Natural Language Process (NLP) Notes

NLP stands for "Natural Language Processing." It is a subfield of artificial intelligence (AI) and linguistics that focuses on the interaction between computers and human language. NLP aims to enable computers to understand, interpret, and generate human language in a way that is both meaningful and useful. This involves a range of tasks, including but not limited to:

1. **Text Understanding:** Extracting meaning and context from text, including tasks like sentiment analysis, entity recognition, and topic modeling.
2. **Language Generation:** Creating coherent and contextually relevant text, such as machine-generated articles, dialogue responses, and poetry.
3. **Machine Translation:** Translating text from one language to another while preserving its meaning.
4. **Speech Recognition:** Converting spoken language into written text, enabling voice commands and transcription services.
5. **Speech Synthesis:** Generating spoken language from written text, also known as text-to-speech conversion.
6. **Language Modeling:** Developing statistical or neural models that predict the next word in a sentence or sequence of words, which forms the basis for many NLP tasks.
7. **Question Answering:** Designing systems that can understand and answer questions posed in natural language, often utilizing large textual datasets.
8. **Text Summarization:** Creating concise and coherent summaries of longer texts, such as news articles or research papers.
9. **Information Extraction:** Identifying and extracting structured information from unstructured text, like relationships between entities and events.
10. **Sentiment Analysis:** Determining the sentiment or emotional tone of a piece of text, often used to gauge public opinion on various topics.
11. **Named Entity Recognition (NER):** Identifying and classifying entities like names of people, organizations, dates, and locations in text.
12. **Part-of-Speech Tagging:** Assigning grammatical labels (such as noun, verb, adjective) to words in a sentence.
13. **Parsing:** Analyzing the grammatical structure of sentences to understand their syntactic relationships.

NLP involves a combination of linguistic analysis, machine learning, and computational methods. The goal is to bridge the gap between human communication and machine understanding, enabling computers to interact with humans in a more natural and intuitive manner.

Short notes

NLP

- Algorithm that deals with text or language related data its called NLP algorithm
- NLP = Language processing algorithm (or) Traditional algorithm + ML algorithm
- NLP = Structured + Numerical data

Steps involved in NLP

Natural Language Processing (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language. The steps involved in NLP can vary depending on the specific task or application you're working on, but here is a general overview of the key steps typically involved in NLP:

1. Text Preprocessing: This step involves preparing the raw text data for analysis. Preprocessing steps can include:

- **Tokenization:** Splitting text into individual words or tokens.
- **Lowercasing:** Converting all text to lowercase to ensure consistency.
- **Removing Punctuation:** Stripping away punctuation marks.
- **Stopword Removal:** Removing common words (stopwords) that don't carry significant meaning.
- **Spell Correction:** Correcting common spelling errors.
- **Lemmatization and Stemming:** Reducing words to their base or root form.

2. Text Representation: Converting text data into a numerical format that can be processed by machine learning algorithms. Common techniques include:

- **Bag-of-Words:** Representing text as a set of word frequencies in a document.
- **TF-IDF (Term Frequency-Inverse Document Frequency):** Assigning weights to words based on their importance in a document relative to their frequency in the entire dataset.

TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a numerical representation used in natural language processing to measure the importance of a term (word) within a document in a collection of documents. It's commonly used for text analysis and information retrieval tasks.

The formula for calculating TF-IDF for a term within a document is as follows:

TF-IDF(term, document) = TF(term, document) * IDF(term, collection)

Where:

1. **TF(term, document)** (Term Frequency) represents the frequency of the term within the specific document. It's calculated as the number of times the term appears in the document divided by the total number of terms in that document.

   TF(term, document) = (Number of times the term appears in the document) / (Total number of terms in the document)
2. **IDF(term, collection)** (Inverse Document Frequency) represents the importance of the term across the entire collection of documents. It's calculated as the logarithm of the total number of documents divided by the number of documents containing the term, with the result inverted.

   IDF(term, collection) = log((Total number of documents) / (Number of documents containing the term))

The TF-IDF value helps to highlight terms that are frequent within a specific document but relatively rare across the entire collection of documents. This way, it can identify important and distinctive terms that may characterize a particular document in the context of the larger corpus.

- **Word Embeddings:** Mapping words to dense vectors in a lower-dimensional space (e.g., Word2Vec, GloVe, FastText).
- **Character-Level Representations:** Treating characters as basic units for analysis.

3. Feature Extraction: Selecting relevant features or patterns from the text data that can be used in machine learning models. This step can involve:

- **N-grams:** Extracting sequences of N words or characters to capture local context.
- **Named Entity Recognition (NER):** Identifying entities like names, dates, locations, etc.
- **Part-of-Speech Tagging:** Assigning parts of speech (e.g., noun, verb, adjective) to words in a sentence.

4. Model Selection: Choosing the appropriate NLP model or algorithm for the specific task you're addressing. This step depends on the nature of the problem, such as text classification, sentiment analysis, machine translation, etc.

- **Classical Models:** Examples include Naive Bayes, Support Vector Machines (SVMs), and Decision Trees.
- **Deep Learning Models:** Such as Recurrent Neural Networks (RNNs), Convolutional Neural Networks (CNNs), and Transformer-based models (e.g., BERT, GPT).

5. Model Training: Training the chosen NLP model on labeled data to learn patterns and make predictions. This step involves selecting appropriate training data, defining evaluation metrics, and tuning model parameters.

6. Evaluation: Assessing the performance of the trained model using evaluation metrics relevant to the specific task, such as accuracy, precision, recall, F1-score, perplexity, etc.

7. Post-processing: Applying any necessary post-processing steps to the model's output, such as summarization, translation, or generating human-readable results.

8. Deployment and Application: Deploying the NLP model to the target application, whether it's a chatbot, search engine, recommendation system, or any other NLP-driven functionality.

Keep in mind that the steps and techniques used can vary based on the specific NLP task and the available data. NLP is a dynamic and evolving field, and staying updated with the latest research and techniques is important for achieving the best results.

Key steps Follow

1) Import Library

2) Import data

3) Clean and compress (Data Preprocessing)

```
1) Sentence Tokenization (Paragraphs to Sentence) (. ! ? ;)
2) Word Tokenization (Sentance to word) (space, _ :)
3) Punctual and Special character removal and Making text lowercase
4) Stop word removal (is, a, an, the, them, couldn't, ....)
5) Lemmatization and Stemming (Extract only the root words from data)
```

4) Exploratory Data Analysis (EDA)

```
1) Generate a Word Cloud by plotting the data.
```

5) Encoding data (Text data to Numerical data)

- TF-IDF (Term Frequency-Inverse Document Frequency)
- Score of words in a particular row = (Number of times words in row / Total number of words in row) * log (Number of rows / Number of rows containing the word in them)

6) Apply Machine Learning

1) Split the data

```
Features (X-axis) (2D Matrix)
Targets (Y - axis) (1D Array)
Train, Test, Split, Random state
```

2) Scaling the data

```
1) Import model
2) Initialize
3) Fit (Learning process)
4) Transform
```

3) Apply Machine learning algorithm

```
1) Import model
2) Initialize
3) Fit (learning process)
4) Predict
```

4) Evaluation matric (Check whether the model is correct or not)

```
1) Regression - The evaluation metric for regression is R^2 between minus infinite to 1
 A higher the R^2 is a better model

2) Classification - The evaluation metric for classification is
    1) Accuracy score [ Higher accuracy is a better model (The value should be near 1) ]
    2) F1 score [ F1 score between 0 (low) to 1 (high), a Higher F1 score
```

Natural Language Process (NLP)

In [1]:

```
# 1) Import Library

# File read
import pandas as pd
import numpy as np

# EDA
from wordcloud import WordCloud
import seaborn as sns
import matplotlib.pyplot as plt

# NLP
import nltk

# NLP ( Sentence Tokenization)
from nltk.tokenize import sent_tokenize

# NLP (Word Tokenization)
from nltk import word_tokenize
from nltk.corpus import wordnet

# NLP (Stopword removal)
from nltk.corpus import stopwords
# nltk.download('stopwords')

# NLP (Lemmatization and Stemming)
from nltk.stem import WordNetLemmatizer
# nltk.download('punkt')
# nltk.download('wordnet')
# nltk.download('omw-1.4')
# nltk.download('averaged_perceptron_tagger')

# Encoding data (Text data to Numerical data)  using TF-IDF (Term Frequency-Inverse Docum
from sklearn.feature_extraction.text import TfidfVectorizer

# Data spliting
from sklearn.model_selection import train_test_split

# Apply Machine learning algorithm
from sklearn.tree import DecisionTreeClassifier

# Evaluation matric
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
```

In [2]:

```
# 2) Import data
# https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews?resou
data = pd.read_csv('IMDB Dataset.csv')
data
```

Out[2]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

In [3]:

```
# 3) Clean and compress (Data Pre processing)
    # 1) Sentence Tokenization (Paragraps to Sentence) (. ! ? ;)

# The data is already in the form of sentences so there is no need
```

In [4]:

```
# 3) Clean and compress (Data Pre processing)
    # 2) Word Tokenization (Sentance to word) (space , _ :)

# import nltk
# from nltk.tokenize import word_tokenize

# # Function to Word Tokenization
# def Word_Tokenization(text):
#     review_word_tokenize = word_tokenize(text)
#     return review_word_tokenize

# # Word Tokenization calling
# data['review'] = [Word_Tokenization(w) for w in data['review']]
# data['review']
```

In [5]:

```
# 3) Clean and compress (Data Pre processing)
    # 3) Punctual and Special character removal
        # i) Remove Punctuation and Replace Space (.!<>{}’,”(/)-\)
        # ii) Making text lower case
data['review']
```

Out[5]:

```
0        One of the other reviewers has mentioned that ...
1        A wonderful little production. <br /><br />The...
2        I thought this was a wonderful way to spend ti...
3        Basically there's a family where a little boy ...
4        Petter Mattei's "Love in the Time of Money" is...
                               ...
49995    I thought this movie did a down right good job...
49996    Bad plot, bad dialogue, bad acting, idiotic di...
49997    I am a Catholic taught in parochial elementary...
49998    I'm going to have to disagree with the previou...
49999    No one expects the Star Trek movies to be high...
Name: review, Length: 50000, dtype: object
```

In [6]:

```
        # i) Remove Punctuation and Replace space (.!<>{}’,”(/)-\)   [From pandas funtion
data['review'] = data['review'].str.replace("[^a-zA-Z0-9]", " ", regex=True)
data['review']
```

Out[6]:

```
0        One of the other reviewers has mentioned that ...
1        A wonderful little production   br    br   The...
2        I thought this was a wonderful way to spend ti...
3        Basically there s a family where a little boy ...
4        Petter Mattei s  Love in the Time of Money  is...
                               ...
49995    I thought this movie did a down right good job...
49996    Bad plot  bad dialogue  bad acting  idiotic di...
49997    I am a Catholic taught in parochial elementary...
49998    I m going to have to disagree with the previou...
49999    No one expects the Star Trek movies to be high...
Name: review, Length: 50000, dtype: object
```

In [7]:

```
        # ii) Making text lower case
data['review'] = [row.lower() for row in data['review']]
data['review']
```

Out[7]:

```
0        one of the other reviewers has mentioned that ...
1        a wonderful little production   br    br   the...
2        i thought this was a wonderful way to spend ti...
3        basically there s a family where a little boy ...
4        petter mattei s  love in the time of money  is...
                              ...
49995    i thought this movie did a down right good job...
49996    bad plot  bad dialogue  bad acting  idiotic di...
49997    i am a catholic taught in parochial elementary...
49998    i m going to have to disagree with the previou...
49999    no one expects the star trek movies to be high...
Name: review, Length: 50000, dtype: object
```

In [8]:

```
# 3) Clean and compress (Data Pre processing)
    # 4) Stop word removall (is, a, an, the, them, could't, ....)

# Import library
import nltk
nltk.download('punkt')
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk import word_tokenize

# Load the list of stopwords
stop_words = stopwords.words('english') # extracting all the stop words in english langua

# Making custom list of words to be removed
add_words = ['movie','br','go','film','ugh','one','make','even','see','movies','get','mak

# Adding to the list of words
stop_words.extend(add_words)

# Function to remove stop words
def remove_stopwords(rev):

    # Tokenize the text into words
    review_tokenized = word_tokenize(rev)

    # Remove stopwords from the list of words
    filtered_words = [i for i in review_tokenized  if i not in stop_words]

    # Reconstruct the filtered text
    rev_new = " ".join(filtered_words)

    return rev_new

# Removing stopwords Funtion calling
data['review'] = [remove_stopwords(r) for r in data['review']]
data['review']
```

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

Out[8]:

```
0        reviewers mentioned watching 1 oz episode hook...
1        wonderful little production filming technique ...
2        thought wonderful way spend hot summer weekend...
3        basically family little boy jake thinks zombie...
4        petter mattei love money visually stunning mr ...
                              ...
49995    thought right job creative original first expe...
49996    bad plot bad dialogue bad acting idiotic direc...
49997    catholic taught parochial elementary schools n...
49998    going disagree previous comment side maltin se...
49999    expects star trek high art fans expect best ep...
Name: review, Length: 50000, dtype: object
```

In [9]:

```
# 3) Clean and compress (Data Pre processing)
    # 5) Lemmatization and Stemming (Extract only the root words from data)

# Import library
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize
from nltk.corpus import wordnet

# Download WordNet data (only need to do this once)
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN  # Default to noun

def perform_lemmatization(text):
    # Tokenize the input text into words
    words = word_tokenize(text)

    # Part-of-speech tagging
    tagged_words = nltk.pos_tag(words)

    # Initialize the WordNet lemmatizer
    lemmatizer = WordNetLemmatizer()

    # Lemmatize each word based on its POS tag
    lemmatized_words = [lemmatizer.lemmatize(word, get_wordnet_pos(pos_tag)) for word, pc

    # Join the lemmatized words back into a sentence
    lemmatized_text = ' '.join(lemmatized_words)

    return lemmatized_text


# Perform lemmatization
data['review'] = data['review'].apply(lambda x: perform_lemmatization(x))

data['review']
```

```
[nltk_data] Downloading package punkt to C:\Users\AVA
[nltk_data]     Computers\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to C:\Users\AVA
[nltk_data]     Computers\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to C:\Users\AVA
[nltk_data]     Computers\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\AVA Computers\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[9]:

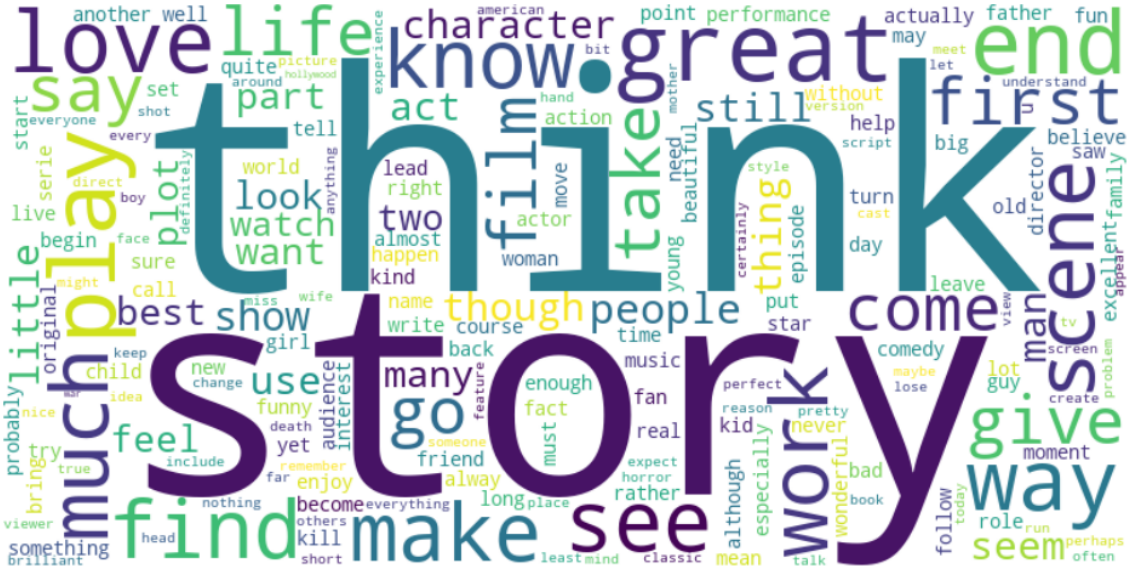
```
0        reviewer mention watch 1 oz episode hook right...
1        wonderful little production film technique una...
2        think wonderful way spend hot summer weekend s...
3        basically family little boy jake think zombie ...
4        petter mattei love money visually stun mr matt...
                               ...
49995    think right job creative original first expect...
49996    bad plot bad dialogue bad act idiotic direct a...
49997    catholic teach parochial elementary school nun...
49998    go disagree previous comment side maltin secon...
49999    expect star trek high art fan expect best epis...
Name: review, Length: 50000, dtype: object
```

In [10]:

```
# 4) Exploratory Data Analysis (EDA)
    # 1) Generate a Word Cloud by plotting the data.
```

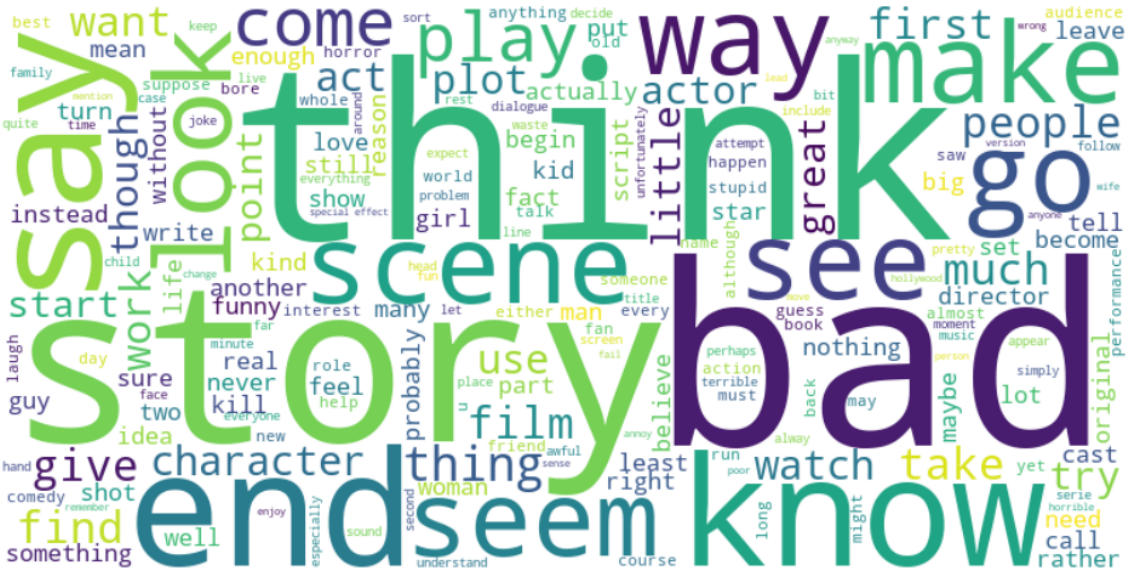
In [11]:

```
# Positive comments

# Import library
from wordcloud import WordCloud
import matplotlib.pyplot as plt

word_cloud = data.loc[data['sentiment'] == 'positive',:]
text = ' '.join([text for text in word_cloud['review']])

# Generate a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud using Matplotlib
plt.figure(figsize=(12, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [12]:

```
# Negative comments

# Import library
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# Sample text data
# text = "Hello world! This is a sample text for generating a word cloud. Word clouds are


word_cloud = data.loc[data['sentiment'] == 'negative',:]
text = ' '.join([text for text in word_cloud['review']])

# Generate a WordCloud object
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

# Display the word cloud using Matplotlib
plt.figure(figsize=(12, 7))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```



In [13]:

```
# 5) Encoding data (Text data to Numerical data)
    # TF-IDF (Term Frequency-Inverse Document Frequency)
    # Score of words in a particular row = (Number of times words in row / Total number o
```

In [14]:

```
# Import library
from sklearn.feature_extraction.text import TfidfVectorizer

# Create a TfidfVectorizer
vectorizer = TfidfVectorizer(max_features=2500)

# Fit and transform the documents
tfidf_matrix = vectorizer.fit_transform(data['review'])

# Convert the TF-IDF matrix to an array
tfidf_array = tfidf_matrix.toarray()

# Get feature names (words)
feature_names = vectorizer.get_feature_names_out()
```

In [15]:

```
vectorizer
```

Out[15]:

```
TfidfVectorizer(max_features=2500)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:

```
tfidf_matrix
```

Out[16]:

```
<50000x2500 sparse matrix of type '<class 'numpy.float64'>'
        with 3319446 stored elements in Compressed Sparse Row format>
```

In [17]:

```
tfidf_array
```

Out[17]:

```
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.]])
```

In [18]:

```
feature_names
```

Out[18]:

```
array(['000', '10', '100', ..., 'youth', 'zero', 'zombie'], dtype=object)
```

In [19]:

```
# Encoding target column
sentiment = data['sentiment'].map({'positive' : 1, 'negative' : 0})
sentiment
```

Out[19]:

```
0        1
1        1
2        1
3        0
4        1
        ..
49995    1
49996    0
49997    0
49998    0
49999    0
Name: sentiment, Length: 50000, dtype: int64
```

In [20]:

```
# 6) Apply Machine Learning
   # 1) Split the data
x = tfidf_array # Features (X - axis)(2D Matrix)
y = sentiment   # Targets  (Y - axis)(1D Array)

# Import library
from sklearn.model_selection import train_test_split

# Split the train and test data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[20]:

```
((40000, 2500), (10000, 2500), (40000,), (10000,))
```

In [21]:

```
# 6) Apply Machine Learning
    # 2) Scaling the data (not mandatory)
```

In [22]:

```
# 6) Apply Machine Learning
    # 3) Apply Machine learning algorithm

        # 1) Import model
from sklearn.tree import DecisionTreeClassifier

        # 2) Initialize
dtc = DecisionTreeClassifier()

        # 3) Fit (learning process)
dtc.fit(x_train, y_train)

        # 4) Predict
y_pred = dtc.predict(x_test)

y_pred
```

Out[22]:

```
array([1, 0, 1, ..., 1, 0, 1], dtype=int64)
```

In [23]:

```
# 6) Apply Machine Learning
    # 4) Evaluation matric (Check whether the model is correct or not)

# Import library
from sklearn.metrics import confusion_matrix, accuracy_score, f1_score

# Calculate accuracy of the model
accuracy = dtc.score(x_test, y_test)
print("Accuracy of the model:", accuracy)

# Confusion matrix
cfm = confusion_matrix(y_test, y_pred)
print('Confusion matrix:\n', cfm)

# Calculate accuracy of y_actual vs y_predict
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Test Accuracy: ", accuracy)
print("Test F1 Score: ", f1)

# Higher accuracy is a better model (The value should near to 1)
# F1 score between 0 (low) to 1 (high)
# Higher F1 score better the model
```

```
Accuracy of the model: 0.72
Confusion matrix:
 [[3592 1369]
 [1431 3608]]
Test Accuracy:  0.72
Test F1 Score:  0.720447284345048
```

In [24]:

```
# 7) Sentiment analysis
    # Most review words

# Assume you have already defined 'feature_names' and 'x_train' (training data) and 'y_tr

# Create an instance of DecisionTreeClassifier
# dtc = DecisionTreeClassifier()

# Fit the DecisionTreeClassifier model on the training data
# dtc.fit(x_train, y_train)

# Get the feature importances from the trained model
importances = dtc.feature_importances_

# Create a dictionary that maps feature importances to feature names
importance_dict = {i: j for i, j in zip(importances, feature_names)}

# Convert the dictionary into a pandas DataFrame
featureImportance = pd.DataFrame(importance_dict.items(), columns=['Importance', 'word'])

# Sort the DataFrame by 'Importance' column in descending order
featureImportance_sorted = featureImportance.sort_values(by='Importance', ascending=False

# Print the sorted feature importances
featureImportance_sorted
```

Out[24]:

	Importance	word
114	0.129906	bad
1412	0.042631	waste
582	0.036581	great
110	0.018876	awful
449	0.017335	excellent
...	...	...
982	0.000033	possess
675	0.000033	insight
1258	0.000033	successful
650	0.000025	hunter
0	0.000000	youth

1467 rows × 2 columns

In [ ]:

```
```