TripAdvisor Review Classifier

March 18, 2017

Author: Konstantinos Oikonomou

1 This program:

1. Scrapes all hotel reviews from a hotel that has mixed reviews in TripAdvisor
2. Cleans and pre-processes the data
3. Recodes the ratings: 1-3 Negative , 4-5 Positive
4. Splits the data 75/25 and uses a naïve Bayesian classifier
5. Prints the classifier's accuracy percentage
6. Prints the 10 most informative features

2 Imports and creating the dataset

In the beginning of my program, I import all the libraries that I will use later in the program

- From the `nltk` library I will use several classes that will help me with data normalization, the list with english stopwords and, finally, the Naïve Bayes Classifier itself.
- I will use the `requests` library in order to get the source code from every page I have to scrape data from
- I will use the `random` library in order to shuffle my data so that my classifier is more accurate
- I will use the `re` library to extract the rating of each review
- Finally, I will use `BeautifulSoup` in order to parse the html code and extract the useful information (data) that I need

Then, I add the urls from which I will scrape reviews (there are 3 pages for the specific hotel) into the list `urls` . The final output of the *Parsing and Getting the Data* Section will be a list of tuples named `documents` . These tuples will contain 2 elements:

- the review text (stripped from html code)
- the classification of the specific review (pos or neg)

```
In [5]: #-----------IMPORTS-----------
        import nltk
        import requests
        import random
        import re
```

```
from bs4 import BeautifulSoup
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
#------------------------------

#----------- Parsing and Getting the Data -----------

#the hotel's review urls (each page)
urls = ['https://www.tripadvisor.com/Hotel_Review-g187147-d586832-Reviews-Sibour_Hotel-P
        'https://www.tripadvisor.com/Hotel_Review-g187147-d586832-Reviews-or10-Sibour_Ho
        'https://www.tripadvisor.com/Hotel_Review-g187147-d586832-Reviews-or20-Sibour_Ho

#documents is a list of tuples containing the review and the classification
documents = []

#iterate through each page
for url in urls:
    #get the source code
    source = requests.get(url).text
    soup = BeautifulSoup(source, 'html.parser')

    #find the reviews
    revs = soup.findAll("p", { "class" : "partial_entry" })

    #normalise a bit
    reviews = [str(review).strip('<p class="partial_entry">').strip('</p>') for review i

    #find the ratings
    rat = soup.findAll("div", {'class' : 'rating reviewItemInline'})

    temp = [re.search(r'ui_bubble_rating bubble_..', str(i)).group(0) for i in rat]

    #convert the bubbles into negative and positive
    # 1-3 Negative , 4-5 Positive
    ratings = []
    for item in temp:
        if int(item[-2]) <= 3:
            ratings.append('neg')
        else:
            ratings.append('pos')

    for x, y in zip(reviews, ratings):
        documents.append((x, y))


#-----------------------------------------------------
```

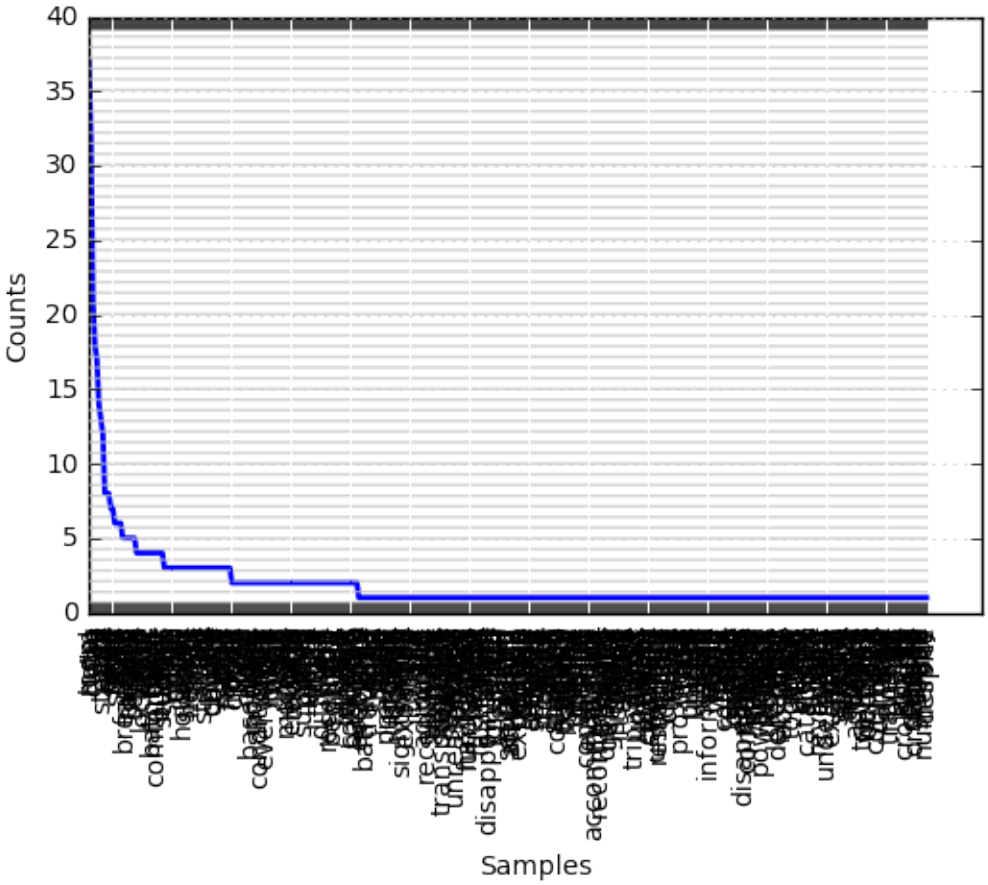
3 Text Normalization

The next section of the program is the `Text Normalization`. Regular Expressions helped me disregard all of the punctuation and spaces in my text, so it worked for the specific case. Another option could be the `nltk.tokenize.word_tokenize` but this could also tokenize punctuation, which I definitelly did not want to. I preferred the WordNetLemmatizer to lemmatize my tokens. Then, I removed all stopwords from the tokens list.

After this process, my `documents` list contained tuples that had lists as their first element, containing the **tokens** that belonged to each review.

After that, I created a list that would contain every single word that appears in all of my `documents`. Then, I turned that list into an `nltk` Frequency Distribution in order for me to visualize what the most frequent terms in a review are.

```
In [6]: #--------------- Further Normalisation -----------------

        tokenizer = RegexpTokenizer(r'\w+')
        lemmatizer = WordNetLemmatizer()

        #Tokenize
        for index, item in enumerate(documents):
            documents[index] = (tokenizer.tokenize(item[0]), item[1])

        #Lemmatize, Lowercase and remove StopWords
        for index, item in enumerate(documents):
            documents[index] = ([lemmatizer.lemmatize(word).lower() for word in item[0] if word

        all_words = []
        for item in documents:
            all_words.extend(item[0])

        all_words = nltk.FreqDist(all_words)

        all_words.plot()

        #-------------------------------------------------------
```



4 Classification Using the `Naive Bayes Classifier`

Finally, the last part of the program is the classification by using the Naive Bayes Classifier that is installed in the NLTK library

4.1 Features:

The features that my Naïve Bayes Classifier would classify the data upon were on me to decide. I chose to have a single feature: for each word in `all_words` my feature would be the presence or absence of this word in the specific review. For Example: The word `extraordinary` appears in one of all the reviews, however it may not appear in a specific one. This would result for the specific feature set to have an element of *`extraordinary = False`* With this process, I obtain my feature sets, different for each review, that I will use in order to train and test my Naïve Bayes Classifier.

4.2 Splitting the data

I know that there are 25 reviews in total, so I chose my training set to be the first 19 of them (aprox. 75%) and my testing set to be the last 6.

4.3 Training

Finally, I train my classifier and output the needed results. NLTK was really helpful here since it provided me with ready methods and functions that otherwise would be ~~Impossible~~ hard to create.

```
In [7]: #---------- Classification with NB Classifier ----------

        #shuffle for better distribution of categories
        random.shuffle(documents)

        word_features = list(all_words.keys())
        def find_features(doc):
            words = set(doc)
            features = {}
            for w in word_features:
                features[w] = (w in words)

            return features

        featuresets = [(find_features(rev), category) for (rev, category) in documents]

        #since all of the documents are 25, we use the first 19 for training and the rest for te
        # 75/25
        training_set = featuresets[:19]
        testing_set = featuresets[19:]

        classifier = nltk.NaiveBayesClassifier.train(training_set)
        print('The classifier is {} percent accurate.'.format(nltk.classify.accuracy(classifier,
        classifier.show_most_informative_features(10)

        #------------------------------------------------------
```

```
The classifier is 66.66666666666666 percent accurate.
Most Informative Features
                   close = True              pos : neg    =      3.1 : 1.0
                friendly = True              pos : neg    =      3.1 : 1.0
                   staff = True              pos : neg    =      3.1 : 1.0
                   great = True              pos : neg    =      3.1 : 1.0
                   clean = False             neg : pos    =      2.2 : 1.0
                 student = True              pos : neg    =      2.2 : 1.0
                 station = True              pos : neg    =      2.2 : 1.0
                   quite = True              pos : neg    =      2.2 : 1.0
                 helpful = True              pos : neg    =      2.2 : 1.0
                  shower = True              pos : neg    =      2.2 : 1.0
```

5 The results:

I could say that I am quite satisfied by the results. I needed the hotel to have mixed reviews so that narrowed down the options of chosing big and famous hotels. This meant that I would have a really small dataset (only 25 reviews). However, 2/3 percentage of `accuracy` was nice and the `most informative features` made sense to me.

6 Possible Improvements:

Obviously there are lots of ways that this program could be improved. Some of them are: - Chosing a hotel that had more reviews (that would still be mixed) (No lots of research) - Using python list comprehensions to perform some processes: this would give a little boost in the speed of the algorithm. - I could store the trained classifier in a pickle file so that I don't have to train it each time I run the program: this would boost the speed **a lot**. - Turning the `all_words` list into a Frequency Distribution does not provide anything to the algorithm, however I did it for my eyes satisfaction (I liked the graph)

These were only some improvements that I could think from the top of my head.