School of Computer Science and Engineering

DIGITAL ASSIGNMENT :- Practice Problem set(Part - 1 + Part-2)

Code: MCSE602P

Course Title: Machine Learning Lab

Faculty: Prof.G.N. Balaji

Name : Nidhi Singh

Register No. : 22MAI0015

1. Create a dataset using an API with Python (Use Web Scrapping/web crawling to create your own dataset) from anyone (discussed in class) of the following application domains.

a. IMDB

b. Flipkart

c. Amazon

d. Twitter

STEP 1 :- IMDB DATASET CREATION

```
import pandas as pd
import requests
from bs4 import BeautifulSoup
import numpy as np
from time import sleep
from random import randint
```

```
headers={'Accept-Language':'en-US,en;q=0.5'}
```

```
page=np.arange(1,1000,100)
```

Code for getting the link of all top 1000 movies of IMDB

```
pages=[]
for x in page:
  pages.append(f"https://www.imdb.com/search/title/?groups=top_1000&sort=user_rating,desc&count=100&start={x}&ref_=adv_nxt")
```

```
links=[]
for x in pages:
  response = requests.get(x)
  soup=BeautifulSoup(response.text,'html.parser')
  for movie in soup.select(".lister-item-header"):
    tag=movie.select_one("a")["href"]
    link=f"https://www.imdb.com{tag}reviews?ref_=tt_urv"
```

```
    link=f"https://www.imdb.com{tag}reviews?ref_=tt_urv"
    links.append(link)
```

```
np.count_nonzero(links)
```

```
1000
```

code for getting all the reviews of top 1000 movies on IMDB

```
reviews = []
for x in links:
  session = requests.Session()
  response = requests.get(x)
  soup = BeautifulSoup(response.text, 'html.parser')
  review_headings = soup.select(".title")
  for s in review_headings:
      review = s.text.strip()
      reviews.append(review)
  load_more = soup.find('div', class_='load-more-data')
  base_url = load_more['data-ajaxurl']
  data_key = load_more['data-key'] if len(review_headings)>25 else ''
  load_more_url = f"https://m.imdb.com{base_url}?ref_=undefined&paginationKey={data_key}"
  while load_more_url:
    response = session.post(load_more_url)
    soup = BeautifulSoup(response.content, 'html.parser')
    review_headings = soup.select(".title")
    for s in review_headings:
      review = s.text.strip()
      reviews.append(review)
    load_more = soup.find('div', class_='load-more-data')
    if not load_more:
      break
    data_key = load_more['data-key']
    load_more_url = f"https://m.imdb.com{base_url}?ref_=undefined&paginationKey={data_key}"
  print(links.index(x))
```

```
0
1
2
3
4
5
6
7
8
```

Numbers of reviews :

```
np.count_nonzero(reviews)
```

```
505677
```

creating dataframe of reviews

```
review_DF=pd.DataFrame({'Review':reviews})
```

Converting dataframe into CSV file dataset

```
review_DF.to_csv('/content/drive/MyDrive/Colab Notebooks/ML_LAB_ASSIGN/IMDB_REVIEW_NOTEBOOKS/IMDB dataset/IMDB_MOVIE_REVIEWS_DATASET_UNCLASSI
```

Labeling a dataset for sentiment analysis involves assigning sentiment labels to the text data in my dataset.

```
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.utils.multiclass import unique_labels
```

```
nltk.download('vader_lexicon')
sid = SentimentIntensityAnalyzer()

def classify_review(review):
  score = sid.polarity_scores(review)
  print(score)
  if score['compound'] >=0:
    return 'positive'
  else:
    return 'negative'
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
file_path='/content/drive/MyDrive/Colab Notebooks/ML_LAB_ASSIGN/IMDB_REVIEW_NOTEBOOKS/IMDB dataset/IMDB_MOVIE_REVIEWS_DATASET_UNCLASSIFIED.cs
df=pd.read_csv(file_path)
```

```
df.head()
```

	Review	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamec
0	Some birds aren't meant to be caged.	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	An incredible movie. One that lives with you.	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	Don't Rent Shawshank.	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	This is How Movies Should Be	NaN	NaN	NaN	NaN	NaN	NaN	Na

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

```
df.head()
```

	Review
0	Some birds aren't meant to be caged.
1	An incredible movie. One that lives with you.
2	Don't Rent Shawshank.
3	This is How Movies Should Be Made
4	A classic piece of unforgettable film-making.

```
df = df.dropna()
df=df[~df.astype(str).apply(lambda x: x.str.isnumeric()).any(axis=1)]
```

```
Sentiments_data=[]
for index, row in df.iterrows():
  sentiment = classify_review(row['Review'])
  Sentiments_data.append(sentiment)
```

```
{'neg': 0.0, 'neu': 0.753, 'pos': 0.247, 'compound': 0.3182}
{'neg': 1.0, 'neu': 0.0, 'pos': 0.0, 'compound': -0.4767}
{'neg': 0.0, 'neu': 0.556, 'pos': 0.444, 'compound': 0.4926}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.423, 'pos': 0.577, 'compound': 0.6249}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.687, 'pos': 0.313, 'compound': 0.6249}
{'neg': 0.546, 'neu': 0.454, 'pos': 0.0, 'compound': -0.3412}
{'neg': 0.825, 'neu': 0.175, 'pos': 0.0, 'compound': -0.5719}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.729, 'neu': 0.271, 'pos': 0.0, 'compound': -0.4005}
{'neg': 0.433, 'neu': 0.382, 'pos': 0.185, 'compound': -0.5994}
{'neg': 0.0, 'neu': 0.471, 'pos': 0.529, 'compound': 0.6696}
{'neg': 0.0, 'neu': 0.185, 'pos': 0.815, 'compound': 0.6588}
{'neg': 0.0, 'neu': 0.215, 'pos': 0.785, 'compound': 0.8074}
{'neg': 0.376, 'neu': 0.624, 'pos': 0.0, 'compound': -0.3412}
{'neg': 0.0, 'neu': 0.508, 'pos': 0.492, 'compound': 0.4404}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.371, 'pos': 0.629, 'compound': 0.7249}
{'neg': 0.0, 'neu': 0.345, 'pos': 0.655, 'compound': 0.2263}
{'neg': 0.0, 'neu': 0.26, 'pos': 0.74, 'compound': 0.6908}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.324, 'neu': 0.676, 'pos': 0.0, 'compound': -0.34}
{'neg': 0.0, 'neu': 0.568, 'pos': 0.432, 'compound': 0.5859}
{'neg': 0.0, 'neu': 0.709, 'pos': 0.291, 'compound': 0.6249}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.159, 'neu': 0.841, 'pos': 0.0, 'compound': -0.2682}
{'neg': 0.0, 'neu': 0.423, 'pos': 0.577, 'compound': 0.6249}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.58, 'pos': 0.42, 'compound': 0.1154}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.468, 'neu': 0.532, 'pos': 0.0, 'compound': -0.6597}
{'neg': 0.722, 'neu': 0.278, 'pos': 0.0, 'compound': -0.3818}
{'neg': 0.372, 'neu': 0.319, 'pos': 0.309, 'compound': -0.1531}
{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.6239}
{'neg': 0.485, 'neu': 0.515, 'pos': 0.0, 'compound': -0.6908}
{'neg': 0.756, 'neu': 0.244, 'pos': 0.0, 'compound': -0.7351}
{'neg': 0.0, 'neu': 0.625, 'pos': 0.375, 'compound': 0.6369}
{'neg': 1.0, 'neu': 0.0, 'pos': 0.0, 'compound': -0.4767}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.256, 'pos': 0.744, 'compound': 0.4404}
{'neg': 0.0, 'neu': 0.672, 'pos': 0.328, 'compound': 0.2382}
{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
{'neg': 0.0, 'neu': 0.385, 'pos': 0.615, 'compound': 0.4939}
{'neg': 0.119, 'neu': 0.518, 'pos': 0.363, 'compound': 0.5888}
{'neg': 0.0, 'neu': 0.263, 'pos': 0.737, 'compound': 0.4215}
{'neg': 0.212, 'neu': 0.323, 'pos': 0.465, 'compound': 0.4805}
{'neg': 0.427, 'neu': 0.104, 'pos': 0.468, 'compound': 0.1007}
{'neg': 0.538, 'neu': 0.462, 'pos': 0.0, 'compound': -0.5423}
{'neg': 0.0, 'neu': 0.556, 'pos': 0.444, 'compound': 0.7506}
{'neg': 0.4, 'neu': 0.353, 'pos': 0.247, 'compound': -0.319}
{'neg': 0.655, 'neu': 0.345, 'pos': 0.0, 'compound': -0.5859}
{'neg': 0.153, 'neu': 0.6, 'pos': 0.246, 'compound': 0.1997}
{'neg': 0.0, 'neu': 0.0, 'pos': 1.0, 'compound': 0.5859}
{'neg': 0.138, 'neu': 0.862, 'pos': 0.0, 'compound': -0.1531}
```

```
np.count_nonzero(Sentiments_data)
```

```
505423
```

```
df['Sentiment_Labels']=Sentiments_data
```

```
df.head()
```

	Review	Sentiment_Labels
0	Some birds aren't meant to be caged.	positive
1	An incredible movie. One that lives with you.	positive
2	Don't Rent Shawshank.	positive
3	This is How Movies Should Be Made	positive
4	A classic piece of unforgettable film-making.	positive

```
df
```

	Review	Sentiment_Labels
0	Some birds aren't meant to be caged.	positive
1	An incredible movie. One that lives with you.	positive
2	Don't Rent Shawshank.	positive
3	This is How Movies Should Be Made	positive
4	A classic piece of unforgettable film-making.	positive
...	...	...
505671	Very good technically, but depressing.	negative
505672	Pointless nihilistic bullshit	negative
505673	good movie, but wished there had been less foc...	positive
505674	Splendid	positive
505675	Mystic River shows that our "present" is affec...	negative

505423 rows × 2 columns

classified labeled dataset

```
df.to_csv('/content/drive/MyDrive/Colab Notebooks/ML_LAB_ASSIGN/IMDB_REVIEW_NOTEBOOKS/classified_IMDB TOP 1000 Movies_review.csv')
```

2. Apply pre-processing techniques such as

• Stopwords Removal

• URL Removal

• Stemming

• Lemmatization

• Convert Numbers to Words

• Tokenization

• Unigram/Bigram Approach

etc., Intermediate Result: Show Pre-processed data in each

Preprocessing a dataset for sentiment analysis involves cleaning and transforming the raw text data into a format that can be used effectively by machine learning models.

```
import pandas as pd
import numpy as np
import nltk

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.utils.multiclass import unique_labels


from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import style
```

```
style.use('ggplot')
import re
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_auc_score, roc_curve, confusion_matrix, precision_score, recall_score
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.utils.multiclass import unique_labels
```

```
pip install num2words
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: num2words in /usr/local/lib/python3.10/dist-packages (0.5.12)
Requirement already satisfied: docopt>=0.6.2 in /usr/local/lib/python3.10/dist-packages (from num2words) (0.6.2)
```

```
import nltk
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from num2words import num2words
from nltk.stem import WordNetLemmatizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
from nltk.util import ngrams
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
# Step 1: Load the IMDb movie review dataset
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ML_LAB_ASSIGN/IMDB_REVIEW_NOTEBOOKS/IMDB_MAIN_PROJECT_CODE_FILE/classified_IMDB TOP
```

```
df.head()
```

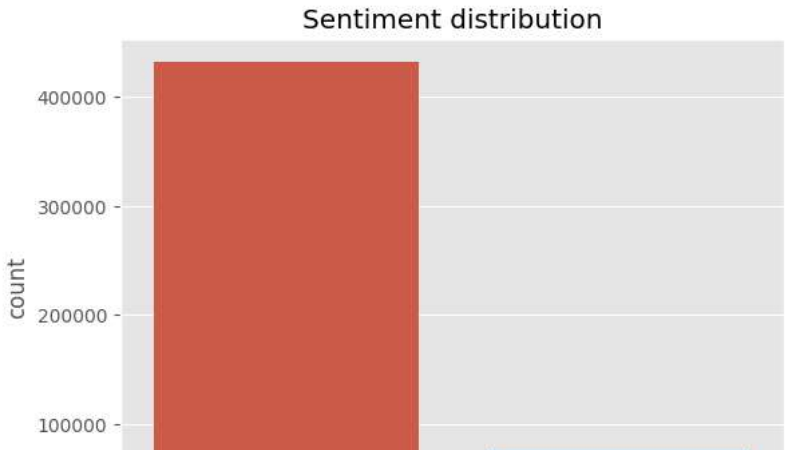
	Unnamed: 0	Review	Sentiment_Labels
0	0	Some birds aren't meant to be caged.	positive
1	1	An incredible movie. One that lives with you.	positive
2	2	Don't Rent Shawshank.	positive
3	3	This is How Movies Should Be Made	positive
4	4	A classic piece of unforgettable film-making.	positive

```
df = df.iloc[:, 1:3]
df.head()
```

	Review	Sentiment_Labels
0	Some birds aren't meant to be caged.	positive
1	An incredible movie. One that lives with you.	positive
2	Don't Rent Shawshank.	positive
3	This is How Movies Should Be Made	positive
4	A classic piece of unforgettable film-making.	positive

```
sns.countplot(x='Sentiment_Labels', data=df)
plt.title("Sentiment distribution")
```

```
Text(0.5, 1.0, 'Sentiment distribution')
```




```
for i in range(5):
    print("Review: ", [i])
    print(df['Review'].iloc[i], "\n")
    print("Sentiment: ", df['Sentiment_Labels'].iloc[i], "\n\n")
```

```
Review:  [0]
Some birds aren't meant to be caged.

Sentiment:  positive


Review:  [1]
An incredible movie. One that lives with you.

Sentiment:  positive


Review:  [2]
Don't Rent Shawshank.

Sentiment:  positive


Review:  [3]
This is How Movies Should Be Made

Sentiment:  positive


Review:  [4]
A classic piece of unforgettable film-making.

Sentiment:  positive
```

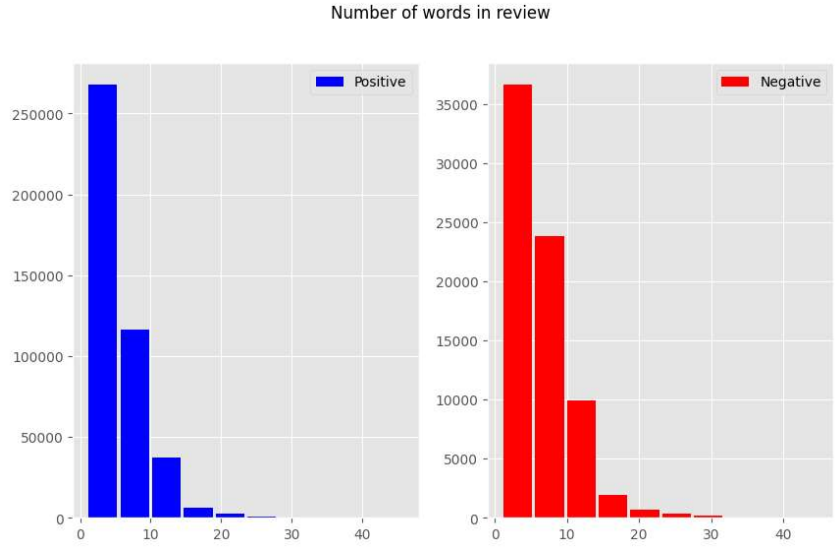
```
def no_of_words(text):
    words= text.split()
    word_count = len(words)
    return word_count


df['word_count'] = df['Review'].apply(no_of_words)


df.head()
```

	Review	Sentiment_Labels	word_count
0	Some birds aren't meant to be caged.	positive	7
1	An incredible movie. One that lives with you.	positive	8
2	Don't Rent Shawshank.	positive	3
3	This is How Movies Should Be Made	positive	7
4	A classic piece of unforgettable film-making.	positive	6

```
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].hist(df[df['Sentiment_Labels'] == 'positive']['word_count'], label='Positive', color='blue', rwidth=0.9);
ax[0].legend(loc='upper right');
ax[1].hist(df[df['Sentiment_Labels'] == 'negative']['word_count'], label='Negative', color='red', rwidth=0.9);
ax[1].legend(loc='upper right');
fig.suptitle("Number of words in review")
plt.show()
```




```
fig, ax = plt.subplots(1,2, figsize=(10,6))
ax[0].hist(df[df['Sentiment_Labels'] == 'positive']['Review'].str.len(), label='Positive', color='blue', rwidth=0.9);
ax[0].legend(loc='upper right');
ax[1].hist(df[df['Sentiment_Labels'] == 'negative']['Review'].str.len(), label='Negative', color='red', rwidth=0.9);
ax[1].legend(loc='upper right');
fig.suptitle("Number of words in review")
plt.show()
```

```
df.Sentiment_Labels.replace("positive", 1, inplace=True)
df.Sentiment_Labels.replace("negative", 0, inplace=True)
```

```
df.head()
```

	Review	Sentiment_Labels	word_count
0	Some birds aren't meant to be caged.	1	7
1	An incredible movie. One that lives with you.	1	8
2	Don't Rent Shawshank.	1	3
3	This is How Movies Should Be Made	1	7
4	A classic piece of unforgettable film-making.	1	6

2. Apply pre-processing techniques such as

Stopwords Removal

URL Removal

Stemming

Lemmatization

Convert Numbers to Words

Tokenization

Unigram/Bigram Approach

etc.

```
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib import style
style.use('ggplot')
import re
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
```

```
import nltk
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
nltk.download('punkt')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

Clean the text data

```
def clean_text(text):
    text = text.lower()
    text = re.sub('', '', text)
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)
    text = re.sub(r'\@w+|\#','', text)
```

```
    text = re.sub(r'[^\w\s]','', text)
    text = re.sub('<.*?>', '', text)  # Remove HTML tags
    text = re.sub('[^a-zA-Z]', ' ', text)  # Remove non-alphabetic characters
    text = re.sub('', '', text)
    text_tokens = word_tokenize(text)
    filtered_text = [w for w in text_tokens if not w in stop_words]
    return " ".join(filtered_text)


df.Review = df['Review'].apply(clean_text)


df
```

	Review	Sentiment_Labels	word_count
0	birds arent meant caged	1	7
1	incredible movie one lives	1	8
2	dont rent shawshank	1	3
3	movies made	1	7
4	classic piece unforgettable filmmaking	1	6
...	...	...	...
505418	good technically depressing	0	5
505419	pointless nihilistic bullshit	0	3
505420	good movie wished less focus criminal investig...	1	22
505421	splendid	1	1
505422	mystic river shows present affected past	0	11

505423 rows × 3 columns

```
df.word_count= df['Review'].apply(no_of_words)

df
```

	Review	Sentiment_Labels	word_count
0	birds arent meant caged	1	4
1	incredible movie one lives	1	4
2	dont rent shawshank	1	3
3	movies made	1	2
4	classic piece unforgettable filmmaking	1	4
...	...	...	...
505418	good technically depressing	0	3
505419	pointless nihilistic bullshit	0	3
505420	good movie wished less focus criminal investig...	1	12
505421	splendid	1	1
505422	mystic river shows present affected past	0	6

505423 rows × 3 columns

Tokenization of the reviews text

```
df.Review= df['Review'].apply(word_tokenize)

df
```

	Review	Sentiment_Labels	word_count
0	[birds, arent, meant, caged]	1	4
1	[incredible, movie, one, lives]	1	4
2	[dont, rent, shawshank]	1	3
3	[movies, made]	1	2
4	[classic, piece, unforgettable, filmmaking]	1	4
...	...	...	...
505418	[good, technically, depressing]	0	3
505419	[pointless, nihilistic, bullshit]	0	3
505420	[good, movie, wished, less, focus, criminal, i...	1	12
505421	[splendid]	1	1
505422	[mystic, river, shows, present, affected, past]	0	6

Perform stemming

```
stemmer = PorterStemmer()
df['Review'] = df['Review'].apply(lambda x: [stemmer.stem(token) for token in x])
# The preprocessed data with stemming is now available in the 'Reviews' column of the DataFrame.
df
```

	Review	Sentiment_Labels	word_count
0	[birds, arent, meant, caged]	1	4
1	[incredible, movie, one, lives]	1	4
2	[dont, rent, shawshank]	1	3
3	[movies, made]	1	2
4	[classic, piece, unforgettable, filmmaking]	1	4
...	...	...	...
505418	[good, technically, depressing]	0	3
505419	[pointless, nihilistic, bullshit]	0	3
505420	[good, movie, wished, less, focus, criminal, i...	1	12
505421	[splendid]	1	1
505422	[mystic, river, shows, present, affected, past]	0	6

505423 rows × 3 columns

Remove stop words

```
stop_words = set(stopwords.words('english'))
df.Review = df['Review'].apply(lambda x: [token for token in x if token.lower() not in stop_words])
# The preprocessed data is now available in the 'Reviews' column of the DataFrame.
df
```

	Review	Sentiment_Labels	word_count
0	[birds, arent, meant, caged]	1	4
1	[incredible, movie, one, lives]	1	4
2	[dont, rent, shawshank]	1	3
3	[movies, made]	1	2

```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

Perform lemmatization

```
lemmatizer = WordNetLemmatizer()
df['Review'] = df['Review'].apply(lambda x: [lemmatizer.lemmatize(token) for token in x])

# The preprocessed data with lemmatization is now available in the 'Reviews' column of the DataFrame.
```

```
df
```

	Review	Sentiment_Labels	word_count
0	[bird, arent, meant, caged]	1	4
1	[incredible, movie, one, life]	1	4
2	[dont, rent, shawshank]	1	3
3	[movie, made]	1	2
4	[classic, piece, unforgettable, filmmaking]	1	4
...	...	...	...
505418	[good, technically, depressing]	0	3
505419	[pointless, nihilistic, bullshit]	0	3
505420	[good, movie, wished, le, focus, criminal, inv...	1	12
505421	[splendid]	1	1
505422	[mystic, river, show, present, affected, past]	0	6

505423 rows × 3 columns

Convert numbers to words

```
def convert_numbers_to_words(tokens):
    converted_tokens = []
    for token in tokens:
        if token.isdigit():
            converted_token = num2words(int(token))
            converted_tokens.append(converted_token)
        else:
            converted_tokens.append(token)
    return converted_tokens


df['Review'] = df['Review'].apply(convert_numbers_to_words)
# The preprocessed data with converted numbers to words is now available in the 'Reviews' column of the DataFrame.
df
```

	Review	Sentiment_Labels	word_count
0	[bird, arent, meant, caged]	1	4
1	[incredible, movie, one, life]	1	4
2	[dont, rent, shawshank]	1	3
3	[movie, made]	1	2
4	[classic, piece, unforgettable, filmmaking]	1	4
...	...	...	...
505418	[good, technically, depressing]	0	3
505419	[pointless, nihilistic, bullshit]	0	3

Apply the Unigram/Bigram approach

505422	[mystic, river, show, present, affected, past]	0	6

```
def generate_ngrams(tokens, n):
    ngram_list = list(ngrams(tokens, n))
    ngrams_joined = [' '.join(ngram) for ngram in ngram_list]
    return ngrams_joined

df['Review'] = df['Review'].apply(lambda x: generate_ngrams(x, 1))  # Generate unigrams
# The preprocessed data with unigrams and bigrams is now available in the 'unigrams' and 'bigrams' columns of the DataFrame.
```

```
df
```

	Review	Sentiment_Labels	word_count
0	[bird, arent, meant, caged]	1	4
1	[incredible, movie, one, life]	1	4
2	[dont, rent, shawshank]	1	3
3	[movie, made]	1	2
4	[classic, piece, unforgettable, filmmaking]	1	4
...	...	...	...
505418	[good, technically, depressing]	0	3
505419	[pointless, nihilistic, bullshit]	0	3
505420	[good, movie, wished, le, focus, criminal, inv...	1	12
505421	[splendid]	1	1
505422	[mystic, river, show, present, affected, past]	0	6

505423 rows × 3 columns

```
import pandas as pd
df.Review = df['Review'].apply(lambda x: ' '.join(x))
# Print the DataFrame
print(df)
```

```
                                                Review  Sentiment_Labels  \
0                                bird arent meant caged                 1
1                              incredible movie one life                1
2                                   dont rent shawshank                 1
3                                            movie made                 1
4                  classic piece unforgettable filmmaking                1
...                                                  ...               ...
505418                         good technically depressing                0
505419                       pointless nihilistic bullshit                0
505420  good movie wished le focus criminal investigat...                1
505421                                          splendid                 1
505422            mystic river show present affected past                0

        word_count
0                4
1                4
2                3
3                2
```

```
4               4
...           ...
505418          3
505419          3
505420         12
505421          1
505422          6

[505423 rows x 3 columns]
```

Removal of dublicate Reviews

```
duplicated_count = df.duplicated().sum()
print("Number of duplicate entries: ", duplicated_count)
```

```
Number of duplicate entries:  178166
```

```
df = df.drop_duplicates('Review')
```

```
duplicated_count = df.duplicated().sum()
print("Number of duplicate entries: ", duplicated_count)
```

```
Number of duplicate entries:  0
```

Count the number of words into the reviews

```
df.word_count = df['Review'].apply(no_of_words)
df.head()
```

```
<ipython-input-149-7c7c6fd12498>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
  df.word_count = df['Review'].apply(no_of_words)
```

	Review	Sentiment_Labels	word_count
0	bird arent meant caged	1	4
1	incredible movie one life	1	4
2	dont rent shawshank	1	3
3	movie made	1	2
4	classic piece unforgettable filmmaking	1	4

```
pos_reviews = df[df.Sentiment_Labels ==1]
pos_reviews.head()
```

	Review	Sentiment_Labels	word_count
0	bird arent meant caged	1	4
1	incredible movie one life	1	4
2	dont rent shawshank	1	3
3	movie made	1	2
4	classic piece unforgettable filmmaking	1	4

```
text = ' '.join([word for word in pos_reviews['Review']])
plt.figure(figsize=(20,15), facecolor='None')
wordcloud = WordCloud(max_words=500, width=1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Most frequent words in positive reviews', fontsize = 19)
plt.show()
```

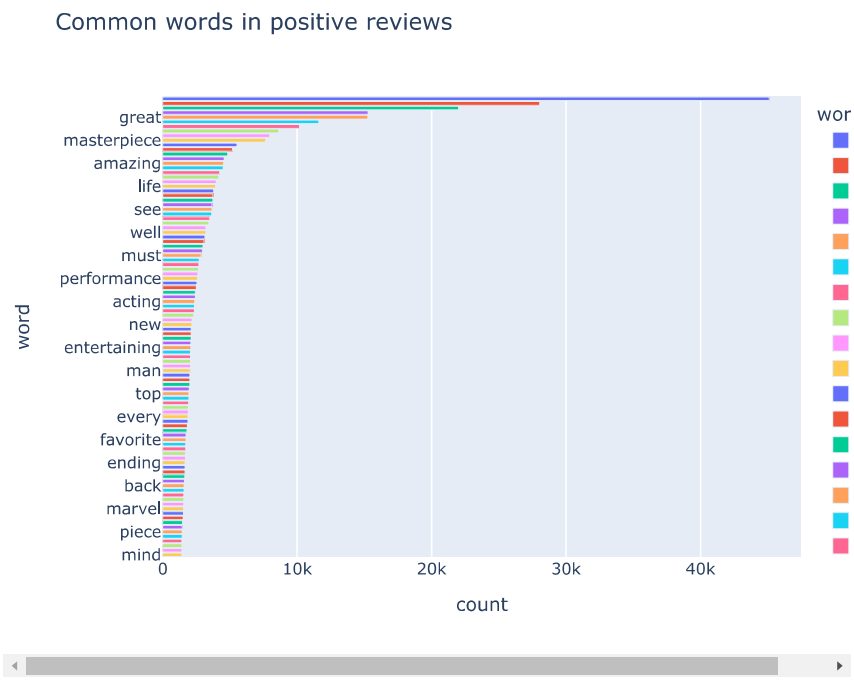

```
from collections import Counter
count = Counter()
for text in pos_reviews['Review'].values:
    for word in text.split():
        count[word] +=1
count.most_common(1000)
```

```
 ('depp', 169),
 ('within', 168),
 ('trying', 168),
 ('dance', 168),
 ('heavy', 168),
 ('video', 168),
 ('late', 167),
 ('red', 167),
 ('essential', 167),
 ('filmmaker', 167),
 ('element', 167),
 ('step', 167),
 ('odyssey', 167),
 ('english', 166),
 ('especially', 166),
 ('notch', 166),
 ('citizen', 166),
 ('charlie', 166),
 ('mel', 166),
 ('recommended', 165),
 ('golden', 165),
 ('blend', 165),
 ('becomes', 165),
 ('country', 165),
 ('private', 165),
 ('aint', 164),
 ('four', 164),
 ('impossible', 164),
 ('otherwise', 164)]
```

```
pos_words = pd.DataFrame(count.most_common(100))
pos_words.columns = ['word', 'count']
pos_words.head()
```

	word	count
0	movie	45120
1	film	28052
2	best	22007
3	one	15278
4	great	15258

```
px.bar(pos_words, x='count', y='word', title='Common words in positive reviews', color='word')
```

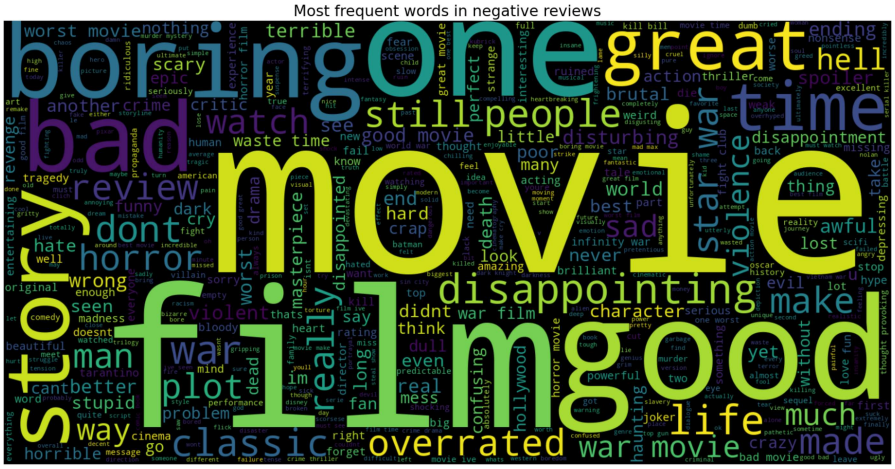



```
neg_reviews = df[df.Sentiment_Labels == 0]
neg_reviews.head()
```

	Review	Sentiment_Labels	word_count
10	alltime prison film classic	0	4
16	go prison learn crook	0	4
18	im convicted murderer provides sound financial...	0	7
20	never give hope	0	3
23	shawshank redemption prison film redeemed quality	0	6

```
text = ' '.join([word for word in neg_reviews['Review']])
plt.figure(figsize=(20,15), facecolor='None')
wordcloud = WordCloud(max_words=500, width=1600, height=800).generate(text)
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.title('Most frequent words in negative reviews', fontsize = 19)
plt.show()
```

Most frequent words in negative reviews




```
count = Counter()
for text in neg_reviews['Review'].values:
    for word in text.split():
        count[word] += 1
count.most_common(1000)
```

```
 ('thrilling', 44),
 ('controversial', 44),
 ('self', 44),
 ('easily', 44),
 ('biased', 44),
 ('german', 44),
 ('directing', 44),
 ('language', 44),
 ('among', 44),
 ('grave', 44),
 ('sex', 44),
 ('south', 44),
 ('monster', 44),
 ('loud', 44),
 ('police', 44),
 ('paced', 44),
 ('extraordinary', 43),
 ('impressed', 43),
 ('technically', 43),
 ('brings', 43),
 ('living', 43),
 ('travesty', 43),
 ('deal', 43),
 ('run', 43),
 ('crude', 43),
 ('st', 43),
 ('guess', 43),
 ('irritating', 43),
 ('wtf', 43),
 ('otherwise', 43),
 ('exceptional', 43),
 ('lesson', 43),
 ('gave', 43),
 ('cost', 43),
 ('falling', 43),
 ('premise', 43),
 ('travel', 43),
 ('murderer', 42),
 ('happy', 42),
 ('trust', 42),
 ('narrative', 42)]
```

```
neg_words = pd.DataFrame(count.most_common(1000))
neg_words.columns = ['word', 'count']
neg_words.head()
```

	word	count
0	movie	8040
1	film	4599
2	good	3041
3	war	2898
4	one	2131

```
px.bar(neg_words, x='count', y='word', title='Common words in negative reviews', color='word')
```

Common words in negative reviews


```
X = df['Review']
Y = df['Sentiment_Labels']
```

Preprocessed dataset

```
df.to_csv('/content/drive/MyDrive/Colab Notebooks/ML_LAB_ASSIGN/IMDB_REVIEW_NOTEBOOKS/IMDB_MAIN_PROJECT_CODE_FILE/processed_classified_IMDB T
```

3. Apply feature selection algorithms to extract the predominant features.

```
review_dataset = df.iloc[:, 0:2]
review_dataset.columns = ['review', 'label']
review_dataset
```

	review	label
0	bird arent meant caged	1
1	incredible movie one life	1
2	dont rent shawshank	1
3	movie made	1
4	classic piece unforgettable filmmaking	1
...	...	...
505417	id love worship temple sean devine	1
505418	good technically depressing	0
505419	pointless nihilistic bullshit	0
505420	good movie wished le focus criminal investigat...	1
505422	mystic river show present affected past	0

325831 rows × 2 columns

```
review_dataset.label.replace(1, "positive", inplace=True)
review_dataset.label.replace(0, "negative", inplace=True)
review_dataset
```

	review	label
0	bird arent meant caged	positive
1	incredible movie one life	positive
2	dont rent shawshank	positive
3	movie made	positive
4	classic piece unforgettable filmmaking	positive
...	...	...
505417	id love worship temple sean devine	positive
505418	good technically depressing	negative
505419	pointless nihilistic bullshit	negative
505420	good movie wished le focus criminal investigat...	positive
505422	mystic river show present affected past	negative

325831 rows × 2 columns

```
review_dataset.groupby('label').count()
```

label	review
negative	56854
positive	268977

```
review_dataset['label'].value_counts(normalize=True)
```

```
positive    0.825511
negative    0.174489
Name: label, dtype: float64
```

```
#splitting into training and testing

X_train, X_test, Y_train, Y_test = train_test_split(review_dataset['review'],
                                                    review_dataset['label'],
                                                    random_state = 0)
```

```
#feature extraction: ngram
vectorizer = CountVectorizer(ngram_range = (1, 2)).fit(X_train)
X_train_vectorized = vectorizer.transform(X_train)

X_train_vectorized.toarray().shape
```

```
(244373, 439157)
```

4. Use Classification algorithms for classification such as

a. Naive Bayes

b. Multinomial Naive Bayes

c. SVM

d. Random Forest

```
#creating multinomial naive bayes

review_predictor = MultinomialNB(alpha = 0.1)
review_predictor.fit(X_train_vectorized, Y_train)
```

```
MultinomialNB
MultinomialNB(alpha=0.1)
```

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.

```
#using test dataset

predictions = review_predictor.predict(vectorizer.transform(X_test))
print("Accuracy: ", 100 * sum(predictions == Y_test) / len(predictions), "%")
```

```
Accuracy:  89.54430504063444 %
```

```
#using real life examples

review_predictor.predict(vectorizer.transform(
    [
        "Good Movie",
        "Worth to watch",
        "Worst Movie ever",
        "Waste of time and Money"
    ]
))
```

```
array(['positive', 'positive', 'negative', 'negative'], dtype='<U8')
```

5. Interpret the result

a. Print confusion matrix

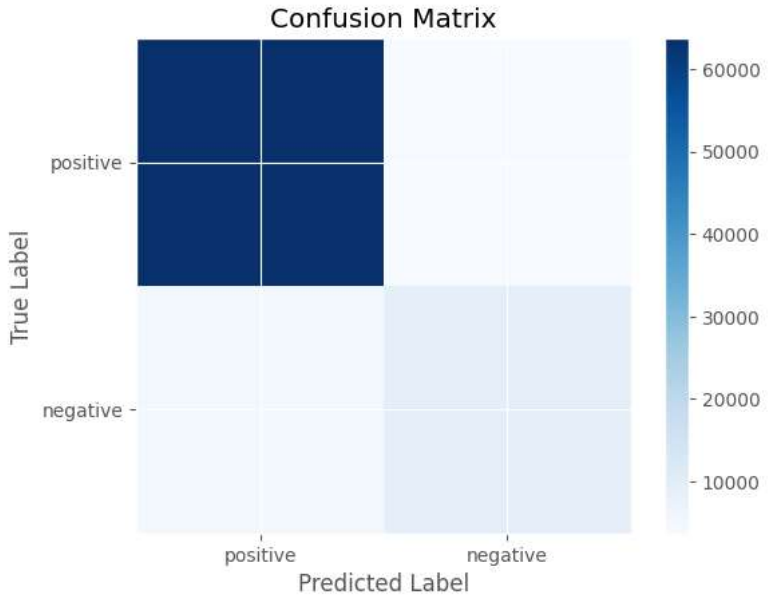
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


# Create a confusion matrix
cm = confusion_matrix(Y_test, predictions, labels=["positive", "negative"])

# Print the confusion matrix
print("Confusion Matrix:")
print(cm)

# Visualize the confusion matrix
plt.imshow(cm, interpolation="nearest", cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.colorbar()
tick_marks = np.arange(len(["positive", "negative"]))
plt.xticks(tick_marks, ["positive", "negative"])
plt.yticks(tick_marks, ["positive", "negative"])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Confusion Matrix:
[[63743  3705]
 [ 4812  9198]]
```



✓ 2s completed at 9:32 PM