Lesson 9- NLP

Movie Reviews Sentiment Analysis example with Scikit-Learn

Load movie_reviews corpus data through sklearn

In [1]:

```
import sklearn
from sklearn.datasets import load_files
```

In [2]:

```
moviedir = r'D:\Lab\nltk_data\corpora\movie_reviews'
#replace with path in your system for movie_reviews data file above
```

In [3]:

```
# loading all files as training data.
movie_train = load_files(moviedir, shuffle=True)
```

In [4]:

```
len(movie_train.data)
```

Out[4]:

```
2000
```

In [5]:

```
# target names ("classes") are automatically generated from subfolder names
movie_train.target_names
```

Out[5]:

```
['neg', 'pos']
```

In [6]:

```
# First file seems to be about a Schwarzenegger movie.
movie_train.data[0][:500]
```

Out[6]:

```
b"arnold schwarzenegger has been an icon for action enthusiasts , since the
late 80's , but lately his films have been very sloppy and the one-liners a
re getting worse . \nit's hard seeing arnold as mr . freeze in batman and r
obin , especially when he says tons of ice jokes , but hey he got 15 millio
n , what's it matter to him ? \nonce again arnold has signed to do another
expensive blockbuster , that can't compare with the likes of the terminator
series , true lies and even eraser . \nin this so cal"
```

In [7]:

```
# first file is in "neg" folder
movie_train.filenames[0]
```

Out[7]:

```
'D:\\Lab\\nltk_data\\corpora\\movie_reviews\\neg\\cv405_21868.txt'
```

In [8]:

```
# first file is a negative review and is mapped to 0 index 'neg' in target_
names
movie_train.target[0]
```

Out[8]:

0

A detour: try out CountVectorizer & TF-IDF

In [9]:

```
# import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
```

In [10]:

```
# Turn off pretty printing of jupyter notebook... it generates long lines
%pprint
```

```
Pretty printing has been turned OFF
```

In [11]:

```
import nltk
```

In [12]:

```
sents = ['A rose is a rose is a rose is a rose.',
         'Oh, what a fine day it is.',
        "It ain't over till it's over, I tell you!!"]
```

In [13]:

```
# Initialize a CoutVectorizer to use NLTK's tokenizer instead of its
# default one (which ignores punctuation and stopwords).
# Minimum document frequency set to 1.
foovec = CountVectorizer(min_df=1, tokenizer=nltk.word_tokenize)


# sents turned into sparse vector of word frequency counts
import nltk
nltk.download('punkt')
sents_counts = foovec.fit_transform(sents)
# foovec now contains vocab dictionary which maps unique words to indexes
foovec.vocabulary_
```

Out[14]:

```
{'a': 4, 'rose': 14, 'is': 9, '.': 3, 'oh': 12, ',': 2, 'what': 17, 'fine':
7, 'day': 6, 'it': 10, 'ai': 5, "n't": 11, 'over': 13, 'till': 16, "'s": 1,
'i': 8, 'tell': 15, 'you': 18, '!': 0}
```

In [15]:

```
# sents_counts has a dimension of 3 (document count) by 19 (# of unique wor
ds)
sents_counts.shape
```

Out[15]:

```
(3, 19)
```

In [16]:

```
# this vector is small enough to view in full!
sents_counts.toarray()
```

Out[16]:

```
array([[0, 0, 0, 1, 4, 0, 0, 0, 0, 3, 0, 0, 0, 0, 4, 0, 0, 0, 0],
       [0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0],
```

```
       [2, 1, 1, 0, 0, 1, 0, 0, 1, 0, 2, 1, 0, 2, 0, 1, 1, 0, 1]], dtype=in
t64)
```

In [17]:

```
# Convert raw frequency counts into TF-IDF (Term Frequency -- Inverse Docum
ent Frequency) values
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
sents_tfidf = tfidf_transformer.fit_transform(sents_counts)
```

In [18]:

```
# TF-IDF values
# raw counts have been normalized against document length,
# terms that are found across many docs are weighted down
sents_tfidf.toarray()
```

Out[18]:

```
array([[ 0.        ,  0.        ,  0.        ,  0.13650997,  0.54603988,
         0.        ,  0.        ,  0.        ,  0.        ,  0.40952991,
         0.        ,  0.        ,  0.        ,  0.        ,  0.71797683,
         0.        ,  0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.28969526,  0.28969526,  0.28969526,
         0.        ,  0.38091445,  0.38091445,  0.        ,  0.28969526,
         0.28969526,  0.        ,  0.38091445,  0.        ,  0.        ,
         0.        ,  0.        ,  0.38091445,  0.        ],
       [ 0.47282517,  0.23641258,  0.17979786,  0.        ,  0.        ,
         0.23641258,  0.        ,  0.        ,  0.23641258,  0.        ,
         0.35959573,  0.23641258,  0.        ,  0.47282517,  0.        ,
         0.23641258,  0.23641258,  0.        ,  0.23641258]])
```

Back to real data: transforming movie reviews

In [19]:

```
# initialize movie_vector object, and then turn movie train data into a vec
tor
movie_vec = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize)
# use all 25K words. 82.2% acc.
# movie_vec = CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize, max_f
eatures = 3000) # use top 3000 words only. 78.5% acc.
movie_counts = movie_vec.fit_transform(movie_train.data)
```

```
# huge dimensions! 2,000 documents, 25K unique terms.
movie_counts.shape
```

Out[22]:

```
(2000, 25313)
```

In [23]:

```
# Convert raw frequency counts into TF-IDF values
tfidf_transformer = sklearn.feature_extraction.text.TfidfTransformer()
movie_tfidf = tfidf_transformer.fit_transform(movie_counts)
```

In [24]:

```
# Same dimensions, now with tf-idf values instead of raw frequency counts
```

```
movie_tfidf.shape
```

Out[24]:

```
(2000, 25313)
```

Training and testing a Naive Bayes classifier

In [25]:

```
# Now ready to build a classifier.
# We will use Multinominal Naive Bayes as our model
from sklearn.naive_bayes import MultinomialNB
```

In [26]:

```
# Split data into training and test sets
# from sklearn.cross_validation import train_test_split  # deprecated in 0.
18
from sklearn.model_selection import train_test_split
docs_train, docs_test, y_train, y_test = train_test_split(
    movie_tfidf, movie_train.target, test_size = 0.20, random_state = 12)
```

In [27]:

```
# Train a Multimoda Naive Bayes classifier
clf = MultinomialNB().fit(docs_train, y_train)
```

In [28]:

```
# Predicting the Test set results, find accuracy
y_pred = clf.predict(docs_test)
sklearn.metrics.accuracy_score(y_test, y_pred)
```

Out[28]:

```
0.82250000000000001
```

In [29]:

Trying classifier on fake movie reviews

In [30]:

```
# very short and fake movie reviews
reviews_new = ['This movie was excellent', 'Absolute joy ride',
            'Steven Seagal was terrible', 'Steven Seagal shined through.',
              'This was certainly a movie', 'Two thumbs up', 'I fell asleep
halfway through',
              "We can't wait for the sequel!!", '!', '?', 'I cannot recomme
nd this highly enough',
              'instant classic.', 'Steven Seagal was amazing. His performan
ce was Oscar-worthy.']
reviews_new_counts = movie_vec.transform(reviews_new)
reviews_new_tfidf = tfidf_transformer.transform(reviews_new_counts)
```

In [31]:

```
# have classifier make a prediction
pred = clf.predict(reviews_new_tfidf)
```

In [32]:

```
# print out results
```

```
for review, category in zip(reviews_new, pred):
    print('%r => %s' % (review, movie_train.target_names[category]))
```

```
'This movie was excellent' => pos
'Absolute joy ride' => pos
'Steven Seagal was terrible' => neg
'Steven Seagal shined through.' => neg
'This was certainly a movie' => neg
'Two thumbs up' => neg
'I fell asleep halfway through' => neg
"We can't wait for the sequel!!" => neg
'!' => neg
'?' => neg
'I cannot recommend this highly enough' => pos
'instant classic.' => pos
'Steven Seagal was amazing. His performance was Oscar-worthy.' => neg
```

In [33]:

```
# Mr. Seagal simply cannot win!
```

movie_reviews.zip