Movie Reviews Sentiment Analysis with Scikit-Learn

PyLing meeting, Feb 8 2017. Updated Feb 21, 2019

Following this tutorial on scikit-learn.org: http://scikit-learn.org/dev/tutorial/text_analytics/working_with_text_data.html (http://scikit-learn.org/dev/tutorial/text_analytics/working_with_text_data.html)

Adapted to

- work with off-line movie review corpus, which was also covered/used in NLTK book (http://www.nltk.org/book/ch06.html#document-classification), downloadable here (http://www.nltk.org/nltk_data/)
- use the NLTK's tokenizer (so symbols and stopwords are not thrown out)

Also, check out documentation on dataset loading: https://scikit-learn.org/stable/datasets.html (https://scikit-learn.org/stable/datasets.html)

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'

        # loading all files.
        movie = load_files(moviedir, shuffle=True)
```

```
In [3]: len(movie.data)

Out[3]: 2000
```

```
In [4]: # target names ("classes") are automatically generated from subfold
        er names
        movie.target_names

Out[4]: ['neg', 'pos']
```

```
In [5]: # First file seems to be about a Schwarzenegger movie.
        movie.data[0][:500]

Out[5]: b"arnold schwarzenegger has been an icon for action enthusiasts ,
        since the late 80's , but lately his films have been very sloppy a
        nd the one-liners are getting worse . \nit's hard seeing arnold as
        mr . freeze in batman and robin , especially when he says tons of
        ice jokes , but hey he got 15 million , 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 , tru
        e lies and even eraser . \nin this so cal"
```

```
In [6]: # first file is in "neg" folder
        movie.filenames[0]

Out[6]: 'D:\\Lab\\nltk_data\\corpora\\movie_reviews\\neg\\cv405_21868.txt'
```

```
In [7]: # first file is a negative review and is mapped to 0 index 'neg' in
        target_names
        movie.target[0]

Out[7]: 0
```

A detour: try out CountVectorizer & TF-IDF

```
In [8]: # import CountVectorizer, nltk
        from sklearn.feature_extraction.text import CountVectorizer
        import nltk
```

```
In [9]: # Turn off pretty printing of jupyter notebook... it generates long
        lines
        %pprint

        Pretty printing has been turned OFF
```

```
In [10]: # Three tiny "documents"
         docs = ['A rose is a rose is a rose is a rose.',
                 'Oh, what a fine day it is.',
                 "A day ain't over till it's truly over."]
```

```
In [11]: # Initialize a CountVectorizer to use NLTK's tokenizer instead of i
         ts
         #    default one (which ignores punctuation and stopwords).
         # Minimum document frequency set to 1.
         fooVzer = CountVectorizer(min_df=1, tokenizer=nltk.word_tokenize)
```

```
In [12]: # .fit_transform does two things:
         # (1) fit: adapts fooVzer to the supplied text data (rounds up top
         words into vector space)
         # (2) transform: creates and returns a count-vectorized output of d
         ocs
         docs_counts = fooVzer.fit_transform(docs)

         # fooVzer now contains vocab dictionary which maps unique words to
         indexes
         fooVzer.vocabulary_
```

```
Out[12]: {'a': 3, 'rose': 12, 'is': 7, '.': 2, 'oh': 10, ',': 1, 'what': 1
         5, 'fine': 6, 'day': 5, 'it': 8, 'ai': 4, "n't": 9, 'over': 11, 't
         ill': 13, "'s": 0, 'truly': 14}
```

```
In [13]: # docs_counts has a dimension of 3 (document count) by 16 (# of uni
         que words)
         docs_counts.shape
```

```
Out[13]: (3, 16)
```

```
In [14]: # this vector is small enough to view in a full, non-sparse form!
         docs_counts.toarray()
```

```
Out[14]: array([[0, 0, 1, 4, 0, 0, 0, 3, 0, 0, 0, 0, 4, 0, 0, 0],
                [0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1],
                [1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 2, 0, 1, 1, 0]], dtype=in
         t64)
```

```
In [15]: # Convert raw frequency counts into TF-IDF (Term Frequency -- Inver
         se Document Frequency) values
         from sklearn.feature_extraction.text import TfidfTransformer
         fooTfmer = TfidfTransformer()

         # Again, fit and transform
         docs_tfidf = fooTfmer.fit_transform(docs_counts)
```

In [16]:
```
# TF-IDF values
# raw counts have been normalized against document length,
# terms that are found across many docs are weighted down ('a' vs.
'rose')
docs_tfidf.toarray()
```

Out[16]:
```
array([[0.        , 0.        , 0.11337964, 0.45351858, 0.        
,
         0.        , 0.        , 0.4379908 , 0.        , 0.        
,
         0.        , 0.        , 0.7678737 , 0.        , 0.        
,
         0.        ],
       [0.        , 0.39427404, 0.2328646 , 0.2328646 , 0.        
,
         0.29985557, 0.39427404, 0.29985557, 0.29985557, 0.        
,
         0.39427404, 0.        , 0.        , 0.        , 0.        
,
         0.39427404],
       [0.30352608, 0.        , 0.17926739, 0.17926739, 0.3035260
8,
         0.23083941, 0.        , 0.        , 0.23083941, 0.3035260
8,
         0.        , 0.60705216, 0.        , 0.30352608, 0.3035260
8,
         0.        ]])
```

So that completes the feature extraction process for our toy document set

- What if we get new ones?

In [17]:
```
# A list of new documents
newdocs = ["I have a rose and a lily.", "What a beautiful day."]

# This time, no fitting needed: transform the new docs into count-v
ectorized form
# Unseen words ('lily', 'beautiful', 'have', etc.) are ignored
newdocs_counts = fooVzer.transform(newdocs)
newdocs_counts.toarray()
```

Out[17]:
```
array([[0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=in
t64)
```

In [18]:
```
# Again, transform using tfidf
newdocs_tfidf = fooTfmer.transform(newdocs_counts)
newdocs_tfidf.toarray()
```

Out[18]:
```
array([[0.        , 0.        , 0.35653519, 0.71307037, 0.
,
        0.        , 0.        , 0.        , 0.        , 0.
,
        0.        , 0.        , 0.60366655, 0.        , 0.
,
        0.        ],
       [0.        , 0.        , 0.39148397, 0.39148397, 0.
,
        0.50410689, 0.        , 0.        , 0.        , 0.
,
        0.        , 0.        , 0.        , 0.        , 0.
,
        0.66283998]])
```

Back to real data: movie reviews

In [19]:
```
# Split data into training and test sets
from sklearn.model_selection import train_test_split
docs_train, docs_test, y_train, y_test = train_test_split(movie.dat
a, movie.target,
                                                          test_size
= 0.20, random_state = 12)
```

In [20]:
```
# initialize CountVectorizer
movieVzer= CountVectorizer(min_df=2, tokenizer=nltk.word_tokenize,
max_features=3000) # use top 3000 words only. 78.25% acc.
# movieVzer = CountVectorizer(min_df=2, tokenizer=nltk.word_tokeniz
e)         # use all 25K words. Higher accuracy

# fit and tranform using training text
docs_train_counts = movieVzer.fit_transform(docs_train)
```

In [21]:
```
# 'screen' is found in the corpus, mapped to index 2290
movieVzer.vocabulary_.get('screen')
```

Out[21]:
```
2290
```

In [22]:
```
# Likewise, Mr. Steven Seagal is present...
movieVzer.vocabulary_.get('seagal')
```

Out[22]:
```
2297
```

In [23]:
```
# huge dimensions! 1,600 documents, 3K unique terms.
docs_train_counts.shape
```

Out[23]:
```
(1600, 3000)
```

```
In [24]: # Convert raw frequency counts into TF-IDF values
         movieTfmer = TfidfTransformer()
         docs_train_tfidf = movieTfmer.fit_transform(docs_train_counts)
```

```
In [25]: # Same dimensions, now with tf-idf values instead of raw frequency
         counts
         docs_train_tfidf.shape

Out[25]: (1600, 3000)
```

The feature extraction functions and traning data are ready.

- Vectorizer and transformer have been built from the training data
- Training data text was also turned into TF-IDF vector form

Next up: test data

- You have to prepare the test data using the same feature extraction scheme.

```
In [26]: # Using the fitted vectorizer and transformer, tranform the test da
         ta
         docs_test_counts = movieVzer.transform(docs_test)
         docs_test_tfidf = movieTfmer.transform(docs_test_counts)
```

Training and testing a Naive Bayes classifier

```
In [27]: # Now ready to build a classifier.
         # We will use Multinominal Naive Bayes as our model
         from sklearn.naive_bayes import MultinomialNB
```

```
In [28]: # Train a Multimoda Naive Bayes classifier. Again, we call it "fitt
         ing"
         clf = MultinomialNB()
         clf.fit(docs_train_tfidf, y_train)

Out[28]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

```
In [29]: # Predict the Test set results, find accuracy
         y_pred = clf.predict(docs_test_tfidf)
         sklearn.metrics.accuracy_score(y_test, y_pred)

Out[29]: 0.7825
```

```
In [30]: # Making the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, y_pred)
         cm

Out[30]: array([[164,  42],
                [ 45, 149]], dtype=int64)
```

Trying the classifier on fake movie reviews

```
In [31]: # very short and fake movie reviews
         reviews_new = ['This movie was excellent', 'Absolute joy ride',
                     'Steven Seagal was terrible', 'Steven Seagal shone thro
         ugh.',
                       'This was certainly a movie', 'Two thumbs up', 'I fel
         l asleep halfway through',
                       "We can't wait for the sequel!!", '!', '?', 'I cannot
         recommend this highly enough',
                       'instant classic.', 'Steven Seagal was amazing. His p
         erformance was Oscar-worthy.']

         reviews_new_counts = movieVzer.transform(reviews_new)         # tur
         n text into count vector
         reviews_new_tfidf = movieTfmer.transform(reviews_new_counts)  # tur
         n into tfidf vector

In [32]: # have classifier make a prediction
         pred = clf.predict(reviews_new_tfidf)

In [33]: # print out results
         for review, category in zip(reviews_new, pred):
             print('%r => %s' % (review, movie.target_names[category]))

         'This movie was excellent' => pos
         'Absolute joy ride' => pos
         'Steven Seagal was terrible' => neg
         'Steven Seagal shone 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 [34]: # Mr. Seagal simply cannot win!
```

Final notes

- In practice, you should use **`TfidfVectorizer`**, which is `CountVectorizer` and `TfidfTranformer` conveniently rolled into one:

  ```
  from sklearn.feature_extraction.text import TfidfVectorizer
  ```

- Also: It is a popular practice to use **pipeline**, which pairs up your feature extraction routine with your choice of ML model:

  ```
  model = make_pipeline(TfidfVectorizer(), MultinomialNB())
  ```