Movie Review project with nlp

In [2]:
```
import nltk
from nltk.corpus import movie_reviews, stopwords, wordnet
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize
import random
from nltk import pos_tag
```

In [103…
```
wordnet.NOUN # it return the single string that understand by wordnetlemmitazier
```

Out[103]:
```
'n'
```

In [104…
```
### this function take the tag of which returned by pos_tag function
```

In [105…
```
def get_simple_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

In [106…
```
movie_reviews.categories() # checking the categories of the movie_riviews datastet
```

Out[106]:
```
['neg', 'pos']
```

In [107…
```
data = movie_reviews.fileids() #it conaing 1000 pos and 1000 neg
len(data) #pos --> Positive, neg --> Negative
```

Out[107]:
```
2000
```

In [109…
```
print(len(movie_reviews.words(data[:1]))) # getting the words from the first docume
```

```
879
```

In [110…
```
# In cleaning process we need to remove the stopping words from documents
stop_words = stopwords.words('english') # nltk has inbuilt list that contain stop w
len(stop_words)
```

Out[110]:
```
179
```

In [111…
```
import string
stop_words += list(string.punctuation) # we have to add the punctuation to stop_wor
```

In [10]:
```
len(stop_words)
```

Out[10]:
```
211
```

In [12]:
```
movie_reviews.words(data[:1])
```

Out[12]:
```
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', ...]
```

In [13]:
```
## extracting words and categories into 1 file
def get_document_movie(shuffle=True): # here the suffle parameter act to shuffle th
    documents = [] # creating a empty list to store the words
    for cat in movie_reviews.categories(): # extracting the categories
        for doc in movie_reviews.fileids(cat): # based on category getting the txt
            documents.append((movie_reviews.words(doc), cat)) #append the tuple(wor
    if shuffle: # check the condition if it's true it suffle the documents else it
        random.shuffle(documents)
    return documents
```

In [113…
```
documents = get_document_movie() # this function return the all documents you can f
```

In [114…
```
documents[:2]
```

Out[114]:
```
[(['to', 'put', 'it', 'bluntly', ',', 'ed', 'wood', ...], 'neg'),
 (['the', 'question', 'isn', "'", 't', 'why', 'has', ...], 'pos')]
```

In [115…
```
pos_tag(['hari'])[0][1] # demonstation of getting part of speech tag from pos_tag m
```

Out[115]:
```
'NN'
```

In [116…
```
lemmit = WordNetLemmatizer() # creating a object of lemmitizer
```

In [117…
```
# this function clean the words and return the useful words based on pos
def clean_words(words): # words-> list of words
    output_words = [] # appending the all words after cleaning
    for w in words:
        if w.lower() not in stop_words: # converting the word into lower and checki
            pos = pos_tag([w])[0][1] # pos_tag([word])[tuple][tag]
            clean_word = lemmit.lemmatize(w, pos = get_simple_pos(pos)) # lemmatize
            output_words.append(clean_word.lower()) # after cleaning word appending
    return output_words  # returning the cleaned words list
```

In [118…
```
cleaned_document = [(clean_words(word), cat) for word, cat in documents]
```

In [20]:
```
print(cleaned_document[:1])
```

```
[(['dr', 'alan', 'grant', 'sam', 'neill', 'jurassic', 'park', 'become', 'disillusi
on', 'paleontology', 'longer', 'sexy', 'science', 'since', 'ingen', 'corporation',
'clone', 'subject', 'matter', 'lecture', 'bring', 'people', 'interested', 'adventu
re', 'isla', 'nubla', 'rather', 'research', 'funding', 'dollar', 'dry', 'kirbys',
'william', 'h', 'macy', 'fargo', 'tea', 'leoni', 'family', 'man', 'ask', 'guide',
'anniversary', 'flyover', 'isla', 'sorna', 'notorious', 'site', 'b', 'lose', 'worl
d', 'disdainful', 'wave', 'checkbook', 'reconsiders', 'however', 'kirbys', 'give',
'dr', 'grant', 'real', 'agenda', 'jurassic', 'park', 'iii', 'course', 'audience',
'tipped', 'give', 'film', 'begin', 'show', 'u', 'eric', 'trevor', 'morgan', 'patri
ot', 'young', 'boy', 'ben', 'mark', 'harelik', 'election', 'go', 'paragliding', 'a
dventure', 'island', 'go', 'awry', 'look', 'like', 'cheesy', 'rear', 'projection',
'grant', 'establish', 'back', 'home', 'new', 'right', 'hand', 'man', 'billy', 'bre
nnan', 'alessandro', 'nivola', 'love', 'labour', 'lose', 'site', 'dig', 'montana',
'sorely', 'lack', 'fund', 'also', 'pay', 'visit', 'old', 'flame', 'dr', 'ellie',
'sattler', 'laura', 'dern', 'jurassic', 'park', 'married', 'another', 'young', 'so
n', 'call', 'grant', 'dinosaur', 'man', 'apparently', 'sole', 'purpose', 'dredge',
'film', 'poorly', 'imagine', 'finale', 'grant', 'take', 'billy', 'along', 'kirby
s', 'trip', 'really', 'illegal', 'gambit', 'save', 'son', 'young', 'paraglider',
'couple', 'millionaire', 'make', 'grant', 'check', 'bogus', 'separate', 'well', 'e
ric', 'amanda', 'new', 'boyfriend', 'make', 'much', 'sense', 'meaning', 'gooey',
'family', 'dynamic', 'wait', 'dino', 'din', 'kirbys', 'hire', 'hand', 'obvious',
'bait', 'threesome', 'lead', 'mr', 'udesky', 'michael', 'jeter', 'gift', 'anyone',
'consider', 'cast', 'michael', 'jeter', 'william', 'h', 'macy', 'together', 'relat
ed', 'little', 'odd', 'direct', 'joe', 'johnston', 'october', 'sky', 'jumanji', 's
pielberg', 'produce', 'one', 'risible', 'script', 'peter', 'buchman', 'election',
'team', 'alexander', 'payne', 'jim', 'taylor', 'jurassic', 'park', 'iii', 'nothin
g', 'quickie', 'monster', 'flick', 'couple', 'new', 'dinos', 'spinosauraus', 'go',
'head', 'head', 'rex', 'pteranodons', 'plot', 'series', 'coincidence', 'combine',
'extreme', 'leap', 'faith', 'trifecta', 'stupid', 'cell', 'phone', 'trick', 'effec
t', 'longer', 'new', 'shot', 'television', 'cinematographer', 'shelly', 'johnson',
'rather', 'murky', 'look', 'time', 'film', 'edit', 'robert', 'dalva', 'october',
'sky', 'presumably', 'do', 'machete', 'keep', '90', 'minute', 'run', 'time', 'kno
w', 'reason', 'explain', 'ridiculous', 'end', 'feature', 'survivor', 'confront',
'pack', 'raptor', 'save', 'ludicrous', 'logic', 'jump', 'within', 'minute', 'origi
nal', 'music', 'davis', 'repeat', 'john', 'williams', 'original', 'theme', 'neil
l', 'young', 'morgan', 'attempt', 'inject', 'humor', 'humanity', 'proceeding', 're
st', 'cast', 'plod', 'unexceptional', 'jurassic', 'park', 'iii', 'probably', 'prov
ide', 'quick', 'entertainment', 'go', 'know', 'expect', 'crowd', 'maybe', 'like',
'lose', 'world', 'jurassic', 'park'], 'neg')]
```

In [119…

```
training_document = cleaned_document[:1500] # training and testing data for nltk cl
testing_document = cleaned_document[1500:]
# nltk need classifier of list that need to be tuple of words and cat
```

In [120…

```
all_words = []
for word in training_document:
    all_words += word[0] # extracting all words from training data
```

In [121…

```
freq = nltk.FreqDist(all_words) # using FreqDist() we can find the word frequency
print(freq.most_common(50))  # getting most frequent words
```

```
[('film', 8350), ('movie', 5181), ('one', 4479), ('make', 3269), ('like', 3014),
('character', 2884), ('get', 2810), ('see', 2349), ('go', 2245), ('time', 2234),
('well', 2214), ('even', 1986), ('scene', 1978), ('good', 1824), ('story', 1725),
('take', 1646), ('would', 1588), ('much', 1588), ('come', 1498), ('bad', 1479),
('also', 1456), ('life', 1446), ('give', 1443), ('two', 1429), ('look', 1413), ('k
now', 1395), ('way', 1387), ('end', 1385), ('seem', 1382), ('first', 1373), ('yea
r', 1321), ('--', 1314), ('work', 1310), ('say', 1232), ('thing', 1230), ('plot',
1202), ('really', 1188), ('little', 1163), ('play', 1159), ('people', 1152), ('sho
w', 1142), ('could', 1116), ('star', 1072), ('love', 1051), ('man', 1043), ('grea
t', 1006), ('big', 1000), ('never', 999), ('performance', 996), ('try', 995)]
```

In [122…
```
def freq_dict(document):
    freq_dict = []
    for words, cat in document:
        freq_dict.append((nltk.FreqDist(words), cat))
    return freq_dict
```

In [123…
```
def get_features(all_words, how_many=200):
    freq = nltk.FreqDist(all_words)
    most = freq.most_common(how_many)
    return [i[0] for i in most]
```

In [124…
```
features = get_features(all_words, how_many=300)
```

In [125…
```
print(features[:100])
```

```
['film', 'movie', 'one', 'make', 'like', 'character', 'get', 'see', 'go', 'time',
'well', 'even', 'scene', 'good', 'story', 'take', 'would', 'much', 'come', 'bad',
'also', 'life', 'give', 'two', 'look', 'know', 'way', 'end', 'seem', 'first', 'yea
r', '--', 'work', 'say', 'thing', 'plot', 'really', 'little', 'play', 'people', 's
how', 'could', 'star', 'love', 'man', 'great', 'big', 'never', 'performance', 'tr
y', 'director', 'best', 'want', 'new', 'many', 'actor', 'action', 'think', 'find',
'watch', 'u', 'role', 'act', 'another', 'still', 'audience', 'back', 'turn', 'some
thing', 'world', 'day', 'however', 'old', 'set', 'though', 'every', 'feel', 'guy',
'comedy', 'begin', 'use', 'real', 'enough', 'around', 'part', 'cast', 'last', 'poi
nt', 'may', 'interest', 'run', 'write', 'woman', 'young', 'name', 'actually', 'joh
n', 'lot', 'script', 'funny']
```

In [28]:
```
def get_features_dict(words):
    current_features = {}
    word_set = set(words)
    for w in features:
        current_features[w] = w in word_set
    return current_features
```

In [126…
```
training_data = [(get_features_dict(words), cat) for words, cat in training_documer
testing_data = [(get_features_dict(words), cat) for words, cat in testing_document]
```

In [127…
```
len(training_data), len(testing_data)
```

Out[127]:
```
(1500, 500)
```

classification using nltk naive bayes classifier

In [128…
```
from nltk import NaiveBayesClassifier
```

In [129…
```
random.shuffle(training_data)
```

In [130…
```
classifier = NaiveBayesClassifier.train(training_data)
```

In [131…
```
nltk.classify.accuracy(classifier, testing_data)
```

Out[131]:
```
0.758
```

Using sklear classifiers with nltk

In [132…
```
classifier.show_most_informative_features(15)
```

```
Most Informative Features
                  suppose = True              neg : pos    =      2.1 : 1.0
             unfortunately = True              neg : pos    =      1.9 : 1.0
                      war = True              pos : neg    =      1.9 : 1.0
                      bad = True              neg : pos    =      1.9 : 1.0
                      bad = False             pos : neg    =      1.8 : 1.0
                     true = True              pos : neg    =      1.8 : 1.0
                 american = True              pos : neg    =      1.7 : 1.0
                     joke = True              neg : pos    =      1.7 : 1.0
                especially = True              pos : neg    =      1.7 : 1.0
                    scream = True              neg : pos    =      1.7 : 1.0
                 different = True              pos : neg    =      1.6 : 1.0
                      save = True              neg : pos    =      1.6 : 1.0
                   present = True              pos : neg    =      1.6 : 1.0
                    minute = True              neg : pos    =      1.6 : 1.0
                       car = True              neg : pos    =      1.6 : 1.0
```

In [133…
```
from sklearn.svm import SVC
from nltk.classify.scikitlearn import SklearnClassifier
```

In [134…
```
svc = SVC(C=0.9)
classifier = SklearnClassifier(svc)
classifier.train(training_data)
nltk.classify.accuracy(classifier, testing_data)
```

Out[134]:
```
0.746
```

In [135…
```
train = {'hey this is hari krishna', 'this a data scientis krishna'}
from sklearn.feature_extraction.text import CountVectorizer
vec = CountVectorizer(max_features=3)
a = vec.fit_transform(train)
a.todense()
```

Out[135]:
```
matrix([[0, 1, 1],
        [1, 1, 1]])
```

In [136…
```
vec.get_feature_names_out()
```

Out[136]:
```
array(['data', 'krishna', 'this'], dtype=object)
```

In [137…
```
Y = [cat for word, cat in cleaned_document]
```

In [138…
```
X = [" ".join(word) for word, cat in cleaned_document]
```

In [139…
```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, random_state=0, test_size
```

In [140…
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [141…
```
vector = CountVectorizer()
X_train_vector = vector.fit_transform(X_train)
X_test_vector = vector.transform(X_test)
```

In [142…
```
vector.get_feature_names_out(X_test_vector)
```

Out[142]:
```
array(['00', '000', '007', ..., 'zwigoff', 'zycie', 'zzzzzzz'],
      dtype=object)
```

In [143…
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=225,bootstrap=False,  min_samples_leaf=7,
clf.fit(X_train_vector, Y_train)
score = clf.score(X_test_vector, Y_test)
score
```

Out[143]:
```
0.825
```

In [144…
```
Y_pred = clf.predict(X_test_vector)
```

In [145…
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [146…
```
print(confusion_matrix(Y_test, Y_pred))
```

```
[[247  35]
 [ 70 248]]
```

In [147…
```
print(classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

         neg       0.78      0.88      0.82       282
         pos       0.88      0.78      0.83       318

    accuracy                           0.82       600
   macro avg       0.83      0.83      0.82       600
weighted avg       0.83      0.82      0.83       600
```