sentiment_analysis with explanation

March 24, 2021

1 Sentiment Analysis with Python

1.1 Import and read data

[1]:
```
## Importing the necessary libraries along with the standard import

import numpy as np
import pandas as pd
import nltk # this is the Natural Language Tool Kit which contains a lot of␣
 ↪functionalities for text analytics
import random
import string # this is used for string manipulations

from nltk.corpus import movie_reviews

#nltk.download('stopwords')
#nltk.download('punkt')
#nltk.download('movie_reviews')
```

[2]:
```
len(movie_reviews.fileids()) # Checking the length/no. fileids
```

[2]:
```
2000
```

[3]:
```
movie_reviews.raw(movie_reviews.fileids()[0])
```

[3]:
```
'plot : two teen couples go to a church party , drink and then drive . \nthey
get into an accident . \none of the guys dies , but his girlfriend continues to
see him in her life , and has nightmares . \nwhat\'s the deal ? \nwatch the
movie and " sorta " find out . . . \ncritique : a mind-fuck movie for the teen
generation that touches on a very cool idea , but presents it in a very bad
package . \nwhich is what makes this review an even harder one to write , since
i generally applaud films which attempt to break the mold , mess with your head
and such ( lost highway & memento ) , but there are good and bad ways of making
all types of films , and these folks just didn\'t snag this one correctly .
\nthey seem to have taken this pretty neat concept , but executed it terribly .
\nso what are the problems with the movie ? \nwell , its main problem is that
it\'s simply too jumbled . \nit starts off " normal " but then downshifts into
this " fantasy " world in which you , as an audience member , have no idea
```

```
what\'s going on . \nthere are dreams , there are characters coming back from
the dead , there are others who look like the dead , there are strange
apparitions , there are disappearances , there are a looooot of chase scenes ,
there are tons of weird things that happen , and most of it is simply not
explained . \nnow i personally don\'t mind trying to unravel a film every now
and then , but when all it does is give me the same clue over and over again , i
get kind of fed up after a while , which is this film\'s biggest problem .
\nit\'s obviously got this big secret to hide , but it seems to want to hide it
completely until its final five minutes . \nand do they make things entertaining
, thrilling or even engaging , in the meantime ? \nnot really . \nthe sad part
is that the arrow and i both dig on flicks like this , so we actually figured
most of it out by the half-way point , so all of the strangeness after that did
start to make a little bit of sense , but it still didn\'t the make the film all
that more entertaining . \ni guess the bottom line with movies like this is that
you should always make sure that the audience is " into it " even before they
are given the secret password to enter your world of understanding . \ni mean ,
showing melissa sagemiller running away from visions for about 20 minutes
throughout the movie is just plain lazy ! ! \nokay , we get it . . . there \nare
people chasing her and we don\'t know who they are . \ndo we really need to see
it over and over again ? \nhow about giving us different scenes offering further
insight into all of the strangeness going down in the movie ? \napparently , the
studio took this film away from its director and chopped it up themselves , and
it shows . \nthere might\'ve been a pretty decent teen mind-fuck movie in here
somewhere , but i guess " the suits " decided that turning it into a music video
with little edge , would make more sense . \nthe actors are pretty good for the
most part , although wes bentley just seemed to be playing the exact same
character that he did in american beauty , only in a new neighborhood . \nbut my
biggest kudos go out to sagemiller , who holds her own throughout the entire
film , and actually has you feeling her character\'s unraveling . \noverall ,
the film doesn\'t stick because it doesn\'t entertain , it\'s confusing , it
rarely excites and it feels pretty redundant for most of its runtime , despite a
pretty cool ending and explanation to all of the craziness that came before it .
\noh , and by the way , this is not a horror or teen slasher flick . . . it\'s
\njust packaged to look that way because someone is apparently assuming that the
genre is still hot with the kids . \nit also wrapped production two years ago
and has been sitting on the shelves ever since . \nwhatever . . . skip \nit !
\nwhere\'s joblo coming from ? \na nightmare of elm street 3 ( 7/10 ) - blair
witch 2 ( 7/10 ) - the crow ( 9/10 ) - the crow : salvation ( 4/10 ) - lost
highway ( 10/10 ) - memento ( 10/10 ) - the others ( 9/10 ) - stir of echoes (
8/10 ) \n'
```

[4]:
```
nltk.FreqDist(movie_reviews.words()).most_common(10)
```

[4]:
```
[(',', 77717),
 ('the', 76529),
 ('.', 65876),
 ('a', 38106),
```

```
 ('and', 35576),
 ('of', 34123),
 ('to', 31937),
 ("'", 30585),
 ('is', 25195),
 ('in', 21822)]
```

[5]:
```
# Defining a variable 'stopwords' which contains the list of punctuations from␣
 ↪the string library and the english stopwords
# from nltk
stopwords = nltk.corpus.stopwords.words('english') +list(string.punctuation)

# Converting all the words to lower case
all_words = (w.lower() for w in movie_reviews.words())
# Only keeping the words which are not the 'stopwords'
all_words_clean = [word for word in all_words if word not in stopwords]


# Creating a frequency distribution of the lower case words which does not␣
 ↪contain any stopwords
all_words_freq = nltk.FreqDist(all_words_clean)

# Extracting the  most common 2000 words after the list of words have been␣
 ↪converted to lowercase and the stopwords
word_features = [item[0] for item in all_words_freq.most_common(2000)]
```

[6]:
```
word_features[0:15] # looking at first 5 word_features
```

[6]:
```
['film',
 'one',
 'movie',
 'like',
 'even',
 'good',
 'time',
 'story',
 'would',
 'much',
 'character',
 'also',
 'get',
 'two',
 'well']
```

[7]:
```
documents = [(list(movie_reviews.words(fileid)), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
```

```
# In the first line, we are creating a list where we need entries from both the␣
 ↪'category' and 'fileid'.
# A variable 'category' has been defined which will give output to all the␣
 ↪categories given by the following
# first loop "for category in movie_reviews.categories()".
# This particular value of the variable 'category' is then fitted into the␣
 ↪second loop "for fileid in movie_reviews.fileids(category)".
# So, the second loop is dependent on the first loop in the sense that it takes␣
 ↪the entries of the first loop and then executes it.
# In the end, the output of both these loops are stored in the list defined in␣
 ↪the first line.
random.shuffle(documents)
```

[8]:
```
documents[0][0][0:15] # Checking first 15
```

[8]:
```
['oh',
 'god',
 'how',
 'many',
 'john',
 'grisham',
 'lawyer',
 'films',
 'we',
 'have',
 'been',
 'munundated',
 'with',
 '!',
 'in']
```

[9]:
```
## We are defining a function to appropriately process the text document

def document_features(document): # we are naming the function as␣
 ↪document_features
    document_words = set(document) #getting the unique number of entries in the␣
 ↪document variable
    features = {} #defining an empty dictionary
    for word in word_features: #looping over the 'word_features' which has been␣
 ↪defined in the last code block
        features['contains({})'.format(word)] = (word in document_words)␣
 ↪#defining 'features' in  particular format
        # and checking whether the unique elements of the input 'document' are␣
 ↪contained in the 'word_features'
        # defined before
```

```
    return features
```

[10]:
```
## We are defining our combined data frame which we will split into training
 ↪and test before fitting a classifier

# We are creating a list the entries of which are a tuple. We are appending the
 ↪list with tuples whose entries are the
# pre-processed tweets and the corresponding sentiment attached to it.
featuresets = [(document_features(d), c) for (d,c) in documents]
```

[11]:
```
# Train Naive Bayes classifier
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

[12]:
```
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.81
```

[13]:
```
classifier.show_most_informative_features(10)
```

```
Most Informative Features
   contains(outstanding) = True              pos : neg    =     11.1 : 1.0
         contains(damon) = True              pos : neg    =      9.9 : 1.0
         contains(mulan) = True              pos : neg    =      8.4 : 1.0
   contains(wonderfully) = True              pos : neg    =      6.5 : 1.0
         contains(awful) = True              neg : pos    =      6.0 : 1.0
         contains(flynt) = True              pos : neg    =      5.7 : 1.0
          contains(lame) = True              neg : pos    =      5.6 : 1.0
        contains(wasted) = True              neg : pos    =      5.6 : 1.0
        contains(poorly) = True              neg : pos    =      5.5 : 1.0
         contains(waste) = True              neg : pos    =      5.2 : 1.0
```

[14]:
```
# A little difference in the results is due to random.shuffle as it randomly
 ↪shuffle the list
```