All Class Tasks

Natural Language Processing

Name: K.N.Sri Dharmasastha
Registration Number: 16BCE0956

BROWN Corpus Subsections

In [1]:
```
import nltk
from nltk.corpus import brown
brown.categories()
```

Out[1]:
```
['adventure',
 'belles_lettres',
 'editorial',
 'fiction',
 'government',
 'hobbies',
 'humor',
 'learned',
 'lore',
 'mystery',
 'news',
 'religion',
 'reviews',
 'romance',
 'science_fiction']
```

BROWN Corpus – loading data from subsections

```
from nltk.corpus import brown
news_text = brown.words(categories='news')
print(news_text)
```

```
['The', 'Fulton', 'County', 'Grand', 'Jury', 'said', ...]
```

INAUGURAL Corpus – Subsections

In [22]:
```
from nltk.corpus import inaugural
inaugural.fileids()
```

Out[22]:
```
['1789-Washington.txt',
 '1793-Washington.txt',
 '1797-Adams.txt',
 '1801-Jefferson.txt',
 '1805-Jefferson.txt',
 '1809-Madison.txt',
 '1813-Madison.txt',
 '1817-Monroe.txt',
 '1821-Monroe.txt',
 '1825-Adams.txt',
 '1829-Jackson.txt',
 '1833-Jackson.txt',
 '1837-VanBuren.txt',
 '1841-Harrison.txt',
 '1845-Polk.txt',
 '1849-Taylor.txt',
 '1853-Pierce.txt',
 '1857-Buchanan.txt',
 '1861-Lincoln.txt',
 '1865-Lincoln.txt',
 '1869-Grant.txt',
 '1873-Grant.txt',
 '1877-Hayes.txt',
 '1881-Garfield.txt',
 '1885-Cleveland.txt',
 '1889-Harrison.txt',
 '1893-Cleveland.txt',
```

INAUGURAL Corpus – Loading a file

In [29]:
```
from nltk.corpus import inaugural
inaugural.raw('1789-Washington.txt')
```

Out[29]:
```
'Fellow-Citizens of the Senate and of the House of Representatives:\n\nAmong the vicissitudes incident to life no event could h
ave filled me with greater anxieties than that of which the notification was transmitted by your order, and received on the 14t
h day of the present month. On the one hand, I was summoned by my Country, whose voice I can never hear but with veneration and
love, from a retreat which I had chosen with the fondest predilection, and, in my flattering hopes, with an immutable decision,
as the asylum of my declining years -- a retreat which was rendered every day more necessary as well as more dear to me by the
addition of habit to inclination, and of frequent interruptions in my health to the gradual waste committed on it by time. On t
he other hand, the magnitude and difficulty of the trust to which the voice of my country called me, being sufficient to awaken
in the wisest and most experienced of her citizens a distrustful scrutiny into his qualifications, could not but overwhelm with
despondence one who (inheriting inferior endowments from nature and unpracticed in the duties of civil administration) ought to
be peculiarly conscious of his own deficiencies. In this conflict of emotions all I dare aver is that it has been my faithful s
tudy to collect my duty from a just appreciation of every circumstance by which it might be affected. All I dare hope is that i
f, in executing this task, I have been too much swayed by a grateful remembrance of former instances, or by an affectionate sen
sibility to this transcendent proof of the confidence of my fellow citizens, and have thence too little consulted my incapacity
as well as disinclination for the weighty and untried cares before me, my error will be palliated by the motives which mislead
me, and its consequences be judged by my country with some share of the partiality in which they originated.\n\nSuch being the
impressions under which I have, in obedience to the public summons, repaired to the present station, it would be peculiarly imp
roper to omit in this first official act my fervent supplications to that Almighty Being who rules over the universe, who presi
des in the councils of nations, and whose providential aids can supply every human defect, that His benediction may consecrate
to the liberties and happiness of the people of the United States a Government instituted by themselves for these essential pur
poses, and may enable every instrument employed in its administration to execute with success the functions allotted to his cha
rge. In tendering this homage to the Great Author of every public and private good, I assure myself that it expresses your sent
iments not less than my own, nor those of my fellow citizens at large less than either. No people can be bound to acknowledge a
nd adore the Invisible Hand which conducts the affairs of men more than those of the United States. Every step by which they ha
ve advanced to the character of an independent nation seems to have been distinguished by some token of providential agency; an
d in the important revolution just accomplished in the system of their united government the tranquil deliberations and volunta
ry consent of so many distinct communities from which the event has resulted can not be compared with the means by which most g
overnments have been established without some return of pious gratitude, along with an humble anticipation of the future blessi
ngs which the past seem to presage. These reflections, arising out of the present crisis, have forced themselves too strongly o
n my mind to be suppressed. You will join with me, I trust, in thinking that there are none under the influence of which the pr
oceedings of a new and free government can more auspiciously commence.\n\nBy the article establishing the executive department
it is made the duty of the President "to recommend to your consideration such measures as he shall judge necessary and expedien
t." The circumstances under which I now meet you will acquit me from entering into that subject further than to refer to the gr
eat constitutional charter under which you are assembled, and which, in defining your powers, designates the objects to which y
our attention is to be given. It will be more consistent with those circumstances, and far more congenial with the feelings whi
ch actuate me, to substitute, in place of a recommendation of particular measures, the tribute that is due to the talents, the
rectitude, and the patriotism which adorn the characters selected to devise and adopt them. In these honorable qualifications I
behold the surest pledges that as on one side no local prejudices or attachments, no separate views nor party animosities, will
```

Pipelining

i. Code: (Tokenizing)

In [27]:
```
import nltk
sent = "#fun#party"
words = nltk.word_tokenize(sent)
print (words)
```

```
['#', 'fun', '#', 'party']
```

ii. Code: (PoS)

In [2]:
```
import nltk
tokens = nltk.word_tokenize("Can you order a pizza for me? I'll pay you back.")
print("Parts fo Speech: ", nltk.pos_tag(tokens))
```

```
Parts fo Speech:  [('Can', 'MD'), ('you', 'PRP'), ('order', 'NN'), ('a', 'DT'), ('pizza', 'NN'), ('for', 'IN'), ('me', 'PRP'),
('?', '.'), ('I', 'PRP'), ("'ll", 'MD'), ('pay', 'VB'), ('you', 'PRP'), ('back', 'RB'), ('.', '.')]
```

Lexicons

i. Stopwords:

```
In [4]: import nltk
        from nltk.corpus import stopwords
        stopwords.words('swedish')
```

```
Out[4]: ['och',
         'det',
         'att',
         'i',
         'en',
         'jag',
         'hon',
         'som',
         'han',
         'på',
         'den',
         'med',
         'var',
         'sig',
         'för',
         'så',
         'till',
         'är',
         'men',
```

ii. cmudict:

```
In [13]: entries = nltk.corpus.cmudict.entries()
         len(entries)
```

```
Out[13]: 133737
```

```
In [14]: for entry in entries[10000:10025]:
             print(entry)
```

```
('belford', ['B', 'EH1', 'L', 'F', 'ER0', 'D'])
('belfry', ['B', 'EH1', 'L', 'F', 'R', 'IY0'])
('belgacom', ['B', 'EH1', 'L', 'G', 'AH0', 'K', 'AA0', 'M'])
('belgacom', ['B', 'EH1', 'L', 'JH', 'AH0', 'K', 'AA0', 'M'])
('belgard', ['B', 'EH0', 'L', 'G', 'AA1', 'R', 'D'])
('belgarde', ['B', 'EH0', 'L', 'G', 'AA1', 'R', 'D', 'IY0'])
('belge', ['B', 'EH1', 'L', 'JH', 'IY0'])
('belger', ['B', 'EH1', 'L', 'G', 'ER0'])
('belgian', ['B', 'EH1', 'L', 'JH', 'AH0', 'N'])
('belgians', ['B', 'EH1', 'L', 'JH', 'AH0', 'N', 'Z'])
('belgique', ['B', 'EH0', 'L', 'ZH', 'IY1', 'K'])
("belgique's", ['B', 'EH0', 'L', 'JH', 'IY1', 'K', 'S'])
('belgium', ['B', 'EH1', 'L', 'JH', 'AH0', 'M'])
("belgium's", ['B', 'EH1', 'L', 'JH', 'AH0', 'M', 'Z'])
('belgo', ['B', 'EH1', 'L', 'G', 'OW2'])
('belgrade', ['B', 'EH1', 'L', 'G', 'R', 'EY0', 'D'])
('belgrade', ['B', 'EH1', 'L', 'G', 'R', 'AA2', 'D'])
("belgrade's", ['B', 'EH1', 'L', 'G', 'R', 'EY0', 'D', 'Z'])
("belgrade's", ['B', 'EH1', 'L', 'G', 'R', 'AA2', 'D', 'Z'])
('belgrave', ['B', 'EH1', 'L', 'G', 'R', 'EY2', 'V'])
('beli', ['B', 'EH1', 'L', 'IY0'])
('belich', ['B', 'EH1', 'L', 'IH0', 'K'])
('belie', ['B', 'IH0', 'L', 'AY1'])
('belied', ['B', 'IH0', 'L', 'AY1', 'D'])
('belief', ['B', 'IH0', 'L', 'IY1', 'F'])
```

iii. Wordnet:

```
In [17]: from nltk.corpus import wordnet as wn
         wn.synsets('left') # you get id for subsets #gets synonyms
```

```
Out[17]: [Synset('left.n.01'),
          Synset('left.n.02'),
          Synset('left.n.03'),
          Synset('left_field.n.01'),
          Synset('left.n.05'),
          Synset('leave.v.01'),
          Synset('leave.v.02'),
          Synset('leave.v.03'),
          Synset('leave.v.04'),
          Synset('exit.v.01'),
          Synset('leave.v.06'),
          Synset('leave.v.07'),
          Synset('leave.v.08'),
          Synset('entrust.v.02'),
          Synset('bequeath.v.01'),
          Synset('leave.v.11'),
          Synset('leave.v.12'),
          Synset('impart.v.01'),
          Synset('forget.v.04'),
          Synset('left.a.01'),
          Synset('leftover.s.01'),
          Synset('left.s.03'),
          Synset('left.a.04'),
          Synset('left.r.01')]
```

```
In [18]: wn.synset('left.n.05').lemma_names()
```

```
Out[18]: ['left']
```

```
In [19]: wn.synset('bequeath.v.01').lemma_names()
```

```
Out[19]: ['bequeath', 'will', 'leave']
```

TweetTokenizer

```
In [27]: import nltk
         sent = "#fun#party"
         words = nltk.word_tokenize(sent)
         print (words)
```

```
['#', 'fun', '#', 'party']
```

```
In [28]: import nltk
         from nltk.tokenize import TweetTokenizer
         text = 'The party was sooo  fun :D #superfun'
         twtkn = TweetTokenizer()
         twtkn.tokenize(text)
```

```
Out[28]: ['The', 'party', 'was', 'sooo', 'fun', ':D', '#superfun']
```

```
In [2]: import nltk
        tokens = nltk.word_tokenize("Can you order a pizza for me? I'll pay you back.")
        print("Parts fo Speech: ", nltk.pos_tag(tokens))
```

```
Parts fo Speech:  [('Can', 'MD'), ('you', 'PRP'), ('order', 'NN'), ('a', 'DT'), ('pizza', 'NN'), ('for', 'IN'), ('me', 'PRP'),
('?', '.'), ('I', 'PRP'), ("'ll", 'MD'), ('pay', 'VB'), ('you', 'PRP'), ('back', 'RB'), ('.', '.')]
```

Stemmers:

Porter Stemmer:

In [1]:
```
import nltk
from nltk.stem import  PorterStemmer
Stemmerporter = PorterStemmer()
Stemmerporter.stem('happiness')
```

Out[1]: 'happi'

In [2]:
```
import nltk
from nltk.stem import  PorterStemmer
Stemmerporter = PorterStemmer()
Stemmerporter.stem('happily')
```

Out[2]: 'happili'

In [3]:
```
import nltk
from nltk.stem import  PorterStemmer
Stemmerporter = PorterStemmer()
Stemmerporter.stem('ugly')
```

Out[3]: 'ugli'

In [4]:
```
import nltk
from nltk.stem import  PorterStemmer
Stemmerporter = PorterStemmer()
Stemmerporter.stem('munchies')
```

Out[4]: 'munchi'

Lancaster Stemmer:

In [7]:
```
import nltk
from nltk.stem import  LancasterStemmer
StemmerLancaster = LancasterStemmer()
StemmerLancaster.stem('happily')
```

Out[7]: 'happy'

In [8]:
```
import nltk
from nltk.stem import  LancasterStemmer
StemmerLancaster = LancasterStemmer()
StemmerLancaster.stem('ugly')
```

Out[8]: 'ug'

In [10]:
```
import nltk
from nltk.stem import  LancasterStemmer
StemmerLancaster = LancasterStemmer()
StemmerLancaster.stem('munchies')
```

Out[10]: 'munchy'

In [11]:
```
import nltk
from nltk.stem import  LancasterStemmer
StemmerLancaster = LancasterStemmer()
StemmerLancaster.stem('busses')
```

Out[11]: 'buss'

In [12]:
```
import nltk
from nltk.stem import  LancasterStemmer
StemmerLancaster = LancasterStemmer()
StemmerLancaster.stem('fussing')
```

Out[12]: 'fuss'

Regex Stemmer:

In [19]:
```
import nltk
from nltk.stem import  RegexpStemmer
Stemmerregexp = RegexpStemmer('ing')
Stemmerregexp.stem('singing')
```

Out[19]: 's'

Snowball Stemmer:

In [26]:
```
import nltk
from nltk.stem import  SnowballStemmer
SnowballStemmer.languages
frenchstemmer = SnowballStemmer('french')
frenchstemmer.stem('dormez')
```

Out[26]: 'dorm'

Text Classification:

In [9]:
```
from nltk.corpus import names
def gender_features(word):
    return {'last_letter':word[-1]}
```

In [12]:
```
from nltk.corpus import names
labeled_names = ([(name, 'male') for name in names.words('male.txt')]+[(name, 'female') for name in names.words('female.txt')])
```

In [13]:
```
import random
random.shuffle(labeled_names)
```

In [17]:
```
featuresets = [(gender_features(n), gender) for (n,gender) in labeled_names]
train_set, test_set = featuresets[500:], featuresets[:500]
```

In [18]:
```
import nltk
classifier = nltk.NaiveBayesClassifier.train(train_set)
classifier.classify(gender_features('David'))
classifier.classify(gender_features('Michelle'))
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.734
```

Vectorizer:

In [1]:
```
import nltk
from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

example = "The cat was chasing a mouse"
example = [stemmer.stem(token) for token in example.split(" ")]
```

In [2]:
```
print(" ".join(example))
```

```
the cat wa chase a mous
```

In [3]:
```
import nltk
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

example = "The cat was chasing the mouse. There were cacti round the corner."
example = [lemmatizer.lemmatize(token) for token in example.split(" ")]
```

In [4]:
```
print(" ".join(example))
```

```
The cat wa chasing the mouse. There were cactus round the corner.
```

In [5]:
```
print(lemmatizer.lemmatize('better', pos = 'a'))
```

```
good
```

In [6]:
```
from sklearn.feature_extraction.text import CountVectorizer
#from sklearn.feature_extraction.text import TfidVectorizer
```

In [13]:
```
vect = CountVectorizer(binary = True) #Explore the other parameters
corpus = ["Tessaract is an optical character recognition engine", "deduction skills are important for a detective"]
vect.fit(corpus)
print(vect.transform(["Detectives are useful"]).toarray())
#print(vect.transform(corpus).toarray())
```

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

In [14]:
```
vocab = vect.vocabulary_ #vocabulary_ built-in
for key in sorted(vocab.keys()):
    print("{}:{}".format(key, vocab[key]))
```

```
an:0
are:1
character:2
deduction:3
detective:4
engine:5
for:6
important:7
is:8
optical:9
recognition:10
skills:11
tessaract:12
```

In [15]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [18]:
```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vect.transform(["Tessaract is an optical character recognition engine"]).toarray(), vect.transfor
print(similarity)
```

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

Similarity Check Between Articles To Decide Whether To Hire Or Not

In [1]:
```
import nltk
from nltk.corpus import stopwords
from nltk.corpus import inaugural
from nltk.stem.snowball import SnowballStemmer
import numpy as np
from collections import defaultdict

def pre(sam):
    tokens=nltk.word_tokenize(sam)
    words=[w.lower() for w in tokens if w.isalpha()]
    stop_words=set(stopwords.words('english'))
    fill=[w for w in words if not w in stop_words]
    #sb=SnowballStemmer('english')
    #snowball=[sb.stem(data) for data in fill]
    count=nltk.defaultdict(int)
    for word in fill:
        count[word]+=1
    return(count)
def cosine(x,y):
    product=np.dot(x,y)
    mag1=np.linalg.norm(x)
    mag2=np.linalg.norm(y)
    return(product/(mag1*mag2))
def similarity(s1,s2):
    words=[]
    for key in s1:
        words.append(key)
    for key in s2:
        words.append(key)
    v1=np.zeros(len(words),dtype=int)
    v2=np.zeros(len(words),dtype=int)
    i=0
    for (key) in words:
        v1[i]=s1.get(key,0)
        v2[i]=s2.get(key,0)
        i=i+1
    return(cosine(v1,v2))

def main():
    s1=pre('data1.txt')
    #sx=inaugural.fileids()
    #for file in sx:
    s2=pre('data2.txt')
    #inter=set(s1) & set(s2)
    similarity1=similarity(s1,s2)
    print(similarity1,s2)

if __name__=='__main__':
    main()
```

```
nan defaultdict(<class 'int'>, {})
```

In [2]:
```python
from sklearn.feature_extraction.text import CountVectorizer
```

In [3]:
```python
vect = CountVectorizer(binary = True) #Explore the other parameters
data1 = """Summer is a charming flirt. Easy-going and casual. Summer doesn’t huff and puff to win our affections. It has us at “

The season’s reputation precedes itself, and often, not in a good way. It has a way of whittling down everything to its bare bon

Winter travel is an exercise in negotiation, especially for sunshine souls. “How many extra clothes do I have to pack now?” “The

The allure of winter lies in nature—so immense, overwhelming and, of course, achingly beautiful. In his collection of letters to

Solitary revelations aside, winter can match summer for convivial fun. Think of Yuletide. Is there a more exuberant time of the

Our October edition has delightful winter escapes, spanning some of the prettiest landscapes in Europe. Finland is an underrated

Unlike summer, winter is not a flash in the pan. But if those great novels of the past are any indication, you will get lost in

data2 = """If there is a phrase I would prefer to retire from online bios, personal or professional, it is, “I love travel.” Or

In February, the month of love as endowed by our great gifting industrial complex, we are wrestling with what “love for travel”

This world, however, is not the most conducive for long-term passion, the kind that demands unflinching sustenance in the midst

Travellers for life are compulsive. They have to be; there is no other existence. Climate change, marriages, deaths, protests an

In finding stories about travel as a lifetime affair, we looked for longevity. We chose accounts of love, which tempered by the

Falling in love with travel is much like falling in love with a song. You wear it out until it makes you sick. But you have to b

data3 = """One of the finer books I read this year was John Kaag’s Hiking With Nietzsche, in which Kaag, a professor of philosop

In the book, Kaag quotes Neitzsche writing to his mother after he had spent time in Splügen, “I was overcome by the desire to re

Travel writing’s bogeyman, I have been told again and again, is the Internet. Information and narrative are in abundance from to

What must then a travelogue accomplish? Kaag’s book reminded me that it was to establish an intimate and immediate human connect

As another year dawns, NGTI has an exhaustive compendium of places, beloved and obscure, from across India and the world that sh
corpus = [data1, data2, data3]
vect.fit(corpus)
print(vect.transform(["Detectives are useful"]).toarray())
```

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

In [4]:
```
vocab = vect.vocabulary_ #vocabulary_ built-in
for key in sorted(vocab.keys()):
    print("{}:{}".format(key, vocab[key]))
```

```
2019:0
about:1
abundance:2
accomplish:3
accounts:4
achingly:5
acolyte:6
across:7
activities:8
actually:9
addition:10
adequate:11
admiration:12
adulthood:13
adventurer:14
affair:15
affecting:16
affection:17
affections:18
affinity:19
```

In [5]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [6]:
```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vect.transform([data3]).toarray(), vect.transform([data2]))
if similarity<0.3:
    print("Similarity is: {}, Hire!".format(similarity))
else:
    print("Similarity is: {}, Do not hire!".format(similarity))
```

```
Similarity is: [[0.24532669]], Hire!
```