3 April 2018

Python NLTK: Sentiment Analysis on Movie Reviews [Natural Language Processing (NLP)]

This article shows how you can perform sentiment analysis on movie reviews using Python and Natural Language Toolkit (NLTK).

Sentiment Analysis means analyzing the sentiment of a given text or document and categorizing the text/document into a specific class or category (like positive and negative). In other words, we can say that sentiment analysis classifies any particular text or document as positive or negative. Basically, the classification is done for two classes: positive and negative. However, we can add more classes like neutral, highly positive, highly negative, etc.

Sentiment Analysis is also referred as Opinion Mining. It's mostly used in social media and customer reviews data.

In this article, we will learn about labeling data, extracting features, training classifier, and testing the accuracy of the classifier.

Supervised Classification

Here, we will be doing supervised text classification. In supervised classification, the classifier is trained with **labeled training data**.

In this article, we will use the NLTK's `movie_reviews` corpus as our labeled training data. The `movie_reviews` corpus contains 2K movie reviews with sentiment polarity classification. It's compiled by Pang, Lee.

Here, we have two categories for classification. They are: positive and negative. The movie_reviews corpus already has the reviews categorized as positive and negative.

```python
from nltk.corpus import movie_reviews

# Total reviews
print (len(movie_reviews.fileids())) # Output: 2000

# Review categories
print (movie_reviews.categories()) # Output: [u'neg', u'pos']

# Total positive reviews
print (len(movie_reviews.fileids('pos'))) # Output: 1000

# Total negative reviews
print (len(movie_reviews.fileids('neg'))) # Output: 1000

positive_review_file = movie_reviews.fileids('pos')[0]
print (positive_review_file) # Output: pos/cv000_29590.txt
```

Create list of movie review document

This list contains array containing tuples of all movie review words and their respective category (pos or neg).

```python
documents = []

for category in movie_reviews.categories():
    for fileid in movie_reviews.fileids(category):
        #documents.append((list(movie_reviews.words(fileid)), category))
        documents.append((movie_reviews.words(fileid), category))

print (len(documents)) # Output: 2000

# x = [str(item) for item in documents[0][0]]
# print (x)
```

About

Mukesh Chapagain is a graduate of Kathmandu University (Dhulikhel, Nepal) from where he holds a Masters degree in Computer Engineering. Mukesh is a passionate web developer who has keen interest in open source technologies, programming & blogging. more...

Subscribe via Email

Email: Subscribe

Categories

Select Category

Archives

Select Month

Recent Posts

- Magento 2: Create Widget Programmatically & Assign Static Block to it
- Magento 2: Add/Update CMS Static Block via Install/Upgrade Script Programmatically
- Magento 2: Add/Update CMS Page via Install/Upgrade Script Programmatically
- FFMPEG: Convert & Edit Video via Command Line
- ImageMagick: Convert/Edit Multiple Images
- [INFOGRAPHIC] Magento Basic Facts for Newbies
- ImageMagick: Convert & Edit Image via Command Line
- [SOLVED] ERROR 2006 (HY000): MySQL server has gone away
- [SOLVED] MySQL: The server quit without updating PID file
- Magento 2: Change Increment ID Prefix, Suffix, Start value, Step, Pad length of Order, Invoice, Creditmemo & Shipment

Related Posts

- Python NLTK: Twitter Sentiment Analysis [Natural Language Processing (NLP)]
- Python NLTK: Text Classification [Natural Language Processing (NLP)]
- Natural Language Processing (NLP): Basic Introduction to NLTK [Python]
- Python NLTK: Stop Words [Natural Language Processing (NLP)]
- Python NLTK: Stemming & Lemmatization [Natural Language Processing (NLP)]
- Python NLTK: Working with WordNet [Natural Language Processing (NLP)]
- Python NLTK: Part-of-Speech (POS) Tagging [Natural Language Processing (NLP)]
- Machine Learning & Sentiment Analysis: Text Classification using Python & NLTK
- Python: Twitter Sentiment Analysis on Real Time Tweets using TextBlob
- Python: Twitter Sentiment Analysis using TextBlob

Most Viewed

- How to Calculate Inverter & Battery Backup Time?
- Very Simple Add, Edit, Delete, View (CRUD) in PHP & MySQL [Beginner Tutorial]
- Magento: How to get attribute name and value?
- CodeIgniter: Simple Add, Edit, Delete, View – MVC CRUD Application
- Magento: How to get controller, module, action and router name?
- Magento: How to select, insert, update, and dele
- LaTeX: Generate dummy text (lorem ipsum) in y document
- CRUD with Login & Register in PHP & MySQL (Delete, View)

```
# print first tuple
print (documents[0])
'''
Output:

(['plot', ':', 'two', 'teen', 'couples', 'go', ...], 'neg')
'''

# shuffle the document list
from random import shuffle
shuffle(documents)
```

- PHP: CRUD (Add, Edit, Delete, View) Application using OOP (Object Oriented Programming)
- Magento Admin login problem

Feature Extraction

To classify the text into any category, we need to define some criteria. On the basis of those criteria, our classifier will learn that a particular kind of text falls in a particular category. This kind of criteria is known as `feature`. We can define one or more feature to train our classifier.

In this example, we will use the `top-N words feature`.

Fetch all words from the movie reviews corpus

We first fetch all the words from all the movie reviews and create a list.

```
all_words = [word.lower() for word in movie_reviews.words()]

# print first 10 words
print (all_words[:10])
'''
Output:

['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church', 'party']
'''
```

Create Frequency Distribution of all words

Frequency Distribution will calculate the number of occurence of each word in the entire list of words.

```
from nltk import FreqDist

all_words_frequency = FreqDist(all_words)

print (all_words_frequency)
'''
Output:

<FreqDist with 39768 samples and 1583820 outcomes>
'''

# print 10 most frequently occurring words
print (all_words_frequency.most_common(10))
'''
Output:

[(',', 77717), ('the', 76529), ('.', 65876), ('a', 38106), ('and', 35576), ('of', 34123), ('to', 31937), ("'", 30585), ('is', 25195), ('in', 21822)]
'''
```

Removing Punctuation and Stopwords

From the above frequency distribution of words, we can see the most frequently occurring words are either punctuation marks or stopwords.

Stop words are those frequently words which do not carry any significant meaning in text analysis. For example, I, me, my, the, a, and, is, are, he, she, we, etc.

Punctuation marks like comma, fullstop. inverted comma, etc. occur highly in any text data.

We will do `data cleaning` by removing stop words and punctuations.

Remove Stop Words

```
from nltk.corpus import stopwords

stopwords_english = stopwords.words('english')
print (stopwords_english)
'''
Output:

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'he
```

```
r', 'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', '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', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn', 'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan', 'shouldn', 'wasn', 'weren', 'won', 'wouldn']
'''

# create a new list of words by removing stopwords from all_words
all_words_without_stopwords = [word for word in all_words if word not in stopwords_english]

# print the first 10 words
print (all_words_without_stopwords[:10])
'''
Output:

['plot', ':', 'two', 'teen', 'couples', 'go', 'church', 'party', ',', 'drink']
'''

'''
# Above code is written using the List Comprehension feature of Python
# It's the same thing as writing the following, the output is the same

all_words_without_stopwords = []
for word in all_words:
    if word not in stopwords_english:
        all_words_without_stopwords.append(word)

print (all_words_without_stopwords[:10])
'''
```

You can see that after removing stopwords, the words `to` and `a` has been removed from the first 10 words result.

Remove Punctuation

```
import string

print (string.punctuation)
'''
Output:

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

# create a new list of words by removing punctuation from all_words
all_words_without_punctuation = [word for word in all_words if word not in string.punctuation]

# print the first 10 words
print (all_words_without_punctuation[:10])
'''
Output:

[u'plot', u'two', u'teen', u'couples', u'go', u'to', u'a', u'church', u'party', u'drink']
'''
```

You can see that on the list that all punctuations like semi-colon `:`, comma `,` are removed.

Remove both Stopwords & Punctuation

In the above examples, at first, we only removed stopwords and then in the next code, we only removed punctuation.

Below, we will remove both stopwords and punctuation from the `all_words` list.

```
# Let's name the new list as all_words_clean
# because we clean stopwords and punctuations from the word list

all_words_clean = []
for word in all_words:
    if word not in stopwords_english and word not in string.punctuation:
        all_words_clean.append(word)

print (all_words_clean[:10])
```

```
'''
Output:

['plot', 'two', 'teen', 'couples', 'go', 'church', 'party', 'drink', 'drive', 'get']
'''
```

Frequency Distribution of cleaned words list

Below is the frequency distribution of the new list after removing stopwords and punctuation.

```
all_words_frequency = FreqDist(all_words_clean)

print (all_words_frequency)
'''
Output:

<FreqDist with 39586 samples and 710578 outcomes>
'''

# print 10 most frequently occurring words
print (all_words_frequency.most_common(10))
'''
Output:

[('film', 9517), ('one', 5852), ('movie', 5771), ('like', 3690), ('even', 2565), ('time', 2411), ('good', 2411), ('story', 2169), ('would', 2109), ('much', 2049)]
'''
```

Previously, before removing stopwords and punctuation, the frequency distribution was:

`FreqDist with 39768 samples and 1583820 outcomes`

Now, the frequency distribution is:

`FreqDist with 39586 samples and 710578 outcomes`

This shows that after removing around 200 stop words and punctuation, the outcomes/words number has reduced to around half of the original size.

The `most common words` or highly occurring words list has also got meaningful words in the list. Before, the first 10 frequently occurring words were only stop-words and punctuations.

Create Word Feature using 2000 most frequently occurring words

We take 2000 most frequently occurring words as our feature.

```
print (len(all_words_frequency)) # Output: 39586

# get 2000 frequently occuring words
most_common_words = all_words_frequency.most_common(2000)
print (most_common_words[:10])
'''
Output:

[('film', 9517), ('one', 5852), ('movie', 5771), ('like', 3690), ('even', 2565), ('time', 2411), ('good', 2411), ('story', 2169), ('would', 2109), ('much', 2049)]
'''

print (most_common_words[1990:])
'''
Output:

[('genuinely', 64), ('path', 64), ('eve', 64), ('aware', 64), ('bank', 64), ('bound', 64), ('eric', 64), ('regular', 64), ('las', 64), ('niro', 64)]
'''

# the most common words list's elements are in the form of tuple
# get only the first element of each tuple of the word list
word_features = [item[0] for item in most_common_words]
print (word_features[:10])
'''
Output:

['film', 'one', 'movie', 'like', 'even', 'time', 'good', 'story', 'would', 'much']
'''
```

Create Feature Set

Now, we write a function that will be used to create feature set. The feature set is used to train the classifier.

We define a feature extractor function that checks if the words in a given document are present in the word_features list or not.

```
def document_features(document):
    # "set" function will remove repeated/duplicate tokens in the given list
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains(%s)' % word] = (word in document_words)
    return features

# get the first negative movie review file
movie_review_file = movie_reviews.fileids('neg')[0]
print (movie_review_file)
'''
Output:

neg/cv000_29416.txt
'''

#print (document_features(movie_reviews.words(movie_review_file)))
'''
Output:

{'contains(waste)': False, 'contains(lot)': False, 'contains(rent)': False, 'contains(black)': False, 'contains(rated)': False, 'contains(potential)': False,

............................................................................................................................... 'contains(smile)': False, 'contains(cross)': False, 'contains(barry)': False}
'''
```

In the beginning of this article, we have created the `documents` list which contains data of all the movie reviews. Its elements are tuples with word list as first item and review category as the second item of the tuple.

```
# print first tuple of the documents list
print (documents[0])
'''
Output:

(['plot', ':', 'two', 'teen', 'couples', 'go', ...], 'neg')
'''
```

We now loop through the `documents` list and create a feature set list using the `document_features` function defined above.

– Each item of the feature_set list is a tuple.
– The first item of the tuple is the dictionary returned from `document_features` function
– The second item of the tuple is the category (pos or neg) of the movie review

```
feature_set = [(document_features(doc), category) for (doc, category) in documents]
print (feature_set[0])
'''
Output:

({'contains(waste)': False, 'contains(lot)': False, 'contains(rent)': False, 'contains(black)': False, 'contains(rated)': False, 'contains(potential)': False,
 ...........................................................................
............................................................. 'contains(good)': False, 'contains(live)': False, 'contains(synopsis)': False, 'contains(appropriate)': False, 'contains(towards)': False, 'contains(smile)': False, 'contains(cross)': False, 'contains(barry)': False}, 'neg')
'''

'''
# In the above code, we have used list-comprehension feature of python
# The same code can be written as below:
feature_set = []
for (doc, category) in documents:
    feature_set.append((document_features(doc), category))
print (feature_set[0])
'''
```

Training Classifier

From the feature set we created above, we now create a separate training set and a separate testing/validation set. The train set is used to train the classifier and the test set is used to test the classifier to check how accurately it classifies the given text.

Creating Train and Test Dataset

In this example, we use the first 400 elements of the feature set array as a test set and the rest of the data as a train set. Generally, 80/20 percent is a fair split between training and testing set, i.e. 80 percent training set and 20 percent testing set.

```
print (len(feature_set)) # Output: 2000

test_set = feature_set[:400]
train_set = feature_set[400:]

print (len(train_set)) # Output: 1600
print (len(test_set)) # Output: 400
```

Training a Classifier

Now, we train a classifier using the training dataset. There are different kind of classifiers namely Naive Bayes Classifier, Maximum Entropy Classifier, Decision Tree Classifier, Support Vector Machine Classifier, etc.

In this example, we use the Naive Bayes Classifier. It's a simple, fast, and easy classifier which performs well for small datasets. It's a simple probabilistic classifier based on applying Bayes' theorem. Bayes' theorem describes the probability of an event, based on prior knowledge of conditions that might be related to the event.

```
from nltk import NaiveBayesClassifier

classifier = NaiveBayesClassifier.train(train_set)
```

Testing the trained Classifier

Let's see the accuracy percentage of the trained classifier. The accuracy value changes each time you run the program because of the names array being shuffled above.

```
from nltk import classify

accuracy = classify.accuracy(classifier, test_set)
print (accuracy) # Output: 0.77
```

Let's see the output of the classifier by providing some custom reviews.

```
from nltk.tokenize import word_tokenize

custom_review = "I hated the film. It was a disaster. Poor direction, bad acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = document_features(custom_review_tokens)
print (classifier.classify(custom_review_set)) # Output: neg
# Negative review correctly classified as negative

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: neg
print (prob_result.prob("neg")) # Output: 0.999989264571
print (prob_result.prob("pos")) # Output: 1.07354285262e-05

custom_review = "It was a wonderful and amazing movie. I loved it. Best direction, good acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = document_features(custom_review_tokens)

print (classifier.classify(custom_review_set)) # Output: neg
# Positive review is classified as negative
# We need to improve our feature set for more accurate prediction

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: neg
print (prob_result.prob("neg")) # Output: 0.999791868552
print (prob_result.prob("pos")) # Output: 0.000208131447797
```

Let's see the most informative features among the entire features in the feature set.

```
# show 5 most informative features
print (classifier.show_most_informative_features(10))
'''
Output:

Most Informative Features
   contains(outstanding) = True              pos : neg    =     14.7 : 1.0
         contains(mulan) = True              pos : neg    =      7.8 : 1.0
        contains(poorly) = True              neg : pos    =      7.7 : 1.0
    contains(wonderfully) = True             pos : neg    =      7.5 : 1.0
        contains(seagal) = True              neg : pos    =      6.5 : 1.0
         contains(awful) = True              neg : pos    =      6.1 : 1.0
        contains(wasted) = True              neg : pos    =      6.1 : 1.0
         contains(waste) = True              neg : pos    =      5.6 : 1.0
         contains(damon) = True              pos : neg    =      5.3 : 1.0
         contains(flynt) = True              pos : neg    =      5.1 : 1.0
'''
```

The result shows that the word `outstanding` is used in positive reviews 14.7 times more often than it is used in negative reviews the word `poorly` is used in negative reviews 7.7 times more often than it is used in positive reviews. Similarly, for other letters. These ratios are also called **`likelihood ratios`**.

Therefore, a review has a high chance to be classified as positive if it contains words like `outstanding` and `wonderfully`. Similarly, a review has a high chance of being classified as negative if it contains words like `poorly`, `awful`, `waste`, etc.

> **Note:** You can modify the `document_features` function to generate the feature set which can improve the accuracy of the trained classifier. Feature extractors are built through a process of trail-and-error & guided by intuitions.

Bag of Words Feature

In the above example, we used **`top-N words`** feature. We used 2000 most frequently occurring words as our top-N words feature. The classifier identified negative review as negative. However, the classifier was not able to classify positive review correctly. It classified a positive review as negative.

> **Top-N words feature**
>
> – The top-N words feature is also a bag-of-words feature.
> – But in the top-N feature, we only used the top 2000 words in the feature set.
> – We combined the positive and negative reviews into a single list, randomized the list, and then separated the train and test set.
> – This approach can result in the un-even distribution of positive and negative reviews across the train and test set.
>
> **Bag-of-words feature shown below**
>
> In the bag-of-words feature as shown below:
>
> – We will use all the useful words of each review while creating the feature set.
> – We take a fixed number of positive and negative reviews for train and test set.
> – This result in equal distribution of positive and negative reviews across train and test set.

In the approach shown below, we will modify the feature extractor function.

– We form a list of unique words of each review.
– The category (pos or neg) is assigned to each bag of words.
– Then the category of any given text is calculated by matching the different bag-of-words & their respective category.

```
from nltk.corpus import movie_reviews

pos_reviews = []
for fileid in movie_reviews.fileids('pos'):
    words = movie_reviews.words(fileid)
    pos_reviews.append(words)

neg_reviews = []
for fileid in movie_reviews.fileids('neg'):
    words = movie_reviews.words(fileid)
    neg_reviews.append(words)

# print first positive review item from the pos_reviews list
print (pos_reviews[0])
'''
Output:
```

```

['films', 'adapted', 'from', 'comic', 'books', ...]
'''

# print first negative review item from the neg_reviews list
print (neg_reviews[0])
'''
Output:

['plot', ':', 'two', 'teen', 'couples', 'go', ...]
'''

# print first 20 items of the first item of positive review
print (pos_reviews[0][:20])
'''
Output:

['films', 'adapted', 'from', 'comic', 'books', 'have', 'had', 'plenty', 'of', 'success', ',', 'whether', 'they', "'", 're', 'about', 'superheroes', '(', 'batman', ',']
'''

# print first 20 items of the first item of negative review
print (neg_reviews[0][:20])
'''
Output:

['plot', ':', 'two', 'teen', 'couples', 'go', 'to', 'a', 'church', 'party', ',', 'drink', 'and', 'then', 'drive', '.', 'they', 'get', 'into', 'an']
'''
```

Feature Extraction

We use the bag-of-words feature. Here, we clean the word list (i.e. remove stop words and punctuation). Then, we create a dictionary of cleaned words.

```
from nltk.corpus import stopwords
import string

stopwords_english = stopwords.words('english')

# feature extractor function
def bag_of_words(words):
    words_clean = []

    for word in words:
        word = word.lower()
        if word not in stopwords_english and word not in string.punctuation:
            words_clean.append(word)

    words_dictionary = dict([word, True] for word in words_clean)

    return words_dictionary

# using dict will remove duplicate words from the words list
# note the output: stopword 'the' is also removed
print (bag_of_words(['the', 'the', 'good', 'bad', 'the', 'good']))
'''
Output:

{'bad': True, 'good': True}
'''
```

Create Feature Set

We use the bag-of-words feature and tag each review with its respective category as positive or negative.

```
# positive reviews feature set
pos_reviews_set = []
for words in pos_reviews:
    pos_reviews_set.append((bag_of_words(words), 'pos'))

# negative reviews feature set
neg_reviews_set = []
for words in neg_reviews:
    neg_reviews_set.append((bag_of_words(words), 'neg'))

# print first positive review item from the pos_reviews list
print (pos_reviews_set[0])
'''
Output:

({'childs': True, 'steve': True, 'surgical': True, 'go': True, 'certainly': Tr
True, 'song': True, 'simpsons': True, 'novel': True,
 ........................................................................

```

```
........................................................ 'menace': True, 'stai
riginal': True}, 'pos')
'''

# print first negative review item from the neg_reviews list
print (neg_reviews_set[0])
'''
Output:

({'concept': True, 'skip': True, 'insight': True, 'playing': True, 'executed'
ue, 'still': True, 'find': True, 'seemed': True,

..............................................................................
................................................. 'entertaining': True, 'years
y': True, 'came': True}, 'neg')
'''
```

Create Train and Test Set

There are 1000 positive reviews set and 1000 negative reviews set. We take 20% (i.e. 200) of positive reviews and 20% (i.e. 200) of negative reviews as a test set. The remaining negative and positive reviews will be taken as a training set.

> **Note:**
> – There is difference between pos_reviews & pos_reviews_set array which are defined above.
> – pos_reviews array contains words list only
> – pos_reviews_set array contains words feature list
> – pos_reviews_set & neg_reviews_set arrays are used to create train and test set as shown below

```
print (len(pos_reviews_set), len(neg_reviews_set)) # Output: (1000, 1000)

# radomize pos_reviews_set and neg_reviews_set
# doing so will output different accuracy result everytime we run the progra
m
from random import shuffle
shuffle(pos_reviews_set)
shuffle(neg_reviews_set)

test_set = pos_reviews_set[:200] + neg_reviews_set[:200]
train_set = pos_reviews_set[200:] + neg_reviews_set[200:]

print(len(test_set),  len(train_set)) # Output: (400, 1600)
```

Training Classifier and Calculating Accuracy

We train Naive Bayes Classifier using the training set and calculate the classification accuracy of the trained classifier using the test set.

```
from nltk import classify
from nltk import NaiveBayesClassifier

classifier = NaiveBayesClassifier.train(train_set)

accuracy = classify.accuracy(classifier, test_set)
print(accuracy) # Output: 0.7325

print (classifier.show_most_informative_features(10))
'''
Output:

Most Informative Features
            breathtaking = True              pos : neg    =     20.3 : 1.0
                dazzling = True              pos : neg    =     12.3 : 1.0
               ludicrous = True              neg : pos    =     12.2 : 1.0
             outstanding = True              pos : neg    =     10.6 : 1.0
                 insipid = True              neg : pos    =     10.3 : 1.0
               stretched = True              neg : pos    =     10.3 : 1.0
               stupidity = True              neg : pos    =     10.2 : 1.0
                  annual = True              pos : neg    =      9.7 : 1.0
                headache = True              neg : pos    =      9.7 : 1.0
                  avoids = True              pos : neg    =      9.7 : 1.0
'''
```

Testing Classifier with Custom Review

We provide custom review text and check the classification output of the trained classifier. The classifier correctly predicts both negative and positive reviews provided.

```
from nltk.tokenize import word_tokenize


```

```
custom_review = "I hated the film. It was a disaster. Poor direction, bad acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = bag_of_words(custom_review_tokens)
print (classifier.classify(custom_review_set)) # Output: neg
# Negative review correctly classified as negative

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: neg
print (prob_result.prob("neg")) # Output: 0.776128854994
print (prob_result.prob("pos")) # Output: 0.223871145006

custom_review = "It was a wonderful and amazing movie. I loved it. Best direction, good acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = bag_of_words(custom_review_tokens)

print (classifier.classify(custom_review_set)) # Output: pos
# Positive review correctly classified as positive

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: pos
print (prob_result.prob("neg")) # Output: 0.0972171562901
print (prob_result.prob("pos")) # Output: 0.90278284371
```

Bi-gram Features

N-grams are common terms in text processing and analysis. N-grams are related with words of a text. There are different n-grams like unigram, bigram, trigram, etc.

> Unigram = Item having a single word, i.e. the n-gram of size 1. For example, good.
> Bigram = Item having two words, i.e. the n-gram of size 2. For example, very good.
> Trigram = Item having three words, i.e. the n-gram of size 3. For example, not so good.

In the above bag-of-words model, we only used the unigram feature. In the example below, we will use both unigram and bigram feature, i.e. we will deal with both single words and double words.

Feature Extraction

In this case, both unigrams and bigrams are used as features.

We define two functions:

– `bag_of_words`: that extracts only unigram features from the movie review words
– `bag_of_ngrams`: that extracts only bigram features from the movie review words

We then define another function:

– `bag_of_all_words`: that combines both unigram and bigram features

```
from nltk import ngrams
from nltk.corpus import stopwords
import string

stopwords_english = stopwords.words('english')

# clean words, i.e. remove stopwords and punctuation
def clean_words(words, stopwords_english):
    words_clean = []
    for word in words:
        word = word.lower()
        if word not in stopwords_english and word not in string.punctuation:
            words_clean.append(word)
    return words_clean

# feature extractor function for unigram
def bag_of_words(words):
    words_dictionary = dict([word, True] for word in words)
    return words_dictionary

# feature extractor function for ngrams (bigram)
def bag_of_ngrams(words, n=2):
    words_ng = []
    for item in iter(ngrams(words, n)):
        words_ng.append(item)
    words_dictionary = dict([word, True] for word in words_ng)
    return words_dictionary

'''
```

Privacy - Terms

```
# Alternative Bi-gram feature extractor
# using BigramCollocationFinder module

# Collocations are multiple words which commonly co-occur.
# http://www.nltk.org/howto/collocations.html
# https://streamhacker.com/2010/05/24/text-classification-sentiment-analysis-stopwords-collocations/

import itertools
from nltk.collocations import BigramCollocationFinder
from nltk.metrics import BigramAssocMeasures

# feature extractor function for ngrams (bigram)
# get 200 most frequently occurring bigrams from every review
def bag_of_ngrams(words, score_fn=BigramAssocMeasures.chi_sq, n=200):
    bigram_finder = BigramCollocationFinder.from_words(words)
    bigrams = bigram_finder.nbest(score_fn, n)
    return dict([(ngram, True) for ngram in itertools.chain(words, bigrams)])
'''

from nltk.tokenize import word_tokenize
text = "It was a very good movie."
words = word_tokenize(text.lower())

print (words)
'''
Output:

['it', 'was', 'a', 'very', 'good', 'movie', '.']
'''

print (bag_of_ngrams(words))
'''
Output:

{('very', 'good'): True, ('movie', '.'): True, ('it', 'was'): True, ('good', 'movie'): True, ('was', 'a'): True, ('a', 'very'): True}
'''

# working with cleaning words
# i.e. removing stopwords and punctuation
words_clean = clean_words(words, stopwords_english)
print (words_clean)
'''
Output:

['good', 'movie']
'''

# cleaning words is find for unigrams
# but this can omit important words for bigrams
# for example, stopwords like very, over, under, so, etc. are important for bigrams
# we create a new stopwords list specifically for bigrams by omitting such important words
important_words = ['above', 'below', 'off', 'over', 'under', 'more', 'most', 'such', 'no', 'nor', 'not', 'only', 'so', 'than', 'too', 'very', 'just', 'but']

stopwords_english_for_bigrams = set(stopwords_english) - set(important_words)

words_clean_for_bigrams = clean_words(words, stopwords_english_for_bigrams)
print (words_clean_for_bigrams)
'''
Output:

['very', 'good', 'movie']
'''

# We will use general stopwords for unigrams
# And special stopwords list for bigrams
unigram_features = bag_of_words(words_clean)
print (unigram_features)
'''
Output:

{'movie': True, 'good': True}
'''

bigram_features = bag_of_ngrams(words_clean_for_bigrams)
print (bigram_features)
'''
Output:

{('very', 'good'): True, ('good', 'movie'): True}
'''

# combine both unigram and bigram features
all_features = unigram_features.copy()
all_features.update(bigram_features)
print (all_features)
```

```
'''
Output:

{'movie': True, ('very', 'good'): True, 'good': True, ('good', 'movie'): True}
'''

# let's define a new function that extracts all features
# i.e. that extracts both unigram and bigrams features
def bag_of_all_words(words, n=2):
    words_clean = clean_words(words, stopwords_english)
    words_clean_for_bigrams = clean_words(words, stopwords_english_for_bigrams)

    unigram_features = bag_of_words(words_clean)
    bigram_features = bag_of_ngrams(words_clean_for_bigrams)

    all_features = unigram_features.copy()
    all_features.update(bigram_features)

    return all_features

print (bag_of_all_words(words))
'''
Output:

{'movie': True, ('very', 'good'): True, 'good': True, ('good', 'movie'): True}
'''
```

Working with NLTK's movie reviews corpus

```
from nltk.corpus import movie_reviews

pos_reviews = []
for fileid in movie_reviews.fileids('pos'):
    words = movie_reviews.words(fileid)
    pos_reviews.append(words)

neg_reviews = []
for fileid in movie_reviews.fileids('neg'):
    words = movie_reviews.words(fileid)
    neg_reviews.append(words)
```

Create Feature Set

```
# positive reviews feature set
pos_reviews_set = []
for words in pos_reviews:
    pos_reviews_set.append((bag_of_all_words(words), 'pos'))

# negative reviews feature set
neg_reviews_set = []
for words in neg_reviews:
    neg_reviews_set.append((bag_of_all_words(words), 'neg'))
```

Create Train and Test Set

There are 1000 positive reviews set and 1000 negative reviews set. We take 20% (i.e. 200) of positive reviews and 20% (i.e. 200) of negative reviews as the test set. The remaining negative and positive reviews will be taken as the training set.

```
print (len(pos_reviews_set), len(neg_reviews_set)) # Output: (1000, 1000)

# radomize pos_reviews_set and neg_reviews_set
# doing so will output different accuracy result everytime we run the program
from random import shuffle
shuffle(pos_reviews_set)
shuffle(neg_reviews_set)

test_set = pos_reviews_set[:200] + neg_reviews_set[:200]
train_set = pos_reviews_set[200:] + neg_reviews_set[200:]

print(len(test_set),  len(train_set)) # Output: (400, 1600)
```

Training Classifier and Calculating Accuracy

We train Naive Bayes Classifier using the training set and calculate the classification accuracy of the trained classifier using the test set.

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

```

classifier = NaiveBayesClassifier.train(train_set)

accuracy = classify.accuracy(classifier, test_set)
print(accuracy) # Output: 0.8025

print (classifier.show_most_informative_features(10))
'''
Output:

Most Informative Features
              insulting = True              neg : pos    =     17.0 : 1.0
            outstanding = True              pos : neg    =     14.7 : 1.0
          ('nice', 'see') = True            pos : neg    =     11.7 : 1.0
         ('one', 'worst') = True            neg : pos    =     11.4 : 1.0
       ('would', 'think') = True            neg : pos    =     11.0 : 1.0
        ('quite', 'well') = True            pos : neg    =     11.0 : 1.0
          ('makes', 'no') = True            neg : pos    =     10.3 : 1.0
        ('but', 'script') = True            neg : pos    =     10.3 : 1.0
     ('quite', 'frankly') = True            neg : pos    =     10.3 : 1.0
              animators = True              pos : neg    =     10.3 : 1.0
'''
```

Note:

– The accuracy of the classifier has significantly increased when trained with combined feature set (unigram + bigram).
– Accuracy was 73% while using only Unigram features.
– Accuracy has increased to 80% while using combined (unigram + bigram) features.

Testing Classifier with Custom Review

We provide custom review text and check the classification output of the trained classifier. The classifier correctly predicts both negative and positive reviews provided.

```
from nltk.tokenize import word_tokenize

custom_review = "I hated the film. It was a disaster. Poor direction, bad acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = bag_of_all_words(custom_review_tokens)
print (classifier.classify(custom_review_set)) # Output: neg
# Negative review correctly classified as negative

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: neg
print (prob_result.prob("neg")) # Output: 0.770612685688
print (prob_result.prob("pos")) # Output: 0.229387314312


custom_review = "It was a wonderful and amazing movie. I loved it. Best direction, good acting."
custom_review_tokens = word_tokenize(custom_review)
custom_review_set = bag_of_all_words(custom_review_tokens)

print (classifier.classify(custom_review_set)) # Output: pos
# Positive review correctly classified as positive

# probability result
prob_result = classifier.prob_classify(custom_review_set)
print (prob_result) # Output: <ProbDist with 2 samples>
print (prob_result.max()) # Output: pos
print (prob_result.prob("neg")) # Output: 0.00677736186354
print (prob_result.prob("pos")) # Output: 0.993222638136
```

References:

1. Learning to Classify Text
2. From Text Classification to Sentiment Analysis

Hope this helps. Thanks.

Share this:

Facebook Twitter LinkedIn Print Email

Related Posts:

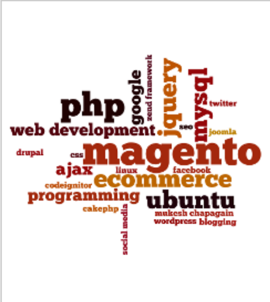
1. **Python NLTK: Twitter Sentiment Analysis [Natural Language Processing (NLP)]**

2. Python NLTK: Text Classification [Natural Language Processing (NLP)]
3. Natural Language Processing (NLP): Basic Introduction to NLTK [Python]
4. Python NLTK: Stop Words [Natural Language Processing (NLP)]
5. Python NLTK: Stemming & Lemmatization [Natural Language Processing (NLP)]
6. Python NLTK: Working with WordNet [Natural Language Processing (NLP)]
7. Python NLTK: Part-of-Speech (POS) Tagging [Natural Language Processing (NLP)]
8. Machine Learning & Sentiment Analysis: Text Classification using Python & NLTK
9. Python: Twitter Sentiment Analysis on Real Time Tweets using TextBlob
10. Python: Twitter Sentiment Analysis using TextBlob

Get New Post by Email

Subscribe

Find me on



ALSO ON **MUKESH CHAPAGAIN'S BLOG**

Node.js, MongoDB & Express: Simple …

3 years ago • 1 comment

This article shows how you can create a simple CRUD (Create, Read, Update, …

Magento 2: Upload File & Image

2 years ago • 1 comment

This article shows how you can upload files and images in Magento 2. You can …

SSH/SCP: Copy Files Folders from Local …

2 years ago • 3 comments

This article shows how you can copy files and folders from your local machine …

0 Comments **Mukesh Chapagain's Blog** **Disqus' Privacy Policy** **Login**

1

Recommend 1 Tweet Share **Sort by Best**

Start the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Be the first to comment.

recent posts

- Magento 2: Create Widget Programmatically & Assign Static Block to it
- Magento 2: Add/Update CMS Static Block via Install/Upgrade Script Programmatically
- Magento 2: Add/Update CMS Page via Install/Upgrade Script Programmatically
- FFMPEG: Convert & Edit Video via Command Line
- ImageMagick: Convert/Edit Multiple Images

most viewed

- How to Calculate Inverter & Battery Backup Time?
- Very Simple Add, Edit, Delete, View (CRUD) in PHP & MySQL [Beginner Tutorial]
- Magento: How to get attribute name and value?
- CodeIgniter: Simple Add, Edit, Delete, View – MVC CRUD Application
- Magento: How to get controller, module, action and router name?

tag cloud

admin array attribute category checkout command line configurable product crud currency customer Database error extension football Google grid HTML image Javascript jQuery latex Linux login machine learning Magento magento2 module MySQL natural language processing NLP nltk nodejs order payment PHP product python query shopping cart template Twitter Ubuntu Wordpress XML

Privacy - Terms