Natural Language Processing: Lab 1 - Preprocessing

In this Notebook, we are going to exam the first lab in Natural Language Processing DeepLearning.AI Course.

Objectives and Layouts:

- Jupyter - Lite Notebook Familiarization.
- Natural Language Processing Python Libraries.
- Preprocessing of given twitter dataset.

Part A: Natural Language Processing Python Library

We will be using the Natural Language Toolkit (NLTK) package, an open-source Python library for natural language processing. It has modules for collecting, handling, and processing Twitter data. For this exercise, we will use a Twitter dataset that comes with NLTK.

NLTK Library

In [1]:
```
import nltk                                # Python Library for Natural Language Processing
from nltk.corpus import twitter_samples # NLTK twitter dataset
```

General Helping Libraries

In [2]:
```
import matplotlib.pyplot as plt         # Matlab Python Library for visualization purposes
import random                           # Pseudo Random Number Generator
```

Twitter DataSet Overview

The sample dataset from NLTK is separated into positive and negative tweets. It contains 5000 positive tweets and 5000 negative tweets exactly. The exact match between these classes is not a coincidence. The intention is to have a balanced dataset. That **does not** reflect the real

distributions of positive and negative classes in live Twitter streams. It is just because balanced datasets simplify the design of most computational methods that are required for sentiment analysis. However, it is better to be aware that this balance of classes is **artificial**.

In [3]:
```
import nltk                                 # Python Library for Natural Language Processing
from nltk.corpus import twitter_samples # NLTK twitter dataset
#----------------------------------------------------------------------------------------
import matplotlib.pyplot as plt         # Matlab Python Library for visualization purposes
import random                               # Pseudo Random Number Generator
#----------------------------------------------------------------------------------------
# select the set of positive and negative tweets
print("Nope")
# downloads sample twitter dataset.
nltk.download('twitter_samples')
```

```
Nope
[nltk_data] Downloading package twitter_samples to
[nltk_data]     /home/julian/nltk_data...
[nltk_data]   Package twitter_samples is already up-to-date!
```

Out[3]:
```
True
```

In [4]:
```
#select the set of positive and negative tweets
all_positive_tweets = twitter_samples.strings('positive_tweets.json')
all_negative_tweets = twitter_samples.strings('negative_tweets.json')
```

We will now display the number of all positive as well as all negative tweets:

In [5]:
```
# display the length of all positive tweets
print('Number of positive tweets: ', len(all_positive_tweets))
# display the length of all negative tweets
print('Number of negative tweets: ', len(all_negative_tweets))
```

```
Number of positive tweets:  5000
Number of negative tweets:  5000
```

One important question may occur as what information do we have about the structure of the data under consideration. Let us first review the type of "all positive tweets".

In [6]:
```
print(type(all_positive_tweets))
```

```
<class 'list'>
```

Therefore we understand that we are dealing with a list, meaning that we have to acess any of the given 5,000 tweets by a numerical index 0 - 4,999. Let us now review the type of a tweet (or a list entry):

In [7]:
```
tweet_entry = all_positive_tweets[0]
print(tweet_entry)
```

```
#FollowFriday @France_Inte @PKuchly57 @Milipol_Paris for being top engaged members in my community this week :)
```

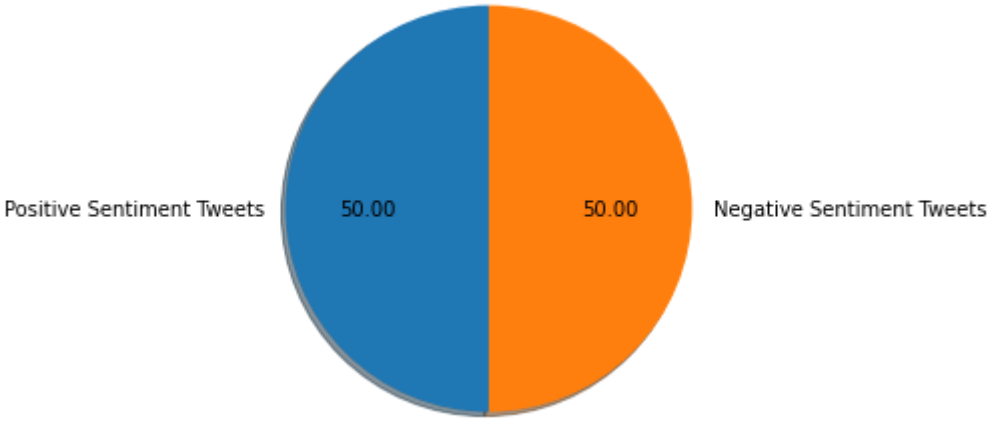
By the prementioned example it is clear that we are refering to a straight-forward string: namely, the tweet itself.

Matplotlib Visualization : Pie Chart

By the following example, we are going to examine a code snipet that is going to present to us, a graphical representation of the distribution of the data at hand (via the matplotlib library).

In [8]:
```
# Declare a figure with a custom size
# Input : (float,float) representing width and height in inches (1 inch = 2.54 cm)
fig = plt.figure(figsize=(4, 4))

# Labels for the two classes
labels = 'Positive Sentiment Tweets', 'Negative Sentiment Tweets'

# Sizes for each slide
sizes = [len(all_positive_tweets), len(all_negative_tweets)]

# Declare pie chart, where the slices will be ordered and plotted counter-clockwise:
# Input:
#     sizes: the wedge sizes
#     labels: the labels
#     autopct: A string or function used to label the wedges with their numeric value.
#              The label will be placed inside the wedge.
#              If it is a format string, the label will be fmt % pct.
#              If it is a function, it will be called.
#     shadow: Draw a shadow beneath the pie.
#     startangle: The angle by which the start of the pie is rotated, counterclockwise from the x-axis.
plt.pie(sizes, labels=labels, autopct='%.2f', shadow=True, startangle=90)

# Equal aspect ratio ensures that pie is drawn as a circle.
plt.axis('equal')
```

```
# Display the chart
plt.show()
```



Data Initial Overview

Before starting the actual preprocessing functionality, it is constructive to initially observe the data at hand, in order to make observations regarding some common structural components of them. It is estimated, that understanding the data is responsible for 80% of the success or failure in data science projects. We can use this time to observe aspects we'd like to consider when preprocessing our data.

In [9]:
```
#List of ASCII common color codes
Red = '\033[91m'
Green = '\033[92m'
Blue = '\033[94m'
Cyan = '\033[96m'
White = '\033[97m'
Yellow = '\033[93m'
Magenta = '\033[95m'
Grey = '\033[90m'
Black = '\033[90m'
Default = '\033[99m'

# Print a random integer in range [0-4999]
# random.randint(0,5000)
```

In [10]:
```
# Print positive in greeen
print('\033[92m' + all_positive_tweets[random.randint(0,4999)])
```

```
print('\033[92m' + all_positive_tweets[random.randint(0,4999)])
print('\033[92m' + all_positive_tweets[random.randint(0,4999)])
print('\033[92m' + all_positive_tweets[random.randint(0,4999)])
print('\033[92m' + all_positive_tweets[random.randint(0,4999)] + '\n')

# print negative in red
print('\033[91m' + all_negative_tweets[random.randint(0,4999)])
print('\033[91m' + all_negative_tweets[random.randint(0,4999)])
print('\033[91m' + all_negative_tweets[random.randint(0,4999)])
print('\033[91m' + all_negative_tweets[random.randint(0,4999)])
print('\033[91m' + all_negative_tweets[random.randint(0,4999)])
```

```
@Prox_Tourney seed 3 beat seed 9 on solar 6-5 as shown thank you! :) http://t.co/jp3Wuu6zIK
@TonyRobbins Absolutely True :) :) :)
Hi BAM ! @BarsAndMelody
Can you follow my bestfriend @969Horan696 ?
She loves you a lot :)
See you in Warsaw &lt;3
Love you &lt;3 x45
@stumac1974 Urquhart Castle: http://t.co/bSgiJFpICc Find all events here: http://t.co/cCS5QGfkk7 Hope that helps :)
With them :)

&lt;3 &lt;3 awsme song &lt;3 :-* :-( :-( :'( http://t.co/IjVWwO32eO
Oh how horrific :( https://t.co/IObpbythB8
*sign out* :(((((( https://t.co/y45q9GAJ5j
Gutted for Reynold. The kind of desserts he has done this season :(
Have a shoot tomorrow and haven't been told where and when because the team are only flying in this evening :(
I hate being unprepared.
```

One observation you may have is the presence of emoticons and URLs in many of the tweets. This info will come in handy in the next steps.

Part B: Preprocessing for sentiment Analysis

Data preprocessing is one of the critical steps in any machine learning project. It includes cleaning and formatting the data before feeding into a machine learning algorithm. For NLP, the preprocessing steps are comprised of the following tasks:

- Tokenizing the string
- Lowercasing
- Removing stop words and punctuation
- Stemming

Let us review a specific example from the dataset, which will provide usefull insights.

In [11]:
```
# Our selected sample. Complex enough to exemplify each step
tweet = all_positive_tweets[2277]
print(tweet)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
```

For exhibition purpsoses, we are going to need to download some aditionall helper libraries. Firstly, we have to download the list of stopwords that NLTK provides:

In [12]:
```
# download the stopwords from NLTK
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /home/julian/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Out[12]:
```
True
```

In [13]:
```
import re                                   # Library for Regular Expression (RegEx) operations
import string                               # Library for String operations

from nltk.corpus import stopwords           # Module for stop - words that come with NLTK
from nltk.stem import PorterStemmer         # Module for Stemming
from nltk.tokenize import TweetTokenizer    # Module for Tokenizing strings
```

Remove Redundancy

In this section, we are going to remove any redundand information that is mainly related to the nature of the string under consideration. For example, as viewed previously, twitter messages commonly contain hashtags ('#'), re-tweets ('RE') or hyperlinks ('https:...'). In this step, we are going to replace them with an empty charachter, using regular expressions. For a complete documentation of re python module, visit the documentation.

In [28]:
```
print('\033[92m' + tweet)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
```

In [27]:
```
# Substitute (pattern,replacment,string origin)
# r'Raw data encapsulated here'
```

```
# Matches the start of the string, and in MULTILINE mode also matches immediately after each newline.
# [] Used to indicate a set of characters. In a set:
# \s Matches whitespace (spaces, tabs and new lines)
# + Causes the resulting RE to match 1 or more repetitions of the preceding RE.
data = re.sub(r'^RT[\s]+', '', tweet)
print('\033[92m' + tweet)
print('\033[94m'+data)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
```

In [26]:
```
# Remove hyperlinks
# ? Causes the resulting RE to match 0 or 1 repetitions of the preceding RE.
data = re.sub(r'https?://[^\s\n\r]+', '', data)
print('\033[92m' + tweet)
print('\033[94m'+data)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…
```

In [22]:
```
# Remove hashtags by only removing the hash # sign from the word
data = re.sub(r'#', '', data)
print('\033[92m' + tweet)
print('\033[94m'+data)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…
```

Tokenize

In this section, we are going to tokenize the string. By tokenization, we refer to the proceedure of extracting every single discrete element of the string (token). Additionally, the tokenize module from NLTK, allows us to additionally lowercase each token.

In [29]:
```
print('\033[92m' + data)
```

```
My beautiful sunflowers on a sunny Friday morning off :) #sunflowers #favourites #happy #Friday off… https://t.co/3tfYom0N1i
```

In [25]:
```
# Instantiate tokenizer class
tokenizer = TweetTokenizer(preserve_case=False, strip_handles=True, reduce_len=True)
# tokenize tweets
data_tokens = tokenizer.tokenize(data)
print('\033[92m' + data)
```

```
print('\033[94m'+'Tokenized string:')
print('\033[94m'+str(data_tokens))
```

```
My beautiful sunflowers on a sunny Friday morning off :) sunflowers favourites happy Friday off…
Tokenized string:
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'fr
iday', 'off', '…']
```

Remove stopwords and punctuations

The next step is to remove stop words and punctuation. Stop words are words that don't add significant meaning to the text under consideration.
Let us review examples via the below - mentioned code:

In [30]:
```
#Import the english stop words list from NLTK
stopwords_english = stopwords.words('english')

print('Stop words\n')
print(stopwords_english)

print('\nPunctuation\n')
print(string.punctuation)
```

```
Stop words

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yo
urself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself',
'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'a
m', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an',
'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'betwee
n', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ove
r', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each',
'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's',
't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren',
"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'i
sn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'was
n', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

Punctuation

!"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
```

We can see that the stop words list above contains some words that could be important in some contexts. These could be words like i, not,

between, because, won, against. You might need to customize the stop words list for some applications. For our exercise, we will use the entire list.

For the punctuation, we saw earlier that certain groupings like ':)' and '...' should be retained when dealing with tweets because they are used to express emotions. In other contexts, like medical analysis, these should also be removed.

Let us proceed.

In [32]:
```
print('\033[92m'+str(data_tokens))
data_clean = []
for word in data_tokens: # Go through every word in your tokens list
    if (word not in stopwords_english and  # remove stopwords
        word not in string.punctuation):  # remove punctuation
        data_clean.append(word)

print('\033[94m'+'removed stop words and punctuation:')
print('\033[94m'+str(data_clean))
```

```
['my', 'beautiful', 'sunflowers', 'on', 'a', 'sunny', 'friday', 'morning', 'off', ':)', 'sunflowers', 'favourites', 'happy', 'fr
iday', 'off', '…']
removed stop words and punctuation:
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites', 'happy', 'friday', '…']
```

Stemming

Stemming is the process of converting a word to its most general form, or stem. This helps in reducing the size of our vocabulary.For example, the words:

- (learn)
- (learn)ing
- (learn)ed
- (learn)t

are stemmed from its common root: learn.

However, in some cases, the stemming process produces words that are not correct spellings of the root word. For example, happi and sunni. That's because it chooses the most common stem for related words. For example, we can look at the set of words that comprises the different forms of happy:

- (happy)
- (happi)ness
- (happi)er

We can see that the prefix happi is more commonly used. We cannot choose happ because it is the stem of unrelated words like happen.

NLTK has different modules for stemming and we will be using the PorterStemmer module which uses the Porter Stemming Algorithm.

In [36]:
```
print('\033[92m'+str(data_clean))

# Instantiate stemming class
stemmer = PorterStemmer()

# Create an empty list to store the stems
data_stem = []

for word in data_clean:
    stem_word = stemmer.stem(word)  # stemming word
    data_stem.append(stem_word)  # append to the list

print('\033[94m'+'stemmed words:')
print('\033[94m'+str(data_stem))
```

```
['beautiful', 'sunflowers', 'sunny', 'friday', 'morning', ':)', 'sunflowers', 'favourites', 'happy', 'friday', '…']
stemmed words:
['beauti', 'sunflow', 'sunni', 'friday', 'morn', ':)', 'sunflow', 'favourit', 'happi', 'friday', '…']
```