1-score   support

         Neg       0.66      0.60      0.63      1000
         Pos       0.64      0.69      0.67      1000

    accuracy                           0.65      2000
   macro avg       0.65      0.65      0.65      2000
weighted avg       0.65      0.65      0.65      2000

Name model : RandomForest
Classification report
              precision    recall  f1-score   support
```

```
         Neg       0.76      0.84      0.80      1000
         Pos       0.82      0.74      0.78      1000

    accuracy                           0.79      2000
   macro avg       0.79      0.79      0.79      2000
weighted avg       0.79      0.79      0.79      2000

Name model : SCV
Classification report
              precision    recall  f1-score   support

         Neg       0.81      0.80      0.81      1000
         Pos       0.80      0.81      0.81      1000

    accuracy                           0.81      2000
   macro avg       0.81      0.81      0.81      2000
weighted avg       0.81      0.81      0.81      2000

Name model : Neural Network
Classification report
              precision    recall  f1-score   support

         Neg       0.82      0.80      0.81      1000
         Pos       0.81      0.82      0.81      1000

    accuracy                           0.81      2000
   macro avg       0.81      0.81      0.81      2000
weighted avg       0.81      0.81      0.81      2000
```

2.5 Model Tuning

Can you try to beat the simple model you created above? Here are some things you may try:

- When creating TfidfVectorizer object, you may tweak sublinear_tf parameter which use the tf with logarithmic scale instead of the usual tf.
- You may also exclude words that are too frequent or too rare, by adjusting max_df and min_df.
- Adjusting parameters available in the model, like neural network structure or number of trees in the forest.

Design at least 3 experiments using these techniques. Show your experimental results.

In [70]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer ,TfidfTransformer

vectorizer = TfidfVectorizer(sublinear_tf=True,stop_words="english" ,max_df =
tf_idf = TfidfTransformer()
vector = tf_idf.fit_transform(vectorizer.fit_transform(data.data)) # x
print(vector.shape)
```

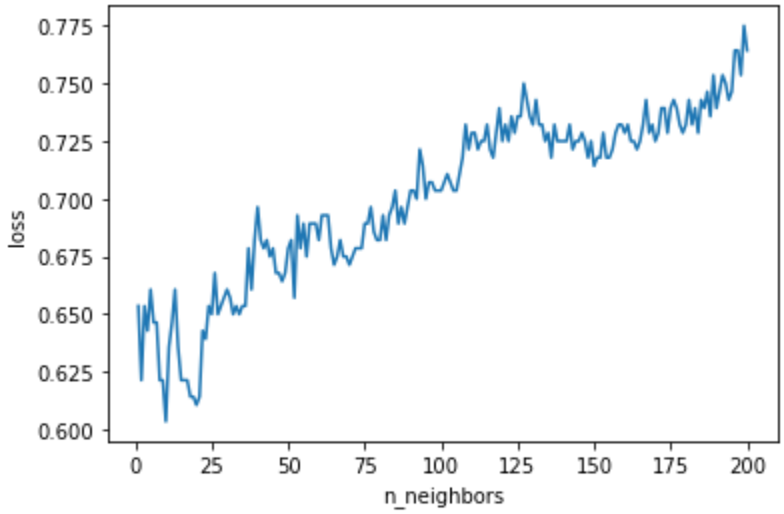
```
(2000, 15570)
```

In [71]:

```python
X_var, x_buff , y_var , y_buff = train_test_split(x_test , y_test , test_size
```

In [72]:
```
# Tune KNN
log_data = {'n_neighbors' : [],
            'loss' : []}
for i in range (1,201):
    KNN = KNeighborsClassifier(n_neighbors=i)
    model = KNN.fit(X_train , y_train)
    y_pred = model.predict(X_var)
    accu = accuracy_score(y_var , y_pred)
    log_data['n_neighbors'].append(i)
    log_data['loss'].append(accu)
plt.plot(log_data['n_neighbors'] ,log_data['loss'])
plt.xlabel('n_neighbors')
plt.ylabel('loss')
```

Out[72]:
```
Text(0, 0.5, 'loss')
```



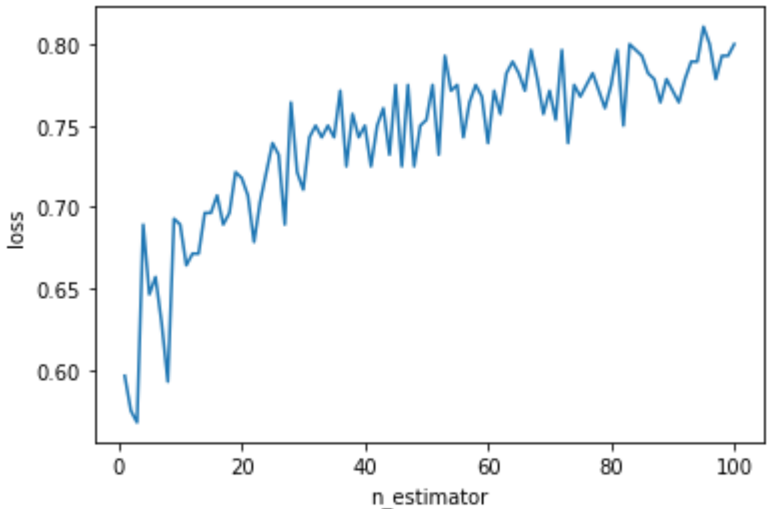
In [76]:
```
from sklearn.metrics import confusion_matrix,classification_report
N = KNeighborsClassifier(n_neighbors= 130)
model = KNN.fit(X_train , y_train)
y_pred = model.predict(x_test)
print(f"Confusion matrix : \
       \n {confusion_matrix(y_test , y_pred)}")
print(f"Classification Report : \
       \n {classification_report (y_test , y_pred , target_names = ['Neg' , '
```

```
Confusion matrix :
 [[153  37]
 [ 54 156]]
Classification Report :
               precision    recall  f1-score   support

         Neg       0.74      0.81      0.77       190
         Pos       0.81      0.74      0.77       210

    accuracy                           0.77       400
   macro avg       0.77      0.77      0.77       400
weighted avg       0.78      0.77      0.77       400
```

In [51]:

```
# Tune Random Forest
log_data = {'n_estimator' : [],
            'loss' : []}

for i in range(1,101):
    model = RandomForestClassifier(n_estimators=i).fit(X_train , y_train)
    y_pred = model.predict(X_var)
    accu = accuracy_score(y_var , y_pred)
    log_data['n_estimator'].append(i)
    log_data['loss'].append(accu)
plt.plot(log_data['n_estimator'] ,log_data['loss'])
plt.xlabel('n_estimator')
plt.ylabel('loss')
```

Out[51]: Text(0, 0.5, 'loss')



In [52]:

```
RFC = RandomForestClassifier(n_estimators=83, criterion="gini")
model = RFC.fit(X_train , y_train)
y_pred = model.predict(x_test)
print(f"Confusion matrix : \
      \n{confusion_matrix(y_test , y_pred)}")
print(f"Classification Report : \
      \n{classification_report (y_test , y_pred , target_names = ['Neg' , 'Po
```

```
Confusion matrix :
[[154  36]
 [ 57 153]]
Classification Report :
              precision    recall  f1-score   support

         Neg       0.73      0.81      0.77       190
         Pos       0.81      0.73      0.77       210

    accuracy                           0.77       400
   macro avg       0.77      0.77      0.77       400
weighted avg       0.77      0.77      0.77       400
```

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.neural_network import MLPClassifier
Nn_tuning = MLPClassifier(max_iter=1000 , early_stopping=True , n_iter_no_cha
hidden_layer_sizes = [(70,55,70),(100)]
activation = ["identity" , "logistic" , "tanh" ,"relu"]
solver = ["lbfgs" , "sgd" ,"adam"]
alpha = [0.001 , 0.005 ]
learning_rate = ['constant' , 'invscaling' ,'adaptive']

param_grid = dict(hidden_layer_sizes= hidden_layer_sizes , activation=activat
                  solver=solver , alpha=alpha , learning_rate = learning_rate
grid = GridSearchCV(Nn_tuning , param_grid ,n_jobs= -1 , cv=10 )
print(param_grid)
grid.fit(X_train , y_train)
```

```
{'hidden_layer_sizes': [(70, 55, 70), 100], 'activation': ['identity', 'logis
tic', 'tanh', 'relu'], 'solver': ['lbfgs', 'sgd', 'adam'], 'alpha': [0.001,
0.005], 'learning_rate': ['constant', 'invscaling', 'adaptive']}
```

Out[ ]:

```
GridSearchCV(cv=10,
             estimator=MLPClassifier(early_stopping=True, max_iter=1000,
                                     n_iter_no_change=20),
             n_jobs=-1,
             param_grid={'activation': ['identity', 'logistic', 'tanh', 'relu
'],
                         'alpha': [0.001, 0.005],
                         'hidden_layer_sizes': [(70, 55, 70), 100],
                         'learning_rate': ['constant', 'invscaling',
                                           'adaptive'],
                         'solver': ['lbfgs', 'sgd', 'adam']})
```

In [ ]:

```
print(f"Good parameters : {grid.best_params_}")
```

```
Good parameters : {'activation': 'relu', 'alpha': 0.005, 'hidden_layer_sizes
': (70, 55, 70), 'learning_rate': 'invscaling', 'solver': 'adam'}
```

3 Text Clustering

We have heard about Google News clustering. In this exercise, we are going to implement it with Python.

3.1 Data Preprocessing

Let's switch up and use another dataset called 20newsgroup data, which is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data is collected from a university's mailing list, where students exchange opinions in everything from motorcycles to middle east politics.

1. Import data using sklearn.datasets.fetch_20newsgroups
2. Transform data to vector with TfidfVectorizer

In [93]:
```
from sklearn.datasets import fetch_20newsgroups

newsgroups_train = fetch_20newsgroups(subset='train')
```

In [94]:
```
data = newsgroups_train.data
```

In [95]:
```
labels = newsgroups_train.target
```

In [78]:
```
labels
```

Out[78]:
```
array([7, 4, 4, ..., 3, 1, 8])
```

In [79]:
```
import numpy as np
true_k = np.unique(labels).shape[0]
true_k
```

Out[79]:
```
20
```

In [80]:
```
newsgroups_train.filenames.shape , newsgroups_train.target.shape
```

Out[80]:
```
((11314,), (11314,))
```

In [81]:
```
from pprint import pprint
pprint(list(newsgroups_train.target_names))
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

In [82]:
```
from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer,T
from sklearn.pipeline import Pipeline
categories = list(newsgroups_train.target_names)
newsgroups_train = fetch_20newsgroups(subset='train' , categories=categories)

vectorizer = TfidfVectorizer( stop_words='english' , analyzer = 'word')
X = vectorizer.fit_transform(newsgroups_train.data)
```

In [83]:
```
idf = vectorizer.idf_
print(dict(zip(vectorizer.get_feature_names(),idf)))
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

In [84]:
```
X.shape
```

Out[84]:
```
(11314, 129796)
```

3.2 Clustering

We are going to use the simplest clustering model, k-means clustering, to do this task. Our hope is that this simple algorithm will result in meaningful news categories, without using labels.

1. Fit K-Means clustering model to the text vector. What is the value of K you should pick? Why?
2. Use Silhouette score to evaluate your clusters. Try to evaluate the model for different values of k to see which k fits best for the dataset.

In [85]:

```python
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(42)
score = []
# ค่า k จริงๆ ที่เราจะใช้เราจะต้องดูตามความเหมาะสม เดี๋ยวจะวน loop เพื่อหาค่า k ที่ดีที่สุด เรา
for i in range(2 , 21, 1):
    Km = KMeans(n_clusters=i )
    %time Km.fit(X)
    silhouette_avg = metrics.silhouette_score(X ,Km.labels_)
    score.append(silhouette_avg)
    print("For n_clusters =", i,
          "The average silhouette_score is :", silhouette_avg)

# print(f"V-measure  : {metrics.v_measure_score(labels , Km.labels_):.2f} ")
# print(f"Silhouette : {metrics.silhouette_score(X ,Km.labels_ , sample_size=
```

```
CPU times: user 38.5 s, sys: 459 ms, total: 39 s
Wall time: 5.77 s
For n_clusters = 2 The average silhouette_score is : 0.0016376013719498323
CPU times: user 39.7 s, sys: 357 ms, total: 40.1 s
Wall time: 5.94 s
For n_clusters = 3 The average silhouette_score is : 0.0019373149769593059
CPU times: user 1min 3s, sys: 618 ms, total: 1min 4s
Wall time: 9.33 s
For n_clusters = 4 The average silhouette_score is : 0.0022808548130325364
CPU times: user 1min 24s, sys: 705 ms, total: 1min 24s
Wall time: 12.5 s
For n_clusters = 5 The average silhouette_score is : 0.0022802429886691973
CPU times: user 1min 38s, sys: 1.03 s, total: 1min 39s
Wall time: 15.1 s
For n_clusters = 6 The average silhouette_score is : 0.002819323912919021
CPU times: user 1min 39s, sys: 1.08 s, total: 1min 40s
Wall time: 15.5 s
For n_clusters = 7 The average silhouette_score is : 0.0031190152118323895
CPU times: user 1min 59s, sys: 1.23 s, total: 2min 1s
Wall time: 18.7 s
For n_clusters = 8 The average silhouette_score is : 0.003801534225081118
CPU times: user 2min 3s, sys: 1.32 s, total: 2min 5s
Wall time: 19.7 s
For n_clusters = 9 The average silhouette_score is : 0.004135617245072058
CPU times: user 2min 6s, sys: 1.36 s, total: 2min 7s
Wall time: 20.3 s
For n_clusters = 10 The average silhouette_score is : 0.0041580040791949074
CPU times: user 2min 35s, sys: 1.7 s, total: 2min 37s
Wall time: 25.5 s
For n_clusters = 11 The average silhouette_score is : 0.004634428715362065
CPU times: user 2min 21s, sys: 1.48 s, total: 2min 22s
Wall time: 23.2 s
For n_clusters = 12 The average silhouette_score is : 0.004855585745390654
CPU times: user 3min 14s, sys: 2.24 s, total: 3min 16s
Wall time: 33.5 s
For n_clusters = 13 The average silhouette_score is : 0.004838180804806398
CPU times: user 3min 2s, sys: 2.27 s, total: 3min 4s
Wall time: 32.4 s
For n_clusters = 14 The average silhouette_score is : 0.005130118901983481
CPU times: user 3min 39s, sys: 2.78 s, total: 3min 42s
```

```
Wall time: 39.9 s
For n_clusters = 15 The average silhouette_score is : 0.005461358890546025
CPU times: user 3min 41s, sys: 4.53 s, total: 3min 46s
Wall time: 43.1 s
For n_clusters = 16 The average silhouette_score is : 0.0063245985555834775
CPU times: user 3min 49s, sys: 5.11 s, total: 3min 54s
Wall time: 44.9 s
For n_clusters = 17 The average silhouette_score is : 0.006621402171186809
CPU times: user 4min 32s, sys: 5.65 s, total: 4min 37s
Wall time: 53.1 s
For n_clusters = 18 The average silhouette_score is : 0.006278427637876618
CPU times: user 4min 18s, sys: 5.69 s, total: 4min 24s
Wall time: 57.2 s
For n_clusters = 19 The average silhouette_score is : 0.0065900728040751535
CPU times: user 4min 9s, sys: 5.87 s, total: 4min 15s
Wall time: 57.4 s
For n_clusters = 20 The average silhouette_score is : 0.007128332081493478
```

In [86]:

```
k = score.index(max(score)) +2
k
```

Out[86]: 20

3.3 Topic Terms

We want to explore each cluster to understand what news articles are in the cluster, what terms are associated with the cluster. This will require a bit of hacking.

1. Use TfidfVectorizer.get feature names to extract words associated with each dimension of the text vector.
2. Extract cluster's centroids using kmeans.cluster centers .
3. For each centroid, print the top 15 words that have the highest frequency.

In [87]:

```
vectorizer.get_feature_names()
```

Out[87]:

```
['00',
 '000',
 '0000',
 '00000',
 '000000',
 '00000000',
 '0000000004',
 '0000000005',
 '00000000b',
 '00000001',
 '00000001b',
 '0000000667',
 '00000010',
 '00000010b',
 '00000011',
 '00000011b',
 '0000001200',
 '00000074',
```

```
'00000093',
'000000e5',
'00000100',
'00000100b',
'00000101',
'00000101b',
'00000110',
'00000110b',
'00000111',
'00000111b',
'00000315',
'000005102000',
'00000510200001',
'000007',
'00000ee5',
'00001000',
'00001000b',
'00001001',
'00001001b',
'00001010',
'00001010b',
'00001011',
'00001011b',
'000010af',
'00001100',
'00001100b',
'00001101',
'00001101b',
'00001110',
'00001110b',
'00001111',
'00001111b',
'000021',
'000042',
'000062david42',
'000094',
'0000vec',
'0001',
'00010000',
'00010000b',
'00010001',
'00010001b',
'00010010',
'00010010b',
'00010011',
'00010011b',
'000100255pixel',
'00010100',
'00010100b',
'00010101',
'00010101b',
'00010110',
'00010110b',
'00010111',
'00010111b',
'00011000',
'00011000b',
'00011001',
'00011001b',
'00011010',
```

```
'00011010b',
'00011011',
'00011011b',
'00011100',
'00011100b',
'00011101',
'00011101b',
'00011110',
'00011110b',
'00011111',
'00011111b',
'00014',
'000152',
'0001mpc',
'0001x7c',
'0002',
'000246',
'000256',
'0003',
'000359',
'0004',
'000406',
'00041032',
'000413',
'0004136',
'0004246',
'0004422',
'00044513',
'0004847546',
'0005',
'0005111312',
'0005111312na1em',
'0005111312na3em',
'000531',
'0005895485',
'000601',
'0007',
'000710',
'00072',
'0007259d',
'00072741',
'00072840',
'00072a27',
'000773b9',
'00077bbe',
'000851',
'0009',
'00090711',
'000956',
'000ds',
'000k',
'000mi',
'000miles',
'000rpm',
'000s',
'000th',
'000usd',
'001',
'0010',
'00100000',
```

```
'00100000b',
'00100001',
'00100001b',
'00100010',
'00100010b',
'00100011',
'00100011b',
'00100100',
'00100100b',
'00100101',
'00100101b',
'00100110',
'00100110b',
'00100111',
'00100111b',
'001004',
'00101000',
'00101000b',
'00101001',
'00101001b',
'00101010',
'00101010b',
'00101011',
'00101011b',
'00101100',
'00101100b',
'00101101',
'00101101b',
'00101110',
'00101110b',
'00101111',
'00101111b',
'0010580b',
'0011',
'00110000',
'00110000b',
'00110001',
'00110001b',
'00110010',
'00110010b',
'00110011',
'00110011b',
'00110100',
'00110100b',
'00110101',
'00110101b',
'00110110',
'00110110b',
'00110111',
'00110111b',
'00111000',
'00111000b',
'00111001',
'00111001b',
'00111010',
'00111010b',
'00111011',
'00111011b',
'00111100',
'00111100b',
```

```
'00111101',
'00111101b',
'00111110',
'00111110b',
'00111111',
'00111111b',
'001116',
'001125',
'001127',
'0012',
'001200201pixel',
'001211',
'001230',
'0013',
'001319',
'001321',
'001323',
'001338',
'0014',
'001428',
'001555',
'001642',
'001707',
'001718',
'001757',
'0018',
'001813',
'001934',
'00196',
'002',
'0020',
'002118',
'002142',
'0022',
'002222',
'002251w',
'0023',
'002302',
'002339',
'0024',
'002651',
'0028',
'002811',
'0029',
'002937',
'002d',
'003',
'0030',
'003015',
'003029',
'00309',
'00314',
'003221',
'003258u19250',
'0033',
'0034',
'003522',
'003719',
'003749',
'0038',
```

```
'003800',
'003848',
'0039',
'004',
'0040000d',
'004021809',
'004158',
'004253agrgb',
'004325',
'0044',
'004418',
'004532',
'004627',
'004808',
'0049',
'005',
'00500',
'005117',
'005131',
'005148',
'005150',
'005204',
'005245',
'005314',
'005512',
'005634',
'0058',
'0059',
'006',
'0060',
'0062',
'00630',
'0065',
'0066',
'0068',
'007',
'0075',
'0076',
'0078',
'0079',
'008',
'00800',
'0082',
'0084',
'008561',
'0086',
'0087',
'0088',
'009',
'00900001',
'0094',
'00969fba',
'0096a95c',
'0096b0f0',
'0096b11b',
'0096b130',
'0096b294',
'0098',
'0099',
'00_',
```

```
'00acearl',
'00am',
'00bjgood',
'00c',
'00each',
'00ecgillespi',
'00ecgillespie',
'00f',
'00g',
'00h',
'00index',
'00lz8bct',
'00m',
'00mbstultz',
'00pm',
'00q',
'00r',
'00tx',
'00tzxb',
'00tzxb2vyn',
'00u',
'00xkv',
'01',
'010',
'0100',
'01000',
'01000000',
'01000000b',
'01000001',
'01000001b',
'01000010',
'01000010b',
'01000011',
'01000011b',
'01000100',
'01000100b',
'01000101',
'01000101b',
'01000110',
'01000110b',
'01000111',
'01000111b',
'01001000',
'01001000b',
'01001001',
'01001001b',
'01001010',
'01001010b',
'01001011',
'01001011b',
'01001100',
'01001100b',
'01001101',
'01001101b',
'01001110',
'01001110b',
'01001111',
'01001111b',
'01002',
'01003',
```

```
 '01010000',
 '01010000b',
 '01010001',
 '01010001b',
 '01010010',
 '01010010b',
 '01010011',
 '01010011b',
 '01010100',
 '01010100b',
 '01010101',
 '01010101b',
 '01010110',
 '01010110b',
 '01010111',
 '01010111b',
 '01011000',
 '01011000b',
 '01011001',
 '01011001b',
 '01011010',
 '01011010b',
 '01011011',
 '01011011b',
 '01011100',
 '01011100b',
 '01011101',
 '01011101b',
 '01011110',
 '01011110b',
 '01011111',
 '01011111b',
 '010116',
 '010235',
 '010256',
 '010305',
 '010326',
 '010329',
 '01050810',
 '010517',
 '010533',
 '010702',
 '010734',
 '010745',
 '01075',
 '010808',
 '010821',
 '010834',
 '0109',
 '010908',
 '010955',
 '011',
 '01100000',
 '01100000b',
 '01100001',
 '01100001b',
 '01100010',
 '01100010b',
 '01100011',
 '01100011b',
```

```
'01100100',
'01100100b',
'01100101',
'01100101b',
'01100110',
'01100110b',
'01100111',
'01100111b',
'01101000',
'01101000b',
'01101001',
'01101001b',
'01101010',
'01101010b',
'01101011',
'01101011b',
'01101100',
'01101100b',
'01101101',
'01101101b',
'01101110',
'01101110b',
'01101111',
'01101111b',
'011033',
'011042',
'0111',
'01110000',
'01110000b',
'01110001',
'01110001b',
'01110010',
'01110010b',
'01110011',
'01110011b',
'01110100',
'01110100b',
'01110101',
'01110101b',
'01110110',
'01110110b',
'01110111',
'01110111b',
'01111000',
'01111000b',
'01111001',
'01111001b',
'01111010',
'01111010b',
'01111011',
'01111011b',
'01111100',
'01111100b',
'01111101',
'01111101b',
'01111110',
'01111110b',
'01111111',
'01111111b',
'011112',
```

```
'0112',
'011255',
'011308pxf3',
'0114',
'0115',
'011605',
'011634edt',
'011653',
'011720',
'011730',
'011743',
'011805',
'011823',
'011855',
'0119',
'012',
'012011',
'012019',
'012344',
'0123456789',
'012451',
'012536',
'012537',
'0126',
'012714',
'012946',
'012c',
'012i',
'013',
'013011',
'013034',
'013037',
'0131',
'013145',
'0134',
'013423tan102',
'013433',
'013559',
'0136',
'013611',
'013651',
'013653rap115',
'013657',
'013752',
'0138',
'013846',
'013847',
'0139',
'013939',
'014',
'014135',
'01420',
'014237',
'014305',
'014506',
'01451',
'01463',
'014638',
'014646',
'014t4',
```

```
'015',
'0150',
'015043',
'015209',
'015225',
'0154',
'015415',
'015442',
'01545',
'015518',
'015551',
'0158',
'01580',
'015844',
'015908',
'015931',
'015936',
'016',
'01609',
'0161',
'0162',
'0164',
'0168',
'01701',
'01720',
'01730',
'01742',
'01752',
'01760',
'01775',
'01776',
'0179',
'018',
'0180',
'01800',
'01801',
'01803',
'01810',
'0182',
'01821',
'01826',
'0183',
'01830',
'0184',
'01852',
'01854',
'0186',
'01867',
'0188',
'01880',
'018801285',
'01890',
'018b',
'019',
'0192',
'0195',
'0199',
'01_0',
'01a',
'01apr93',
```

```
'01c',
'01d',
'01f6',
'01h',
'01h0',
'01h5',
'01ll',
'01ne',
'01o',
'01ob',
'01oe',
'01wb',
'02',
'020',
'0200',
'02000',
'020021',
'020259',
'02026',
'0203',
'020347',
'020356',
'020359',
'020427',
'0205',
'020504',
'020533',
'020555',
'020637',
'020646',
'020655',
'020701tan102',
'020751',
'02086551',
'0209',
'020qw',
'021',
'021021',
'02106',
'021105',
'02115',
'021154',
'02118',
'021225',
'02138',
'02139',
'02142',
'02146',
'02150',
'02154',
'0216',
'02160',
'021635',
'02170289',
'021708',
'02172',
'02173',
'02174',
'02178',
'02180',
```

```
'021846',
'02194',
'021lh2u',
'022',
'022011',
'022113',
'02215',
'022218',
'022222',
'022233',
'0223',
'02238',
'02254',
'0226',
'022621tan102',
'0228',
'02280936',
'022903',
'022922',
'022926',
'022947',
'023',
'023017',
'023039',
'023044',
'0233',
'023428',
'0235',
'0237',
'023730',
'023843',
'0239',
'023937',
'023_',
'023b',
'024',
'024036',
'024103',
'024128',
'024150',
'024222',
'024246',
'024257',
'024423',
'0245',
'02451203',
'024626',
'024646',
'024850',
'024949',
'025',
'0250',
'025027',
'025031',
'025240',
'025258',
'025426',
'025636',
'025818u28037',
'025924',
```

```
'02678944',
'027',
'0272',
'0273',
'0278',
'0279',
'02790',
'027w',
'028',
'0280',
'0283',
'0284',
'029',
'02903',
'02908',
'02917',
'0293',
'02_',
'02_8v',
'02_9',
'02at',
'02bp1m51',
'02bz',
'02cents',
'02d',
'02h',
'02ixl',
'02j',
'02l',
'02m',
'02njp',
'02p',
'02p4',
'02qvq',
'02r',
'02r4e',
'02s',
'02t',
'02tl',
'02tm_',
'02tmn',
'02u',
'02uv',
'02v',
'02va7pu',
'02vx',
'02vy',
'02vyn',
'02vz089',
'02w',
'02x',
'03',
'030',
'0300',
'03000',
'030031',
'0300ff',
'030105',
'0302',
'030204',
```

```
'0303',
'030334',
'030412',
'03051',
'030538',
'03054',
'0306',
'030636',
'030703',
'030706',
'030733mcartwr',
'030734',
'03083',
'030934',
'031',
'031349',
'0314',
'031404',
'031423',
'031520',
'031524',
'031616',
'0317',
'031744',
'0318',
'031823',
'031840',
'031905saundrsg',
'031l86',
'0320',
'032017',
'032022',
'032251',
'032345',
'032350',
'0324',
'032405',
'032554',
'032616mbs110',
'032620',
'032623',
'032746',
'032828',
'032905',
'033',
'0330',
'0332',
'033213',
'033230kevxu',
'033258',
'0334',
'033446',
'0335',
'0338',
'033802',
'033843',
'034',
'0340',
'034101',
'034226',
```

```
'034352',
'034517',
'0346',
'034614',
'0347',
'034724',
'034751',
'0349',
'035',
'0350',
'035020',
'035125',
'0353',
'035406',
'035544',
'0356',
'0357',
'0358',
'035941',
'036',
'0362',
'0366',
'0369',
'037',
'0372',
'03756',
'038',
'0380',
'03800',
'0382',
'03833',
'0384',
'038o8v',
'039',
'0391',
'0395',
'03_',
'03e8',
'03f',
'03ho',
'03hord',
'03hz',
'03hzri',
'03i',
'03i3',
'03ii',
'03imv',
'03is',
'03j1',
'03j1d9',
'03k',
'03k8',
'03k8rg',
'03m4u',
'03o',
'03p',
'03t',
'03u',
'03u0',
'03vo',
```

```
 '03y',
 '03yqu',
 '04',
 '040',
 '0400',
 '0401',
 '040118',
 '040231',
 '040254',
 '040286',
 '0404',
 '040414',
 '040449',
 '04046',
 '040493023201',
 '040493161915',
 '0405',
 '040606',
 '040819',
 '040839',
 '040946',
 '041',
 '0410',
 '041033',
 '0411',
 '04110',
 '041300',
 '041343',
 '0414',
 '041411',
 '041493003715',
 '041505',
 '041741',
 '0418',
 '041810',
 '041929',
 '042',
 '042100',
 '0422',
 '042234',
 '0423',
 '042427',
 '042450',
 '04255',
 '042624',
 '042722mcartwr',
 '042749',
 '042918',
 '043',
 '0430',
 '043112',
 '0433nl',
 '0434',
 '043426',
 '0435',
 '043642',
 '043654',
 '0437',
 '043740',
 '0437643',
```

```
 '043935',
 '044',
 '0440',
 '044001',
 '044018',
 '044045',
 '044140',
 '044201',
 '044248',
 '0443',
 '044323',
 '0444',
 '044405',
 '0446',
 '044636',
 '04473',
 '044749',
 '044946',
 '044958',
 '045',
 '0450',
 '045019',
```

In [88]:

```python
centroids = Km.cluster_centers_.argsort()[:,::-1]
terms = vectorizer.get_feature_names()

for i in range(k):
    print(f"Cluster : {i}")
    for j in centroids[i,:15]:
        print(f'{terms[j]}')
    print("\n")
```

```
Cluster : 0
keith
caltech
livesey
morality
sgi
objective
solntze
wpd
jon
schneider
cco
allan
moral
edu
atheists


Cluster : 1
car
com
cars
edu
___
radar
engine
writes
```

```
just
article
dealer
good
like
don
ca


Cluster : 2
com
gun
people
edu
don
government
guns
think
writes
msg
article
just
like
fbi
stratus


Cluster : 3
columbia
gld
cunixb
cc
dare
gary
edu
keenan
cunixa
insurance
pgf5
garfiel
domi
cunixc
freeman


Cluster : 4
virginia
cramer
optilink
clayton
kaldis
edu
gay
men
homosexual
clas
rutgers
sexual
com
gsh7w
```

```
university


Cluster : 5
key
clipper
encryption
chip
escrow
keys
government
com
crypto
algorithm
intercon
secure
nsa
des
amanda


Cluster : 6
mouse
driver
windows
port
edu
com1
cursor
com3
adb
com
irq
diamond
serial
nlm
sys


Cluster : 7
team
game
edu
ca
hockey
players
games
year
play
season
baseball
nhl
win
league
teams


Cluster : 8
window
motif
```

```
widget
application
server
mit
manager
xterm
com
dresden
tu
uk
problem
windows
edu


Cluster : 9
edu
com
subject
lines
organization
university
posting
host
nntp
article
writes
know
cs
thanks
like


Cluster : 10
god
jesus
christians
bible
people
christian
christ
faith
believe
edu
church
christianity
life
don
say


Cluster : 11
windows
dos
file
access
files
edu
card
com
```

```
digex
graphics
drivers
program
use
ms
lines


Cluster : 12
bike
com
dod
sun
edu
behanna
article
ride
nec
helmet
ca
writes
motorcycle
dog
bikes


Cluster : 13
ohio
cleveland
cwru
edu
magnus
state
acs
freenet
reserve
ins
western
case
university
po
nntp


Cluster : 14
henry
alaska
toronto
zoo
spencer
aurora
zoology
nsmca
edu
space
acad3
moon
utzoo
work
```

```
kipling


Cluster : 15
israel
israeli
jews
arab
jake
arabs
edu
lebanese
adam
israelis
policy
cpr
jewish
lebanon
hernlem


Cluster : 16
drive
scsi
ide
controller
drives
bus
hard
disk
floppy
edu
hd
isa
mac
problem
motherboard


Cluster : 17
turkish
armenian
armenians
armenia
turks
argic
serdar
turkey
genocide
zuma
sera
soviet
people
greek
azerbaijan


Cluster : 18
nasa
gov
```

```
space
jpl
larc
baalke
jsc
gsfc
higgins
kelvin
fnal
center
research
howland
propulsion


Cluster : 19
geb
banks
gordon
pitt
cs
dsl
n3jxp
cadre
chastity
shameful
skepticism
intellect
surrender
edu
pittsburgh
```

In [96]:

```
test_data = vectorizer.transform([data[400]])
Km = KMeans(n_clusters=20)
model = Km.fit(X)
clu = model.predict(test_data)[0]
clu
```

Out[96]: 7

In [97]:

```
data[400]
```

Out[97]:

```
"From: hooper@ccs.QueensU.CA (Andy Hooper)\nSubject: Re: text of White House announcement and Q&As on clipper chip encryption\nOrganization: Queen's University, Kingston\nDistribution: na\nLines: 3\n\nIsn't Clipper a trademark of Fairchild Semiconductor?\n\nAndy Hooper\n"
```

In [ ]: