All Class Tasks

Natural Language Processing (NLP)

16BCE0330

Sharan Kapoor

BROWN Corpus Subsections

```
>>> import nltk
>>> from nltk.corpus import brown
>>> brown.categories()
['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

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
['1789-Washington.txt', '1793-Washington.txt', '1797-Adams.txt', '1801-Jefferson
.txt', '1805-Jefferson.txt', '1809-Madison.txt', '1813-Madison.txt', '1817-Monro
e.txt', '1821-Monroe.txt', '1825-Adams.txt', '1829-Jackson.txt', '1833-Jackson.t
xt', '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', '1
885-Cleveland.txt', '1889-Harrison.txt', '1893-Cleveland.txt', '1897-McKinley.tx
t', '1901-McKinley.txt', '1905-Roosevelt.txt', '1909-Taft.txt', '1913-Wilson.txt
', '1917-Wilson.txt', '1921-Harding.txt', '1925-Coolidge.txt', '1929-Hoover.txt'
, '1933-Roosevelt.txt', '1937-Roosevelt.txt', '1941-Roosevelt.txt', '1945-Roosev
elt.txt', '1949-Truman.txt', '1953-Eisenhower.txt', '1957-Eisenhower.txt', '1961
-Kennedy.txt', '1965-Johnson.txt', '1969-Nixon.txt', '1973-Nixon.txt', '1977-Car
ter.txt', '1981-Reagan.txt', '1985-Reagan.txt', '1989-Bush.txt', '1993-Clinton.t
xt', '1997-Clinton.txt', '2001-Bush.txt', '2005-Bush.txt', '2009-Obama.txt']
>>>
```

INAUGURAL Corpus – Loading a file

```
>>> from nltk.corpus import inaugural
>>> inaugural.raw('1789-Washington.txt')
'Fellow-Citizens of the Senate and of the House of Representatives:\n\nAmong the
vicissitudes incident to life no event could have filled me with greater anxietie
s than that of which the notification was transmitted by your order, and received
on the 14th day of the present month. On the one hand, I was summoned by my Count
ry, whose voice I can never hear but with veneration and love, from a retreat whi
ch 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 h
abit to inclination, and of frequent interruptions in my health to the gradual wa
ste committed on it by time. On the other hand, the magnitude and difficulty of t
he trust to which the voice of my country called me, being sufficient to awaken i
n the wisest and most experienced of her citizens a distrustful scrutiny into his
qualifications, could not but overwhelm with despondence one who (inheriting infe
rior endowments from nature and unpracticed in the duties of civil administration
) ought to be peculiarly conscious of his own deficiencies. In this conflict of e
motions all I dare aver is that it has been my faithful study to collect my duty
```

Pipelining

i. Code: (Tokenizing)

```
>>> import nltk
>>> sent="#fun#party"
>>> words=nltk.word_tokenize(sent)
>>> print(words)
['#', 'fun', '#', 'party']
>>>
```


ii. Code: (PoS)

```
>>> import nltk
>>> tokens=nltk.word_tokenize("CAn you order a Pizza for me?")
>>> print("Parts of Speech: ", nltk.pos_tag(tokens))
Parts of Speech:  [('CAn', 'MD'), ('you', 'PRP'), ('order', 'NN'), ('a', 'DT'), (
'Pizza', 'NN'), ('for', 'IN'), ('me', 'PRP'), ('?', '.')]
>>>
```

Lexicons

i. Stopwords:

```
>>> import nltk
>>> from nltk.corpus import stopwords
>>> stopwords.words('english')
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "y
ou've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him'
, 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its'
, 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'wh
o', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was'
, 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'd
id', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until
', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into
', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up',
'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'o
nce', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each
', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'o
wn', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don',
"don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain
', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't",
'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mi
ghtn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'sho
uldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn
', "wouldn't"]
>>>
```


ii. cmudict:

```
>>> entries=nltk.corpus.cmudict.entries()
>>> len(entries)
133737
>>> for entry in entries[10000:10010]:
        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'])
>>>
```

iii. Wordnet:

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('left')
[Synset('left.n.01'), Synset('left.n.02'), Synset('left.n.03'), Synset('left_fiel
d.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('beque
ath.v.01'), Synset('leave.v.11'), Synset('leave.v.12'), Synset('impart.v.01'), Sy
nset('forget.v.04'), Synset('left.a.01'), Synset('leftover.s.01'), Synset('left.s
.03'), Synset('left.a.04'), Synset('left.r.01')]
>>> wn.synset('left.n.05').lemma_names()
['left']
>>> wn.synset('bequeath.v.01').lemma_names()
['bequeath', 'will', 'leave']
>>>
```

TweetTokenizer

```
>>> import nltk
>>> sent="#fun#party"
>>> words=nltk.word_tokenize(sent)
>>> print(words)
['#', 'fun', '#', 'party']
>>>
>>> from nltk.tokenize import TweetTokenizer
>>> text = 'The party was soo fun :D #superfun'
>>> twtkn = TweetTokenizer()
>>> twtkn.tokenize(text)
['The', 'party', 'was', 'soo', 'fun', ':D', '#superfun']
>>> tokens=nltk.word_tokenize("CAn you order a Pizza for me?")
>>> print("Parts of Speech: ", nltk.pos_tag(tokens))
Parts of Speech:  [('CAn', 'MD'), ('you', 'PRP'), ('order', 'NN'), ('a', 'DT'), (
'Pizza', 'NN'), ('for', 'IN'), ('me', 'PRP'), ('?', '.')]
>>>
```

Stemmers:

Porter Stemmer:

```
>>> import nltk
>>> from nltk.stem import PorterStemmer as ps
>>> s=ps()
>>> s.stem('happiness')
'happi'
>>> s.stem('happily')
'happili'
>>> s.stem('ugly')
'ugli'
>>> s.stem('munchies')
'munchi'
>>>
```


Lancaster Stemmer:

```
>>> import nltk
>>> from nltk.stem import LancasterStemmer as ls
>>> s=ls()
>>> s.stem('happily')
'happy'
>>> s.stem('ugly')
'ug'
>>> s.stem('munchies')
'munchy'
>>> s.stem('busses')
'buss'
>>> s.stem('fussing')
'fuss'
>>>
```


Regex Stemmer:

```
>>> import nltk
>>> from nltk.stem import RegexpStemmer
>>> s=RegexpStemmer('ing')
>>> s.stem('singing')
's'
>>>
```


Snowball Stemmer:

```
>>> import nltk
>>> from nltk.stem import SnowballStemmer as sb
>>> sb.languages
('arabic', 'danish', 'dutch', 'english', 'finnish', 'french', 'german', 'hungaria
n', 'italian', 'norwegian', 'porter', 'portuguese', 'romanian', 'russian', 'spani
sh', 'swedish')
>>> f=sb('french')
>>> f.stem('dormez')
'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:

```
>>> import nltk
>>> from nltk.stem import PorterStemmer as ps
>>> s=ps()
>>> e="the cat was chasing the mouse"
>>> e=[s.stem(token) for token in e.split(" ")]
>>> print(" ".join(e))
the cat wa chase the mous
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> e="the cat was chasing the mouse. There were cacti around the corner"
>>> e=[lemmatizer.lemmatize(token) for token in e.split(" ")]
>>> print(" ".join(e))
the cat wa chasing the mouse. There were cactus around the corner
>>> print(lemmatizer.lemmatize('better', pos='a'))
good
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> vect=CountVectorizer(binary=True)
>>> corpus=["tessaract is an optical character recogniton engine","deuction skill
s are important for detective"]
>>> vect.fit(corpus)
CountVectorizer(analyzer='word', binary=True, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=True, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), preprocessor=None, stop_words=None,
        strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
        tokenizer=None, vocabulary=None)
>>> print(vect.transform(["Dectives are useful"]).toarray())
[[0 1 0 0 0 0 0 0 0 0 0 0 0]]
>>> vocab = vect.vocabulary_
>>> for key in sorted(vocab.key()):
        print("{}:{}".format(key,vocab[key]))
```

Similarity Check Between Articles To Decide Whether To Hire Or Not

```
import nltk
from nltk.corpus import stopwords
from nltk.corpus import inaugural
from nltk.stem.snowball import SnowballStemmer
import numpy as np

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 snowball:
        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(inaugural.raw('2009-Obama.txt'))
    sx=inaugural.fileids()
    for file in sx:
        s2=pre(inaugural.raw(file))
        #inter=set(s1) & set(s2)
        similarity1=similarity(s1,s2)
        print(similarity1,file)

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

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

```
In [3]: 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 "I

        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]]
```

```
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
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
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!
```