python_web_scraping_by_harveen_kaur

June 28, 2021

1 What do people think of Titanic (1997)?

1.1 Description:

We will divide this case study into 3 parts. Only the first one is required, the other 2 are just bonuses. All code should be written in Python and uploaded to a git repo.

1.2 Task 1:

Scrape all reviews of Titanic from the following page https://www.imdb.com/title/tt0120338/reviews?ref_=tt_ov_
(hint: python requests or selenium)

1.3 Task 2:

Classify reviews into three clusters (hint: unsupervised learning, classification)

1.4 Task 3:

Extract the 10 most relevant words for each cluster and print them. This will show how good (or how humanly understandable) the classification of the previous task was. You will need to choose a measure for quantifying how "relevant" is a word (hint: TF-IDF).

[1]:
```
from string import punctuation
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
import time
import pandas as pd
import re
import json
import numpy as np
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.cluster import KMeans
from collections import defaultdict
```

[ ]:
```
# Open in headless mode so as to remove the emulator from view
chrome_options = Options()
chrome_options.add_argument("--headless")

# The chromedriver executable is present in the same folder as the notebook
# Change the driver path to the correct one if the notebook is to be executed␣
↪is a different system.
driver = webdriver.Chrome(executable_path = './chromedriver',␣
↪chrome_options=chrome_options)

# Movie link given in the question.
driver.get("https://www.imdb.com/title/tt0120338/reviews?ref_=tt_ov_rt")

# Check if the "Show More" button exists. If it exists, we press it so as to␣
↪obtain the next page of reviews.
# We continue till we get no more pages
while driver.find_element_by_css_selector("#load-more-trigger"):
    try:
        element = WebDriverWait(driver, 20).until(
            EC.element_to_be_clickable((By.ID, "load-more-trigger"))
        )
        element.click()
    except:
        break
```

[3]:
```
# The source of the current website is parsed using beautifulsoup. This source␣
↪contains all the reviews.
html = driver.page_source
driver.close()
html = BeautifulSoup(html)
```

[4]:
```
# Find all the individual review container through class name selector.
user_review_container = html.find_all('div', attrs={'class':'collapsable'})
user_review_container_spoilers = html.find_all('div', attrs={'class':
↪'with-spoiler'})
```

[5]:
```
# Helper function to format the output from beautifulsoup filters.
def format_into_json(review_container):
    '''
    function to format the scrapted reviews in JSON format.
    '''
    res_list = []
    for review in  review_container:
        review_url = review.find('a', href = re.compile(r'[/
↪]([a-z]|[A-Z])\w+')).attrs['href']
        username = review.find('div', attrs={'class':'display-name-date'}).
↪find('span').find('a').text
        user_url = review.find('div', attrs={'class':'display-name-date'}).
↪find('span').find('a', href = re.compile(r'[/]([a-z]|[A-Z])\w+')).
↪attrs['href']
        title = review.find('a', attrs={'class':'title'}).text
        content = review.find('div', attrs={'class':'content'}).find('div').text
        res_list.append({"review_url":'https://www.imdb.com'+review_url,
                         "username":username,
                         "user_url":'https://www.imdb.com'+user_url,
                         "title": title,
                         "content": content,
                        })
    return res_list
```

[6]:
```
# Json data from the scraper content.
json_data = format_into_json(user_review_container) +␣
↪format_into_json(user_review_container_spoilers)
json_data[0]
```

[6]:
```
{'review_url': 'https://www.imdb.com/review/rw3166784/?ref_=tt_urv',
 'username': 'katherinegranada995',
 'user_url': 'https://www.imdb.com/user/ur57176161/?ref_=tt_urv',
 'title': ' Amazing in 1997, 2005, 2015, 2030, 3010 & forever more a
Masterpiece!\n',
 'content': 'You can watch this movie in 1997, you can watch it again in 2004 or
2009 or you can watch it in 2015 or 2020, and this movie will get you EVERY
TIME. Titanic has made itself FOREVER a timeless classic! I just saw it today
(2015) and I was crying my eyeballs out JUST like the first time I saw it back
in 1998. This is a movie that is SO touching, SO precise in the making of the
boat, the acting and the storyline is BRILLIANT! And the preciseness of the ship
makes it even more outstanding! Kate Winslet and Leonardo Dicaprio definitely
created a timeless classic that can be watched time and time again and will
never get old. This movie will always continue to be a beautiful, painful &
tragic movie. 10/10 stars for this masterpiece!'}
```

[7]:
```
# Storing the intermediate result so that we need not run the scraping again.
import json
with open("data.json", "w") as f:
    f.write(json.dumps(json_data))
```

[8]:
```
# Read the data again.
# If any changes are to be done to the clustering, we only need
# to run the cells from this as the parsing logic remains the
# same.
```

```
df = pd.read_json('./data.json', orient='records')
```

[9]:
```
# Check if the data is valid.
df.head()
```

[9]:
```
                                          review_url              username  \
0  https://www.imdb.com/review/rw3166784/?ref_=tt…  katherinegranada995
1  https://www.imdb.com/review/rw5903837/?ref_=tt…       sander-vanluit
2  https://www.imdb.com/review/rw5954378/?ref_=tt…          sucoaramada
3  https://www.imdb.com/review/rw5503104/?ref_=tt…         MR_Heraclius
4  https://www.imdb.com/review/rw4224702/?ref_=tt…         paulclaassen

                                            user_url  \
0  https://www.imdb.com/user/ur57176161/?ref_=tt_urv
1  https://www.imdb.com/user/ur23952614/?ref_=tt_urv
2  https://www.imdb.com/user/ur122456972/?ref_=tt…
3  https://www.imdb.com/user/ur87850731/?ref_=tt_urv
4   https://www.imdb.com/user/ur2263198/?ref_=tt_urv

                                               title  \
0   Amazing in 1997, 2005, 2015, 2030, 3010 & for…
1                                 Why only a 7.8?\n
2                                  Why low score?\n
3                                           Great\n
4   Despite a lot of plot flaws and conveniences,…

                                             content
0  You can watch this movie in 1997, you can watc…
1  There is no movie which made a bigger emotiona…
2  People are crazy. They rate Avengers so high a…
3  Very beautiful and cinematic movie with lots o…
4  Ah, yes, the film that propelled Leonardi DiCa…
```

[10]:
```
# Helper function to sanitize the text.
def custom_initial_clean(phrases_X):
    phrases_X = phrases_X.str.replace('\\*', ' ', regex=True)
    phrases_X = phrases_X.str.replace('\\/', ' ', regex=True)
    phrases_X = phrases_X.str.replace('\\\\', ' ', regex=True)
    phrases_X = phrases_X.str.replace('\n', ' ', regex=True)
    phrases_X = phrases_X.str.replace('\t', ' ', regex=True)
    phrases_X = phrases_X.str.replace('-', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r'/', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r'``', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r'`', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"''", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r",", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"\.$", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r":", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"# ", '#', regex=True)
    phrases_X = phrases_X.str.replace(r";", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"?", ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"=", ' ', regex=True)
    phrases_X = phrases_X.str.replace("...", ' ', regex=False)
    phrases_X = phrases_X.str.replace("..", ' ', regex=False)
    phrases_X = phrases_X.str.replace('<br>', ' ', regex=True)
    phrases_X = phrases_X.str.replace('</br>', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r'LRB', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r'RRB', ' ', regex=True)
    phrases_X = phrases_X.str.replace(r"[C|c]a n't", 'cannot', regex=True)
    phrases_X = phrases_X.str.replace(r"[W|w]o n't", 'will not', regex=True)
    phrases_X = phrases_X.str.replace(r"[W|w]ere n't", 'were not', regex=True)
    phrases_X = phrases_X.str.replace(r"[W|w]as n't", 'was not', regex=True)
    phrases_X = phrases_X.str.replace(r"[W|w]ould n't", 'would not', regex=True)
    phrases_X = phrases_X.str.replace(r"[D|d]oes n't", 'does not', regex=True)
    phrases_X = phrases_X.str.replace(r"[I|i]s n't", 'is not', regex=True)
    phrases_X = phrases_X.str.replace(r"[C|c]ould n't", 'could not', regex=True)
    phrases_X = phrases_X.str.replace(r"[D|d]id n't", 'did not', regex=True)
    phrases_X = phrases_X.str.replace(r"[H|h]as n't", 'has not', regex=True)
    phrases_X = phrases_X.str.replace(r"[H|h]ave n't", 'have not', regex=True)
    phrases_X = phrases_X.str.replace(r"[D|d]o n't", 'do not', regex=True)
    phrases_X = phrases_X.str.replace(r"[A|a]i n't", "not", regex=True)
    phrases_X = phrases_X.str.replace(r"[N|n]eed n't", "need not", regex=True)
    phrases_X = phrases_X.str.replace(r"[A|a]re n't", "are not", regex=True)
    phrases_X = phrases_X.str.replace(r"[S|s]hould n't", "should not",␣
↪regex=True)
    phrases_X = phrases_X.str.replace(r"[H|h]ad n't", "had not", regex=True)
    phrases_X = phrases_X.str.replace(r"https?://\S+", "", regex=True)
    phrases_X = phrases_X.str.replace(r"<.*?>", "", regex=True)
    phrases_X = phrases_X.str.replace(f"[{re.escape(punctuation)}]", "",␣
↪regex=True)
    phrases_X = phrases_X.str.replace(r"[^A-Za-z0-9\s]+", "", regex=True)

    phrases_X = phrases_X.str.replace(" 's", " ", regex=False)
    phrases_X = phrases_X.str.replace("'s", "", regex=False)
    phrases_X = phrases_X.str.replace("'ve", "have", regex=False)
    phrases_X = phrases_X.str.replace("'d", "would", regex=False)
    phrases_X = phrases_X.str.replace("'ll", "will", regex=False)
    phrases_X = phrases_X.str.replace("'m", "am", regex=False)
    phrases_X = phrases_X.str.replace("'n", "and", regex=False)
    phrases_X = phrases_X.str.replace("'re", "are", regex=False)
    phrases_X = phrases_X.str.replace("'til", "until", regex=False)
    phrases_X = phrases_X.str.replace(" ' ", " ", regex=False)
    phrases_X = phrases_X.str.replace(" '", " ", regex=False)
    phrases_X = phrases_X.str.replace(r'[ ]{2,}', ' ', regex=True)
    phrases_X = phrases_X.str.lower()
    return phrases_X


'''

# Helper function to tokenize the sanitized text.
def tokenize_the_text(phrases):

    from nltk.tokenize import word_tokenize
    from nltk.text import Text

    tokens = [word for word in phrases]
    tokens = [word.lower() for word in tokens]
    tokens = [word_tokenize(word) for word in tokens]

    return tokens

# Helper function to remove the stop words from the tokens. This is done to␣
↪remove words
# that do not add meaning to the sentance and exist only to satisy the␣
↪grammatical requirements
# of the language.
def removing_stopwords(tokens_custom_cleaned):

    from nltk.corpus import stopwords
    stop_words = stopwords.words('english')
    tokens_custom_cleaned_and_without_stopwords = []
    for sentence in tokens_custom_cleaned:
        tokens_custom_cleaned_and_without_stopwords.append([word for word in␣
↪sentence if word not in stop_words])

    return tokens_custom_cleaned_and_without_stopwords

# Helper function to lemmatize the tokens.
def lemmatizing_the_tokens(tokens_custom_cleaned_and_without_stopwords):

    from nltk.stem.wordnet import WordNetLemmatizer
    lem = WordNetLemmatizer()

    tokens_custom_cleaned_and_without_stopwords_and_lemmatized = []

    for sentence in tokens_custom_cleaned_and_without_stopwords:
        tokens_custom_cleaned_and_without_stopwords_and_lemmatized.append([lem.
↪lemmatize(word, pos='v') for word in sentence])
```

```
    return tokens_custom_cleaned_and_without_stopwords_and_lemmatized

# Helper function to find the vocab list from the corpus.
def create_a_vocab(tokens):

    vocab = set()

    for setence in tokens:
        for word in setence:
            vocab.add(word)

    vocab = list(vocab)

    return vocab

'''
```

[11]:
```
# Sanitize the title and content from the dataframe.
title, content = custom_initial_clean(df.title.copy()), custom_initial_clean(df.
 ↪content.copy())
```

[12]:
```
# Merge the title and content so as to simplify the tokenizing.
data = title + content
```

[13]:
```
# Use the tfidf vectorizer from nltk.
vectorizer = TfidfVectorizer(stop_words={'english'})
X = vectorizer.fit_transform(data)
```

[14]:
```
# Extract the feature map so as to do a reverse lookup.
feature_names = np.array(vectorizer.get_feature_names())
```

[16]:
```
# Run k-means with k = 3
km = KMeans(n_clusters=3, max_iter=200, n_init=10)
km = km.fit(X)
```

[18]:
```
# Run the fitted model against the data so as to classify the data into␣
 ↪clusters.
predictions = defaultdict(list)
for i in range(X.shape[0]):
    x = X[i]
    predictions[km.predict(x)[0]].append(title.values[i])
    predictions[km.predict(x)[0]].append(content.values[i])
```

[19]:
```
# Helper function to get the most prominent words from the input
def get_top_tf_idf_words(response, top_n=10):
    sorted_nzs = np.argsort(response.data)[:-(top_n+1):-1]
    return feature_names[response.indices[sorted_nzs]]
```

```python
# Get the top 10 most significant words from the 3 clusters.
for i in range(3):
    print(get_top_tf_idf_words(vectorizer.transform(predictions[i])))
```

```
['epic' 'romantic' 'omg' 'meh' 'negative' 'best' 'blah' 'naaaah' 'awful'
 'favourite']
['amazing' 'classic' 'zarina' 'finally' 'cool' 'awesome' 'magic' 'amazing'
 'epochal' 'titanic']
['overrated' 'good' 'meh' 'titanic' 'masterpiece' 'titanic' 'romantic'
 'classic' 'haunting' 'complimentary']
```

```
[ ]:
```