Application of Natural Language Processing:

```
- Sentiment analysis
- Chatbot
- Keyword search
- Info extraction (ex. from unstructured medical data)
- Spell check
- Speach recognition
- Ad match
- Machine trasnlation
```

In this notebook, I'll use:

- word_tokenize(text) after importing it from nltk.tokenize
- string.punctuation (import string first)
- stopwords.words('english') after downloading stopwords from nltk.corpus
- will also import Counter from collections
- will count most common and least common words using 1) collections' most_common() and 2) FreqDist from nltk.probability

At the end of the notebook, will leave reference to other useful language processing tools. These include:

- Stemming
- Name entity recognition NER (from nltk import ne-chunk
- Part of speach (POS) tagging using nltk.pos_tag([text])

In [1]:

```
import nltk
```

In [2]:

```
from nltk.corpus import movie_reviews
```

In [3]:

```
len(movie_reviews.fileids())
```

Out[3]:

```
2000
```

movie_reviews contain 2000 reviews in total, the first half rated negative (neg) and the other half rated positive (pos)

In [4]:

```
# here is a code to view the first five and last five side by side:

for tub in zip(movie_reviews.fileids()[:5], movie_reviews.fileids()[-5:]):
    print(tub)
```

```
('neg/cv000_29416.txt', 'pos/cv995_21821.txt')
('neg/cv001_19502.txt', 'pos/cv996_11592.txt')
('neg/cv002_17424.txt', 'pos/cv997_5046.txt')
('neg/cv003_12683.txt', 'pos/cv998_14111.txt')
('neg/cv004_12641.txt', 'pos/cv999_13106.txt')
```

We can use nltk.raw method to view one review as shown here:

In [5]:

```
print(movie_reviews.raw(movie_reviews.fileids()[1000]))
```

```
films adapted from comic books have had plenty of success , whether they're about superheroes ( batm
an , superman , spawn ) , or geared toward kids ( casper ) or the arthouse crowd ( ghost world ) , b
ut there's never really been a comic book like from hell before .
for starters , it was created by alan moore ( and eddie campbell ) , who brought the medium to a who
le new level in the mid '80s with a 12-part series called the watchmen .
to say moore and campbell thoroughly researched the subject of jack the ripper would be like saying
michael jackson is starting to look a little odd .
the book ( or " graphic novel , " if you will ) is over 500 pages long and includes nearly 30 more t
hat consist of nothing but footnotes .
in other words , don't dismiss this film because of its source .
if you can get past the whole comic book thing , you might find another stumbling block in from hell
's directors , albert and allen hughes .
getting the hughes brothers to direct this seems almost as ludicrous as casting carrot top in , well
, anything , but riddle me this : who better to direct a film that's set in the ghetto and features
really violent street crime than the mad geniuses behind menace ii society ?
the ghetto in question is , of course , whitechapel in 1888 london's east end .
it's a filthy , sooty place where the whores ( called " unfortunates " ) are starting to get a littl
e nervous about this mysterious psychopath who has been carving through their profession with surgic
al precision .
when the first stiff turns up , copper peter godley ( robbie coltrane , the world is not enough ) ca
lls in inspector frederick abberline ( johnny depp , blow ) to crack the case .
abberline , a widower , has prophetic dreams he unsuccessfully tries to quell with copious amounts o
f absinthe and opium .
upon arriving in whitechapel , he befriends an unfortunate named mary kelly ( heather graham , say i
t isn't so ) and proceeds to investigate the horribly gruesome crimes that even the police surgeon c
an't stomach .
i don't think anyone needs to be briefed on jack the ripper , so i won't go into the particulars her
e , other than to say moore and campbell have a unique and interesting theory about both the identit
y of the killer and the reasons he chooses to slay .
in the comic , they don't bother cloaking the identity of the ripper , but screenwriters terry hayes
( vertical limit ) and rafael yglesias ( les mis ? rables ) do a good job of keeping him hidden from
viewers until the very end .
it's funny to watch the locals blindly point the finger of blame at jews and indians because , after
all , an englishman could never be capable of committing such ghastly acts .
and from hell's ending had me whistling the stonecutters song from the simpsons for days ( " who hol
ds back the electric car/who made steve guttenberg a star ? " ) .
don't worry - it'll all make sense when you see it .
now onto from hell's appearance : it's certainly dark and bleak enough , and it's surprising to see
how much more it looks like a tim burton film than planet of the apes did ( at times , it seems like
sleepy hollow 2 ) .
the print i saw wasn't completely finished ( both color and music had not been finalized , so no com
ments about marilyn manson ) , but cinematographer peter deming ( don't say a word ) ably captures t
he dreariness of victorian-era london and helped make the flashy killing scenes remind me of the cra
zy flashbacks in twin peaks , even though the violence in the film pales in comparison to that in th
e black-and-white comic .
oscar winner martin childs' ( shakespeare in love ) production design turns the original prague surr
oundings into one creepy place .
even the acting in from hell is solid , with the dreamy depp turning in a typically strong performan
ce and deftly handling a british accent .
ians holm ( joe gould's secret ) and richardson ( 102 dalmatians ) log in great supporting roles , b
ut the big surprise here is graham .
i cringed the first time she opened her mouth , imagining her attempt at an irish accent , but it ac
tually wasn't half bad .
the film , however , is all good .
2 : 00 - r for strong violence/gore , sexuality , language and drug content
```

Here will use .words(text) to tokenize every word in the text. Will see that the list includes punctuations and stop words

In [6]:

```
# .words() is a method in nltk that tokenizes text
words = movie_reviews.words(movie_reviews.fileids()[0])
```

In [7]:

```
#let's view the first 15 words
words[0:15]
```

Out[7]:

```
['plot',
 ':',
 'two',
 'teen',
 'couples',
 'go',
 'to',
 'a',
 'church',
 'party',
 ',',
 'drink',
 'and',
 'then',
 'drive']
```

In [8]:

```
# It's easier to see with short text. So let's create one:
my_text = "I want to eat banana. Yes Banana! Banana is good. And I want to eat apple.Is eating banana and apple g
ood? Banana and apple are fruits. They are of course good"
```

In [9]:

```
# to tokenize the words, we need to import word_tokinize:
from nltk.tokenize import word_tokenize
```

In [10]:

```
my_words = word_tokenize(my_text)
```

In [11]:

```
from collections import Counter
```

In [12]:

```
num_words_my = Counter(my_words)
```

In [13]:

```
num_words_my['banana']
```

Out[13]:

```
2
```

In [14]:

```
# we can use most_common to return most common words:
num_words_my.most_common(4)
```

Out[14]:

```
[('.', 3), ('Banana', 3), ('good', 3), ('I', 2)]
```

Now let's go back to movie_reviews

Written text includes punctuations and stop words. Removing them makes language processing easier and more efficient. Here will use stopwords.words and string.punctuations to create a list of useless content.

In [15]:

```
# nltk compiled stopwords, which we can download. Remember to specify the language.

from nltk.corpus import stopwords
stopwords.words('english')[:5]
```

Out[15]:

```
['i', 'me', 'my', 'myself', 'we']
```

In [16]:

```
len(stopwords.words('english'))
```

Out[16]:

```
179
```

In [17]:

```
len(stopwords.words('arabic'))
```

Out[17]:

```
754
```

In [18]:

```
len(stopwords.words('french'))
```

Out[18]:

```
157
```

In [19]:

```
import string
```

In [20]:

```
string.punctuation
```

Out[20]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

In [21]:

```
len(string.punctuation)
```

Out[21]:

```
32
```

In [22]:

```
#now let's compile useless content in one list
useless_content = stopwords.words('english')+ list(string.punctuation)
```

In [23]:

```
def build_bag_of_words_features_filtered(words):
    """This function takes a text, filters out stop words and punctuations, and returns a dictionary
    of words contained in the text"""
    return {
        word:1 for word in words \
        if not word in useless_content}
```

In [24]:

```
all_words = movie_reviews.words()
```

In [25]:

```
len(all_words)/1e6
```

Out[25]:

```
1.58382
```

In [26]:

```
filtered_words = [word for word in movie_reviews.words() if not word in useless_content]
type(filtered_words)
```

Out[26]:

```
list
```

In [27]:

```
len(filtered_words)
```

Out[27]:

```
710579
```

In [28]:

```
# we can use Counter from collections to count the number or occurrence of each word
word_counter = Counter(filtered_words)
```

Using the collections .most_common() method, we can put the number of items as argument inside the paranteces as shown here:

In [29]:

```
word_counter.most_common(7)
```

Out[29]:

```
[('film', 9517),
 ('one', 5852),
 ('movie', 5771),
 ('like', 3690),
 ('even', 2565),
 ('good', 2411),
 ('time', 2411)]
```

or we can add indices at the end:

In [30]:

```
most_common_words = word_counter.most_common()[:5]
```

In [91]:

```
most_common_words
```

Out[91]:

```
[('film', 9517),
 ('one', 5852),
 ('movie', 5771),
 ('like', 3690),
 ('even', 2565)]
```

In [31]:

```
least_common_words=word_counter.most_common()[-5:]
```

In [32]:

```
least_common_words
```

Out[32]:

```
[('tangerine', 1),
 ('timbre', 1),
 ('powaqqatsi', 1),
 ('keyboardist', 1),
 ('capitalized', 1)]
```

In [33]:

```
type(most_common_words)
```

Out[33]:

```
list
```

The result of collection's most common word is a list of tuples. Which means we cannot check the occurance of a particular word. But we can do that using FreqDist from nltk.probability

In [34]:

```
from nltk.probability import FreqDist
```

In [35]:

```
fdist=FreqDist()
```

In [36]:

```
for word in filtered_words:
    fdist[word.lower()] +=1
    #return fdist
```

In [37]:

```
fdist
```

Out[37]:

```
FreqDist({'film': 9517, 'one': 5852, 'movie': 5771, 'like': 3690, 'even': 2565, 'good': 2411, 'time'
: 2411, 'story': 2169, 'would': 2109, 'much': 2049, ...})
```

In [38]:

```
fdist['love']
```

Out[38]:

```
1119
```

In [39]:

```
fdist['peace']
```

Out[39]:

```
34
```

In [40]:

```
fdist['again']
```

Out[40]:

```
0
```

In [41]:

```
# let's check the most_common words using FreqDist and compare it to most_common() from collections
#remember the argumen k inside .most_common(k) is the number of words you want the method to return
fdist.most_common(6)
```

Out[41]:

```
[('film', 9517),
 ('one', 5852),
 ('movie', 5771),
 ('like', 3690),
 ('even', 2565),
 ('good', 2411)]
```

## Sentiment Analysis Using Naive Bayes Classifier

Movie_reviews contain 1000 negatively rated reviews and 100 positively rated reviews. We can train Naive Bayes Classifier to classify a new review as positive or negative based on the most comon words

In [42]:

```
negative_reviews = movie_reviews.fileids('neg')
positive_reviews = movie_reviews.fileids('pos')
```

In [43]:

```
len(negative_reviews), len(positive_reviews)
```

Out[43]:

```
(1000, 1000)
```

In [45]:

```
# let's extract features from negative reviews.
negative_features = [
    (build_bag_of_words_features_filtered(movie_reviews.words(fileids=[f])), 'neg') \
    for f in negative_reviews
]
```

In [48]:

```
print(negative_features[5])
```

```
({'capsule': 1, '2176': 1, 'planet': 1, 'mars': 1, 'police': 1, 'taking': 1, 'custody': 1, 'accused'
: 1, 'murderer': 1, 'face': 1, 'title': 1, 'menace': 1, 'lot': 1, 'fighting': 1, 'whole': 1, 'story'
: 1, 'otherwise': 1, 'john': 1, 'carpenter': 1, 'reprises': 1, 'many': 1, 'ideas': 1, 'previous': 1,
'films': 1, 'especially': 1, 'assault': 1, 'precinct': 1, '13': 1, 'new': 1, 'film': 1, 'comes': 1,
'homage': 1, '0': 1, '4': 1, 'apparently': 1, 'believes': 1, 'action': 1, 'scenes': 1, 'people': 1,
'fight': 1, 'something': 1, 'horrible': 1, 'horror': 1, 'writer': 1, 'director': 1, 'supposedly': 1,
'expert': 1, 'bad': 1, 'mistake': 1, 'make': 1, 'ghosts': 1, 'called': 1, 'movie': 1, 'drawn': 1, 'h
umans': 1, 'surprisingly': 1, 'low': 1, 'powered': 1, 'alien': 1, 'addition': 1, 'anybody': 1, 'made
': 1, 'would': 1, 'grounds': 1, 'sue': 1, 'chock': 1, 'full': 1, 'pieces': 1, 'taken': 1, 'thing': 1
, 'prince': 1, 'darkness': 1, 'fact': 1, 'surprising': 1, 'managed': 1, 'fit': 1, 'work': 1, 'admitt
edly': 1, 'novel': 1, 'way': 1, 'still': 1, 'really': 1, 'good': 1, 'science': 1, 'fiction': 1, 'exp
erience': 1, 'takes': 1, 'place': 1, 'year': 1, 'mostly': 1, 'terraformed': 1, 'walk': 1, 'surface':
1, 'without': 1, 'breathing': 1, 'gear': 1, 'budget': 1, 'never': 1, 'mentioned': 1, 'gravity': 1, '
increased': 1, 'somehow': 1, 'earth': 1, 'normal': 1, 'making': 1, 'easier': 1, 'society': 1, 'chang
ed': 1, 'bit': 1, 'time': 1, 'advanced': 1, 'little': 1, 'culture': 1, 'women': 1, 'much': 1, 'posit
ions': 1, 'control': 1, 'view': 1, 'mess': 1, 'things': 1, 'stagnated': 1, 'female': 1, 'beyond': 1,
'minor': 1, 'technological': 1, 'advances': 1, 'less': 1, '175': 1, 'years': 1, 'might': 1, 'expect'
: 1, 'change': 1, 'ten': 1, 'basic': 1, 'plot': 1, 'common': 1, 'except': 1, '9': 1, 'yes': 1, 'repl
aced': 1, 'somewhat': 1, 'tacky': 1, 'looking': 1, 'rundown': 1, 'martian': 1, 'mining': 1, 'colony'
: 1, 'instead': 1, 'criminal': 1, 'napolean': 1, 'wilson': 1, 'desolation': 1, 'williams': 1, 'facin
g': 1, 'hoodlums': 1, 'automatic': 1, 'weapons': 1, 'well': 1, 'nature': 1, 'behave': 1, 'manner': 1
, 'essentially': 1, 'human': 1, 'savages': 1, 'another': 1, 'lapse': 1, 'imagination': 1, 'told': 1,
'flashback': 1, 'within': 1, 'entirely': 1, 'night': 1, 'filmed': 1, 'almost': 1, 'tones': 1, 'red':
1, 'yellow': 1, 'black': 1, 'manages': 1, 'give': 1, 'us': 1, 'powerful': 1, 'opening': 1, 'scene':
1, 'showing': 1, 'train': 1, 'rushing': 1, 'sound': 1, 'music': 1, 'heavy': 1, 'beat': 1, 'sadly': 1
, 'follows': 1, 'buildup': 1, 'terror': 1, 'creates': 1, 'looks': 1, 'like': 1, 'fugitive': 1, 'wann
abes': 1, 'rock': 1, 'band': 1, 'kiss': 1, 'idea': 1, 'building': 1, 'suspense': 1, 'bunch': 1, 'sud
den': 1, 'jump': 1, 'sucker': 1, 'viewer': 1, 'thinking': 1, 'scary': 1, 'happening': 1, 'prove': 1,
'boring': 1, 'standard': 1, 'haunted': 1, 'house': 1, 'shock': 1, 'effects': 1, 'require': 1, 'great
': 1, 'talent': 1, 'audience': 1, 'newer': 1, 'also': 1, 'unimpressive': 1, 'cgi': 1, 'digital': 1,
'decapitations': 1, 'fights': 1, 'short': 1, 'stretch': 1, 'seen': 1, 'release': 1, 'mission': 1, 'p
anned': 1, 'reviewers': 1, 'better': 1, 'goes': 1, 'rate': 1, '10': 1, 'scale': 1, 'following': 1, '
showed': 1, 'wife': 1, 'liked': 1, 'moderately': 1, 'classic': 1, 'comment': 1, 'seeing': 1, 'twice'
: 1}, 'neg')
```

In [50]:

```
#now the positive features
positive_features = [
    (build_bag_of_words_features_filtered(movie_reviews.words(fileids=[f])), 'pos') \
    for f in positive_reviews
]
```

In [51]:

```
print(positive_features[2])
```

```
({'got': 1, 'mail': 1, 'works': 1, 'alot': 1, 'better': 1, 'deserves': 1, 'order': 1, 'make': 1, 'fi
lm': 1, 'success': 1, 'cast': 1, 'two': 1, 'extremely': 1, 'popular': 1, 'attractive': 1, 'stars': 1
, 'share': 1, 'screen': 1, 'hours': 1, 'collect': 1, 'profits': 1, 'real': 1, 'acting': 1, 'involved
': 1, 'original': 1, 'inventive': 1, 'bone': 1, 'body': 1, 'basically': 1, 'complete': 1, 'shoot': 1
, 'shop': 1, 'around': 1, 'corner': 1, 'adding': 1, 'modern': 1, 'twists': 1, 'essentially': 1, 'goe
s': 1, 'defies': 1, 'concepts': 1, 'good': 1, 'contemporary': 1, 'filmmaking': 1, 'overly': 1, 'sent
imental': 1, 'times': 1, 'terribly': 1, 'mushy': 1, 'mention': 1, 'manipulative': 1, 'oh': 1, 'enjoy
able': 1, 'manipulation': 1, 'must': 1, 'something': 1, 'casting': 1, 'makes': 1, 'movie': 1, 'work'
: 1, 'well': 1, 'absolutely': 1, 'hated': 1, 'previous': 1, 'ryan': 1, 'hanks': 1, 'teaming': 1, 'sl
eepless': 1, 'seattle': 1, 'directing': 1, 'films': 1, 'helmed': 1, 'woman': 1, 'quite': 1, 'yet': 1
, 'figured': 1, 'liked': 1, 'much': 1, 'really': 1, 'important': 1, 'like': 1, 'even': 1, 'question'
: 1, 'storyline': 1, 'cliched': 1, 'come': 1, 'tom': 1, 'plays': 1, 'joe': 1, 'fox': 1, 'insanely':
1, 'likeable': 1, 'owner': 1, 'discount': 1, 'book': 1, 'chain': 1, 'meg': 1, 'kathleen': 1, 'kelley
': 1, 'proprietor': 1, 'family': 1, 'run': 1, 'children': 1, 'called': 1, 'nice': 1, 'homage': 1, 's
oon': 1, 'become': 1, 'bitter': 1, 'rivals': 1, 'new': 1, 'books': 1, 'store': 1, 'opening': 1, 'rig
ht': 1, 'across': 1, 'block': 1, 'small': 1, 'business': 1, 'little': 1, 'know': 1, 'already': 1, 'l
ove': 1, 'internet': 1, 'neither': 1, 'party': 1, 'knows': 1, 'person': 1, 'true': 1, 'identity': 1,
'rest': 1, 'story': 1, 'serve': 1, 'mere': 1, 'backdrop': 1, 'sure': 1, 'mildly': 1, 'interesting':
1, 'subplots': 1, 'fail': 1, 'comparison': 1, 'utter': 1, 'cuteness': 1, 'main': 1, 'relationship':
1, 'course': 1, 'leads': 1, 'predictable': 1, 'climax': 1, 'foreseeable': 1, 'ending': 1, 'damn': 1,
'cute': 1, 'done': 1, 'doubt': 1, 'entire': 1, 'year': 1, 'contains': 1, 'scene': 1, 'evokes': 1, 'p
ure': 1, 'joy': 1, 'part': 1, 'discovers': 1, 'online': 1, 'filled': 1, 'lack': 1, 'word': 1, 'happi
ness': 1, 'first': 1, 'time': 1, 'actually': 1, 'left': 1, 'theater': 1, 'smiling': 1}, 'pos')
```

In [52]:

```
from nltk.classify import NaiveBayesClassifier
```

In [53]:

```
split = 800
```

In [54]:

```
sentiment_classifier = NaiveBayesClassifier.train(positive_features[:split]+negative_features[:split])
```

In [55]:

```
nltk.classify.util.accuracy(sentiment_classifier, positive_features[split:]+negative_features[split:])*100
```

Out[55]:

```
71.75
```

In [56]:

```
sentiment_classifier.show_most_informative_features()
```

```
Most Informative Features
             outstanding = 1                 pos : neg    =     13.9 : 1.0
               insulting = 1                 neg : pos    =     13.7 : 1.0
              vulnerable = 1                 pos : neg    =     13.0 : 1.0
               ludicrous = 1                 neg : pos    =     12.6 : 1.0
             uninvolving = 1                 neg : pos    =     12.3 : 1.0
                  avoids = 1                 pos : neg    =     11.7 : 1.0
              astounding = 1                 pos : neg    =     11.7 : 1.0
             fascination = 1                 pos : neg    =     11.0 : 1.0
                  darker = 1                 pos : neg    =     10.3 : 1.0
                  symbol = 1                 pos : neg    =     10.3 : 1.0
```

In [ ]:

```

```

In [ ]:

```
# to get the number of paragraphs, import blankline_tokenize from nltk.tokenize. blank_text= blankline_tokenize(t
ext)
# blank_text(paragraph #) outputs the paragraph you asked for
```

In [ ]:

```
#bigram, trigram, ngram
```

In [ ]:

```

```

## Other useful methods in nltk include:

*Steming*

Normalize words into its base form or root form

In [ ]:

```
from nltk.stem import PorterStemmer
pst = PorterStemmer()
```

In [ ]:

```
pst.stem('having')
```

In [ ]:

```
from nltk.stem import LancasterStemmer
lst = LancasterStemmer()
```

In [ ]:

```
from nltk.stem import SnowballStemmer
sbst = SnowballStemmer('english')
```

In [ ]:

```
# Lemmantization takes into consideration morphological form of word
# similar to stemming but the output is always a word (unlike lancasterStemmer),
```

In [ ]:

```
from nltk.stem import wordnet
from nltk.stem import WordNetLemmatizer
word_len = WordNetLemmatizer()
```

In [ ]:

```
word_len.lemmatize('corpora')
```

Stop words 197

In [ ]:

```
import re
punctuation = re.compile(r'[-.?!,:;()[0-9]'])
```

In [ ]:

```
# parts of speach SOP

for word in text:
    print(nltk.pos_tag([text]))
```

In [ ]:

```
#NER named entity recognition (eq.movie, organization, monetary value, location, person, quantities )

from nltk import ne_chunk
```

In [ ]:

```
NE_text = 'The US President stays in the White House'
```

In [ ]:

```
NE_tokens = word_tokenize(NE_text)
NE_tags = nltk.pos_tag(NE_tokens)
```

In [ ]:

```
NE_NER = ne_chunk(NE_tags)
print(NE_NER)
```

In [1]:

```
#Chunking opposite of tokenizing
```

In [ ]:

```

```

In [ ]:

```
# based on python for data science week8 case study and tutorial from freeCodeCamp https://www.youtube.com/watch?v=X2vAabgKiuM
```