Natural Language Processing (NLP).

Prepared by: Sagun Shakya (https://github.com/sagsshakya)

- GITAM Institute of Science.

Importing the libraries.

Getting the dataset.

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

In [2]:

```
os.chdir(r'C:\Users\acer\Desktop\P14-Machine-Learning-AZ-Template-Folder\Machine
df = pd.read_csv('Restaurant_Reviews.tsv', delimiter = '\t', quoting = 3)
# delimiter  = '\t' because the tuples are tab separated.
# quoting  = 3 in order to ignore the double quotes within the reviews.
df.head()
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1

Cleaning the text.

- This includes removing non - significant words like 'the', 'a', etc. and some punc
- Stemming: For the word 'loves' or 'loved', we will only choose 'love'.

To learn more about RegEx, CLICK HERE (https://github.com/sagsshakya/Pandas-Notes/blob/master/Regular%20Expressions/Regular_Expressions.ipynb)

In [3]:

```
import re
review = re.sub('[^a-zA-Z]', ' ' , df['Review'][0], flags = re.IGNORECASE)
# We can use '\W' (non - alphanumeric character or non - word character) instead

#Convert into LowerCase.
review = review.lower()
review
```

```
'wow    loved this place '
```

Removing all the non - significant words.

In [4]:

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\acer\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

In [5]:

```
from nltk.corpus import stopwords

# Splitting the string review using a whitespace.
review = review.split()
```

In [6]:

```
review
```

```
['wow', 'loved', 'this', 'place']
```

Including only significant words in this list.

In [8]:

```python
# The list of stopwords in the English language can be viewed as:
mystopper = set(stopwords.words('english'))
print(mystopper)
```

```
{"hadn't", "hasn't", 'is', 'on', 'down', 'are', 'other', 'at', 'where', "should've", "isn't", 'have', "
'to', 'myself', 'were', 'too', 'wasn', 'than', 'as', 'off', "shouldn't", 'doing', 'y', 'them', 't', 'mu
ith', 'why', 'she', 'most', "needn't", 'all', 'how', 'being', 'ma', 'mightn', 'against', 'itself', 'you
'above', 'you', 'again', 'did', 'because', 'weren', 'herself', "you've", 'or', 'no', 'by', 'through', '
e', 'while', 'am', 'further', 'o', 'for', 'the', 'what', 'under', 'about', 'that', 'her', 'below', 'ain
'our', 'between', 'should', 'shouldn', 'there', 'he', 'so', 'once', 'whom', "it's", 'hasn', 'your', "we
ouldn't", 'his', 'but', 's', "wasn't", "won't", 'some', 'him', 'out', 'a', 'do', 'own', 'this', "aren't
selves', 'its', 'needn', 'couldn', "you'll", 'up', 'i', 'in', 'doesn', 'yourself', 'm', 'nor', 'from',
e', 'isn', 'their', 'just', 'ours', "you'd", 'does', 'until', 'before', "didn't", "haven't", 'haven', '
n', 'was', "you're", 'only', 'ourselves', 'few', 'aren', 'and', 'if', 'into', 'they', 'of', 'now', 'don
ightn't", 're', 'each', 'after', 'more', "don't", 'those', 'here', 'when', "mustn't", "she's"}
```

In [11]:

```python
# Now including only those words which are significant.

review = [ii for ii in review if not ii in mystopper]
review
```

```
['wow', 'loved', 'place']
```

We can exclude the above cell and directly proceed to the stemming process.

Stemming.

- To reduce the complexity of the Sparse Matrix (http://www.btechsmartclass.con
  matrix.html).

In [12]:

```python
from nltk.stem.porter import PorterStemmer as PS
ps = PS()

review = [ps.stem(ii) for ii in review if not ii in mystopper]
review
```

```
['wow', 'love', 'place']
```

Joining the items of the list into a string.

In [13]:

```
review = ' '.join(review)
review
```

```
'wow love place'
```

Using for loop to create processed reviews for the Reviews

In [22]:

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer as PS

ps = PS()
```

In [15]:

```
corpus = []
for jj in range(df.shape[0]):
    # Selecting only the non - alphanumeric characters in the reviews.
    review = re.sub('[^a-zA-Z]', ' ' , df['Review'][jj], flags = re.IGNORECASE)

    # Converting into LowerCase.
    review = review.lower()

    # Splitting the string review using a whitespace.
    review = review.split()

    # The list of stopwords in the English language can be viewed as:
    mystopper = set(stopwords.words('english'))

    # Stemming.
    review = [ps.stem(ii) for ii in review if not ii in mystopper]

    # Joing the list into a string.
    review = ' '.join(review)

    corpus.append(review)
```

In [21]:

```python
df_corpus = pd.DataFrame(corpus)
print(df_corpus.head(10))
df.head(10)
```

```
                                                   0
0                                     wow love place
1                                         crust good
2                                tasti textur nasti
3  stop late may bank holiday rick steve recommen...
4                            select menu great price
5                            get angri want damn pho
6                               honeslti tast fresh
7  potato like rubber could tell made ahead time ...
8                                          fri great
9                                        great touch
```

	Review	Liked
0	Wow... Loved this place.	1
1	Crust is not good.	0
2	Not tasty and the texture was just nasty.	0
3	Stopped by during the late May bank holiday of...	1
4	The selection on the menu was great and so wer...	1
5	Now I am getting angry and I want my damn pho.	0
6	Honeslty it didn't taste THAT fresh.)	0
7	The potatoes were like rubber and you could te...	0
8	The fries were great too.	1
9	A great touch.	1

Creating the Bag of Words Model.

Tokenization:

Taking all the unique words from all the tuples and creating a separate column for e

In [24]:

```python
from sklearn.feature_extraction.text import CountVectorizer
```

- class sklearn.feature_extraction.text.CountVectorizer(input='content', encoding strip_accents=None, lowercase=True, preprocessor=None, tokenizer=None, st token_pattern='(?u)\b\w\w+\b', ngram_range=(1, 1), analyzer='word', max_df=1 max_features=None, vocabulary=None, binary=False, dtype=<class 'numpy.int (https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.tex

In [25]:

```
cv = CountVectorizer()
X = cv.fit_transform(corpus).toarray()
```

In [26]:

```
X
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [27]:

```
X.shape
```

```
(1000, 1565)
```

The rows say that there are 1000 entries from the list corpus.

The columns represent the number of unique words in the list.

We can filter out the features (columns) in order to reduce the sparsity in our matrix max_features.

Doing so, we can obtain highly relevant words as well.

In [28]:

```
cv = CountVectorizer(max_features = 1500)
X = cv.fit_transform(corpus).toarray()
X.shape
```

```
(1000, 1500)
```

Creating dependent variable.

In [29]:

```
y = df.iloc[:,1].values
```

Classification using Naive Bayes.

Train Test Split.

In [37]:

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X,y, test_size = 0.20, random_
```

Scaling the data.

In [38]:

```
from sklearn.preprocessing import StandardScaler as SScale
sc = SScale()
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
```

Fitting the training data into the classifier.

In [31]:

```
from sklearn.naive_bayes import GaussianNB as GNB
classifier = GNB()
classifier.fit(X_train, y_train)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

Making predictions.

In [34]:

```
predictions = classifier.predict(X_test)
```

Confusion Matrix.

In [36]:

```
from sklearn.metrics import confusion_matrix as CM

cm = CM(y_test, predictions)
cm
```

```
array([[81, 16],
       [50, 53]], dtype=int64)
```

Classification Report:

In [33]:

```
from sklearn.metrics import classification_report as CR


print(CR(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.62      0.84      0.71        97
           1       0.77      0.51      0.62       103

    accuracy                           0.67       200
   macro avg       0.69      0.67      0.66       200
weighted avg       0.70      0.67      0.66       200
```

We can use other classification models as well to improve the performance metrics

The End.