Dataset for sentiment analysis

IMDB Movie Reviews Dataset

This large movie dataset contains a collection of about 50,000 movie reviews from IMDB. In this dataset, only highly polarised reviews are being considered. The positive and negative reviews are even in number; however, the negative review has a score of ≤ 4 out of 10, and the positive review has a score of ≥ 7 out of 10.

Sentiment140

Sentiment140 is used to discover the sentiment of a brand product or even a topic on the social media platform Twitter. Rather than working on a keywords-based approach, which leverages high precision for lower recall, Sentiment140 works with classifiers built from machine learning algorithms. The Sentiment140 uses classification results for individual tweets along with the traditional surface that aggregates metrics. The Sentiment140 is used for brand management, polling, and planning a purchase.

Twitter US Airline Sentiment

This sentiment analysis dataset contains tweets since Feb 2015 about each of the major US airlines. Each tweet is classified as either positive, negative, or neutral. The included features include Twitter ID, sentiment confidence score, sentiments, negative reasons, airline name, retweet count, name, tweet text, tweet coordinates, date and time of the tweet, and the location of the tweet.

Amazon Product Data

Amazon product data is a subset of a large 142.8 million Amazon review dataset that was made available by Stanford professor, Julian McAuley. This sentiment analysis dataset contains reviews from May 1996 to July 2014. The dataset reviews include ratings, text, helpful votes, product descriptions, category information, price, brand, and image features.

Stanford Sentiment Treebank

This dataset contains just over 10,000 pieces of Stanford data from HTML files of Rotten Tomatoes. The sentiments are rated between 1 and 25, where 1 is the most negative and 25 is the most positive. The deep learning model by Stanford has been built on the representation of sentences based on sentence structure instead of just giving points based on positive and negative words.

For example:

The Interview was neither that funny nor that witty.

Even if there are words like funny and witty, the overall structure is a negative type.

Multi-Domain Sentiment Dataset

This dataset contains positive and negative files for thousands of Amazon products. Although the reviews are for older products, this data set is excellent to use. The data was derived from the Department of Computer Science at Johns Hopkins University.

The reviews contain ratings from 1 to 5 stars that can be converted to binary as needed

Sentiment Classification

This dataset described in the previous paragraph contains review comments on several movies. Comments in the dataset are already labeled as either positive or negative. The dataset contains the following two fields separated by a tab character.

1. text:- Actual review comment

2. sentiment:- Positive sentiments are labelled as 1 and negative sentiments are labelled as 0.

Now this article will discuss a few functions of preprocessing of text datasets.

Text Pre-Processing

Unlike structured data, features are not explicitly available in text data. Thus we need to use a process to extract features from the text data. One way is to consider each word as a feature and find a measure to capture whether a word exists or does not exist in a sentence. This is called the bag-of-words(BoW) model. That is each sentence is treated as a bag of words. Each sentence is called a document and the collection of all documents is called corpus.

This is a list of preprocessing functions that can be performed on text data such as:

1. Bag-of_words(BoW) Model
2. creating count vectors for the dataset
3. Displaying Document Vectors
4. Removing Low-Frequency Words
5. Removing Stop Words
6. Distribution of words Across Different sentiments

Bag-of_words(BoW) Model

The first step in creating a Bow Model is to create a dictionary of all the words used in the corpus. At this stage, we will not worry about grammar and only the occurrence of the words is captured. Then we will convert each document to a vector that represents words available in the documents. There are three ways to identify the importance of words in the BoW Model:

- Count Vector Model
- Term Frequency Vector Model
- Term Frequency-Inverse Document Frequency(TF-IDF) Model

Creating Count Vectors for Dataset

Each document in the dataset needs to be transformed into TF or TF-IDF vectors sklearn.feature_extraction.text module provides classes for creating both TF and TF-IDF vectors from text data. We will use CountVectorizer to create count vectors.

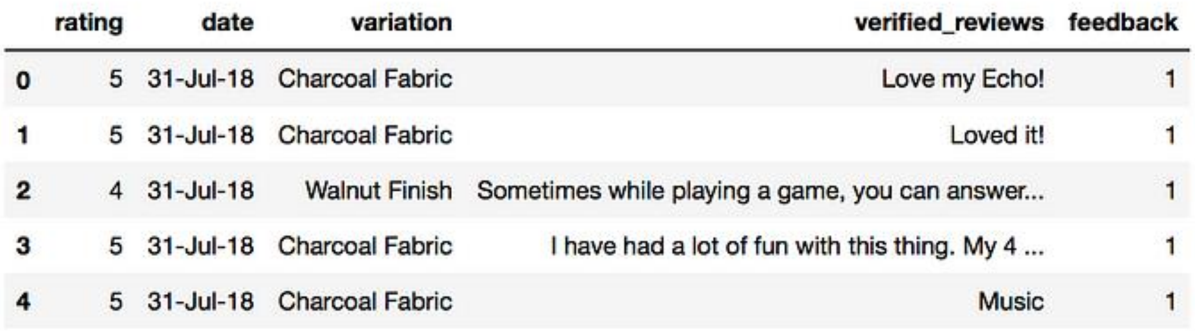
Displaying Document Vectors

To visualize the count vectors, we will convert this matrix into a dataframe and set the column names to the actual feature names.

Removing Low_Frequency Words

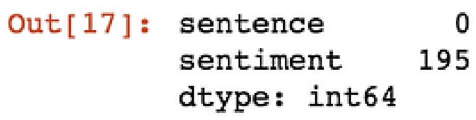
One of the challenges of dealing with text is the number of words of features available in the corpus is too large. The number of features could easily go over tens of thousands. The frequency of each feature or word can be analyzed using the histogram. To calculate the total occurrence of each feature or word, we will use the np. sum() method. The histogram showed that a large number of features have very rare occurrences.

```
import pandas as pd
import numpy as np
import xml.etree.ElementTree as ETxml_path =
'./NLP/ABSA15_RestaurantsTrain2/ABSA-15_Restaurants_Train_Final.xml'def
parse_data_2015(xml_path):
    container = []
    reviews = ET.parse(xml_path).getroot()

    for review in reviews:
        sentences = review.getchildren()[0].getchildren()
        for sentence in sentences:
            sentence_text = sentence.getchildren()[0].text

            try:
                opinions = sentence.getchildren()[1].getchildren()

                for opinion in opinions:
                    polarity = opinion.attrib["polarity"]
                    target = opinion.attrib["target"]

                    row = {"sentence": sentence_text, "sentiment":polarity}
                    container.append(row)

            except IndexError:
                row = {"sentence": sentence_text}
                container.append(row)

    return pd.DataFrame(container)ABSA_df = parse_data_2015(xml_path)
ABSA_df.head()
```



	rating	date	variation	verified_reviews	feedback
0	5	31-Jul-18	Charcoal Fabric	Love my Echo!	1
1	5	31-Jul-18	Charcoal Fabric	Loved it!	1
2	4	31-Jul-18	Walnut Finish	Sometimes while playing a game, you can answer...	1
3	5	31-Jul-18	Charcoal Fabric	I have had a lot of fun with this thing. My 4 ...	1
4	5	31-Jul-18	Charcoal Fabric	Music	1

```
ABSA_df.isnull().sum()
```



```
Out[17]: sentence       0
         sentiment    195
         dtype: int64
```

```
print "Original:", ABSA_df.shape
ABSA_dd = ABSA_df.drop_duplicates()
dd = ABSA_dd.reset_index(drop=True)
print "Drop Dupicates:", dd.shape
dd_dn = dd.dropna()
df = dd_dn.reset_index(drop=True)
print "Drop Nulls:", df.shape
```

```
Original: (1849, 2)
Drop Dupicates: (1396, 2)
Drop Nulls: (1201, 2)
```

Getting started with Text Preprocessing

```python
import numpy as np
import pandas as pd
import re
import nltk
import spacy
import string
pd.options.mode.chained_assignment = None

full_df = pd.read_csv("../input/customer-support-on-twitter/twcs/twcs.csv", nrows=5000)
df = full_df[["text"]]
df["text"] = df["text"].astype(str)
full_df.head()
```

```python
df["text_lower"] = df["text"].str.lower()
df.head()
```

Removal of Punctuations

```python
# drop the new column created in last cell
df.drop(["text_lower"], axis=1, inplace=True)

PUNCT_TO_REMOVE = string.punctuation
def remove_punctuation(text):
    """custom function to remove the punctuation"""
    return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

df["text_wo_punct"] = df["text"].apply(lambda text: remove_punctuation(text))
df.head()
```

Removal of stopwords

```python
from nltk.corpus import stopwords", ".join(stopwords.words('english'))
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df["text_wo_stop"] = df["text_wo_punct"].apply(lambda text: remove_stopwords(text))
df.head()
```

Removal of Frequent words

```
from collections import Counter
cnt = Counter()
for text in df["text_wo_stop"].values:
    for word in text.split():
        cnt[word] += 1

cnt.most_common(10)
```

```
Out[6]:
[('I', 1437),
 ('us', 752),
 ('DM', 514),
 ('help', 479),
 ('Please', 376),
 ('We', 338),
 ('Hi', 293),
 ('Thanks', 287),
 ('get', 279),
 ('please', 247)]
```

```
In [7]:
linkcode
FREQWORDS = set([w for (w, wc) in cnt.most_common(10)])
def remove_freqwords(text):
    """custom function to remove the frequent words"""
    return " ".join([word for word in str(text).split() if word not in FREQWORDS])

df["text_wo_stopfreq"] = df["text_wo_stop"].apply(lambda text: remove_freqwords(text))
df.head()
```