PROJECT

Topic : IMBD dataset analysis.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
import pandas as pd
import numpy as np
import time
```

In [3]:
```
import re
import nltk
from nltk.tokenize import RegexpTokenizer, word_tokenize
from nltk.corpus import stopwords
import string
from nltk.stem import PorterStemmer, WordNetLemmatizer
```

Reading the Csv fiel and loading it to pandas dataframe :

In [4]:
```
imdb = pd.read_csv("IMDB_dataset.csv")
```

In [5]:
```
imdb.sample(10)
```

Out[5]:

	review	sentiment
3278	Though a fan of shock and gore, I found this m...	negative
8462	Terrible movie. Nuff Said. These Li...	negative
11676	I don't understand why people would praise thi...	negative
7047	First of all, I'd like to say that I really en...	negative
17770	In complete contrast to the previous correspon...	positive
4471	I'm afraid I only stayed to watch the first ho...	negative
19081	Artistically speaking, this is a beautiful mov...	positive
10542	I know a lot of people like this show and i ap...	negative
1832	Spoilers of both this and The Matrix follow.<b...	negative
7574	***SPOILERS*** ***SPOILERS*** Continued...<br ...	negative

Performing the EDA of the Data :

In [6]:
```
imdb.shape
```

Out[6]:
```
(25000, 2)
```

- We have 25000 rows and 2 columns.

In [7]:
```
imdb.isnull().sum()
```

Out[7]:
```
review       0
sentiment    0
dtype: int64
```

- No null value present in both the columns.

Lowercase All The Text

In [8]:
```
imdb['review'] = imdb['review'].str.lower()
imdb.sample(10)
```

Out[8]:

	review	sentiment
2669	i really seldom give either one or ten stars t...	negative
103	this film is well cast, often silly and always...	positive
8367	anna lives with her family in a new housing es...	negative
1119	this was one of my favorite series when i was ...	positive
22038	for die-hard judy garland fans only. there are...	negative
444	herculis puaro is, in general, a well establis...	negative
4618	ruggero deodato is often credited for inventin...	negative
487	possible spoilers, perhaps. i must say that "c...	negative
4842	at the start, this one is from england, so, of...	positive
5617	what a joke. i am watching it on channel 1 and...	negative

- Converting the all uppercase words and letters to lowercase, that can be observed in above output.

Preprocess Text Data(Remove punctuation, Perform Tokenization, Remove stopwords and Lemmatize/Stem)

In [9]:
```
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/bhavinpatel/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/bhavinpatel/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/bhavinpatel/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[9]: True

Defining the Stop words and punctuation symbols.

In [10]:
```
stop_words = set(stopwords.words('english'))
#punctuation_symbols = set(string.punctuation)
punctuation_symbols = {'.', ',', ';', ':', '?', '!', '-', '_', '(', ')', '[', ']', '{', '}', '/', '\\', '"', "'", "
print(punctuation_symbols, '\n')
print(stop_words)
```

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

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

- Above output displays the stopwords and punctuation symbols that we need to remove from the review data.

In [11]:
```
imdb_2 = imdb.copy()
```

- Creating the copy of dataframe to process all the operations.

def count_punctuation(text): return sum(text.count(p) for p in string.punctuation) # Apply the function to the DataFrame column imdb_2['punctuation_count'] = imdb['review'].apply(count_punctuation) # Total punctuation count total_punctuation_count = imdb_2['punctuation_count'].sum() print("Total punctuation count:", total_punctuation_count)

Counting the punctuation and stopwords :

In [12]:
```
imdb_2.head()
```

Out[12]:

	review	sentiment
0	i thought this was a wonderful way to spend ti...	positive
1	probably my all-time favorite movie, a story o...	positive
2	i sure would like to see a resurrection of a u...	positive
3	this show was an amazing, fresh & innovative i...	negative
4	encouraged by the positive comments about this...	negative

Def function for counting the punctuation and stopwords present in data

In [13]:
```
def count_punct(text):
    punctuation_count = sum(1 for char in text if char in punctuation_symbols)
    return punctuation_count

def count_stopwords(text):
    ret_tokenizer = RegexpTokenizer(r'\w+')
    tokens = ret_tokenizer.tokenize(text)

    stop_words_count = sum(1 for word in tokens if word.lower() in stopwords.words('english'))
    return stop_words_count
```

In [14]:
```
imdb_2['punctuation_count'] = imdb['review'].apply(count_punct)


total_punctuation_count = imdb_2['punctuation_count'].sum()

print("Total punctuation count:", total_punctuation_count)
```

```
Total punctuation count: 1313554
```

- There are total 1313554 punctuation present in the text data.

In [15]:
```
imdb_2['Stopword_count'] = imdb['review'].apply(count_stopwords)

total_stopwords_count = imdb_2['Stopword_count'].sum()

print("Total Stopwords count:", total_stopwords_count)
```

```
Total Stopwords count: 2896187
```

- There are total 2896187 stopwords present in the text data.

In [16]:
```
imdb_2.head()
```

Out[16]:

	review	sentiment	punctuation_count	Stopword_count
0	i thought this was a wonderful way to spend ti...	positive	40	83
1	probably my all-time favorite movie, a story o...	positive	28	69
2	i sure would like to see a resurrection of a u...	positive	12	86
3	this show was an amazing, fresh & innovative i...	negative	33	96
4	encouraged by the positive comments about this...	negative	31	64

- Added two columns punctuation_count and stopwords_count to analysis the count of punctuation and stopwords for each row.

Removing the punctuation :

Def function for removing the punctuation :

In [17]:
```
def remove_punctuation(text):
    for symbol in punctuation_symbols:
        text = text.replace(symbol, '')
    return text

imdb_2['new_review'] = imdb_2['review'].apply(remove_punctuation)
```

Counting the punctuation after removing it

In [18]:
```
imdb_2['after_removing_punctuation_count'] = imdb_2['new_review'].apply(count_punct)

total_after_removing_punctuation_count = imdb_2['after_removing_punctuation_count'].sum()

print("Total punctuation count after removing the punctuation:", total_after_removing_punctuation_count)
```

```
Total punctuation count after removing the punctuation: 0
```

- The output displays the count of punctuation after removing it means we have successfully removed the punctuation.

Example (Before and after removing the punctuations):

imdb['review'][3]imdb_2['review'][3]

Tokenization and Removing the stopwords :

In [19]:
```
def tokenize_and_remove_stopwords(text):
    tokenizer = RegexpTokenizer(r'\w+')
    tokens = tokenizer.tokenize(text)

    stop_words = set(stopwords.words('english'))
    filtered_tokens = [word for word in tokens if word.lower() not in stop_words]

    return filtered_tokens
```

In [20]:
```
imdb_2['new_review'] = imdb_2['new_review'].apply(tokenize_and_remove_stopwords)
```

In [21]:
```
def after_count_stopwords(text):

    tokens = text

    stop_words_count = sum(1 for word in tokens if word.lower() in stopwords.words('english'))
    return stop_words_count
```

In [22]:
```
imdb_2['after_Stopword_count'] = imdb_2['new_review'].apply(after_count_stopwords)

after_stopwords_count = imdb_2['after_Stopword_count'].sum()

print("Total Stopwords count after tokenizing and removing the stopwords:", after_stopwords_count)
```

```
Total Stopwords count after tokenizing and removing the stopwords: 0
```

- The output displays the count of stopwords after removing it means we have successfully removed all the stopwords.

In [23]:
```
imdb_2.head(10)
```

Out[23]:

	review	sentiment	punctuation_count	Stopword_count	new_review	after_removing_punctuation_count	after_Stopword_c
0	i thought this was a wonderful way to spend ti...	positive	40	83	[thought, wonderful, way, spend, time, hot, su...	0	
1	probably my all-time favorite movie, a story o...	positive	28	69	[probably, alltime, favorite, movie, story, se...	0	
2	i sure would like to see a resurrection of a u...	positive	12	86	[sure, would, like, see, resurrection, dated, ...	0	
3	this show was an amazing, fresh & innovative i...	negative	33	96	[show, amazing, fresh, innovative, idea, 70s, ...	0	
4	encouraged by the positive comments about this...	negative	31	64	[encouraged, positive, comments, film, looking...	0	
5	phil the alien is one of those quirky films wh...	negative	35	43	[phil, alien, one, quirky, films, humour, base...	0	
6	i saw this movie when i was about 12 when it c...	negative	32	96	[saw, movie, 12, came, recall, scariest, scene...	0	
7	so im not a big fan of boll's work but then ag...	negative	86	178	[im, big, fan, bolls, work, many, enjoyed, mov...	0	
8	this a fantastic movie of three prisoners who ...	positive	7	25	[fantastic, movie, three, prisoners, become, f...	0	
9	this movie made it into one of my top 10 most ...	negative	117	112	[movie, made, one, top, 10, awful, movies, hor...	0	

- Above is the Dataframe after removing the stopword and punctuation, for analysing the count of both before and after.

In [24]:
```
imdb['review'][3]
```

Out[24]:
```
"this show was an amazing, fresh & innovative idea in the 70's when it first aired. the first 7 or 8 years were br
illiant, but things dropped off after that. by 1990, the show was not really funny anymore, and it's continued its
decline further to the complete waste of time it is today.<br /><br />it's truly disgraceful how far this show has
fallen. the writing is painfully bad, the performances are almost as bad - if not for the mildly entertaining resp
ite of the guest-hosts, this show probably wouldn't still be on the air. i find it so hard to believe that the sam
e creator that hand-selected the original cast also chose the band of hacks that followed. how can one recognize s
uch brilliance and then see fit to replace it with such mediocrity? i felt i must give 2 stars out of respect for
the original cast that made this show such a huge success. as it is now, the show is just awful. i can't believe i
t's still on the air."
```

In [25]:
```
text_token = imdb_2['new_review'][3]
print(text_token)
```

```
['show', 'amazing', 'fresh', 'innovative', 'idea', '70s', 'first', 'aired', 'first', '7', '8', 'years', 'brilliant',
'things', 'dropped', '1990', 'show', 'really', 'funny', 'anymore', 'continued', 'decline', 'complete', 'waste', 'tim
e', 'todaybr', 'br', 'truly', 'disgraceful', 'far', 'show', 'fallen', 'writing', 'painfully', 'bad', 'performances',
'almost', 'bad', 'mildly', 'entertaining', 'respite', 'guesthosts', 'show', 'probably', 'wouldnt', 'still', 'air',
'find', 'hard', 'believe', 'creator', 'handselected', 'original', 'cast', 'also', 'chose', 'band', 'hacks', 'followe
d', 'one', 'recognize', 'brilliance', 'see', 'fit', 'replace', 'mediocrity', 'felt', 'must', 'give', '2', 'stars',
'respect', 'original', 'cast', 'made', 'show', 'huge', 'success', 'show', 'awful', 'cant', 'believe', 'still', 'ai
r']
```

- Above two are the example of a one review after Preprocess Text Data(Remove punctuation, Perform Tokenization.

Stem

In [26]:
```
def tokenize_and_stem(text):
    tokens = text
    stemmer = PorterStemmer()

    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    return stemmed_tokens

imdb_2['stemmed_text'] = imdb_2['new_review'].apply(tokenize_and_stem)
```

lim

In [27]:
```
def tokenize_and_lemmatize(text):

    tokens = text
    lemmatizer = WordNetLemmatizer()

    lemmatized_tokens = [lemmatizer.lemmatize(word) for word in tokens]

    return lemmatized_tokens

imdb_2['lemmatized_text'] = imdb_2['new_review'].apply(tokenize_and_lemmatize)
```

Output after tokenizing and removing the punctuation and stopwords :

In [28]:
```
text_token = imdb_2['new_review'][3]
print(text_token)
```

```
['show', 'amazing', 'fresh', 'innovative', 'idea', '70s', 'first', 'aired', 'first', '7', '8', 'years', 'brilliant',
'things', 'dropped', '1990', 'show', 'really', 'funny', 'anymore', 'continued', 'decline', 'complete', 'waste', 'tim
e', 'todaybr', 'br', 'truly', 'disgraceful', 'far', 'show', 'fallen', 'writing', 'painfully', 'bad', 'performances',
'almost', 'bad', 'mildly', 'entertaining', 'respite', 'guesthosts', 'show', 'probably', 'wouldnt', 'still', 'air',
'find', 'hard', 'believe', 'creator', 'handselected', 'original', 'cast', 'also', 'chose', 'band', 'hacks', 'followe
d', 'one', 'recognize', 'brilliance', 'see', 'fit', 'replace', 'mediocrity', 'felt', 'must', 'give', '2', 'stars',
'respect', 'original', 'cast', 'made', 'show', 'huge', 'success', 'show', 'awful', 'cant', 'believe', 'still', 'ai
r']
```

Output after tokenizing, removing the punctuation and stopwords, & applying the Lemmatization :

In [29]:
```
l_text = imdb_2['lemmatized_text'][3]
print(l_text)
```

```
['show', 'amazing', 'fresh', 'innovative', 'idea', '70', 'first', 'aired', 'first', '7', '8', 'year', 'brilliant',
'thing', 'dropped', '1990', 'show', 'really', 'funny', 'anymore', 'continued', 'decline', 'complete', 'waste', 'tim
e', 'todaybr', 'br', 'truly', 'disgraceful', 'far', 'show', 'fallen', 'writing', 'painfully', 'bad', 'performance',
'almost', 'bad', 'mildly', 'entertaining', 'respite', 'guesthosts', 'show', 'probably', 'wouldnt', 'still', 'air',
'find', 'hard', 'believe', 'creator', 'handselected', 'original', 'cast', 'also', 'chose', 'band', 'hack', 'followe
d', 'one', 'recognize', 'brilliance', 'see', 'fit', 'replace', 'mediocrity', 'felt', 'must', 'give', '2', 'star', 'r
espect', 'original', 'cast', 'made', 'show', 'huge', 'success', 'show', 'awful', 'cant', 'believe', 'still', 'air']
```

Output after tokenizing, removing the punctuation and stopwords, & applying the Stemming :

In [30]:
```
s_text = imdb_2['stemmed_text'][3]
print(s_text)
```

```
['show', 'amaz', 'fresh', 'innov', 'idea', '70', 'first', 'air', 'first', '7', '8', 'year', 'brilliant', 'thing', 'd
rop', '1990', 'show', 'realli', 'funni', 'anymor', 'continu', 'declin', 'complet', 'wast', 'time', 'todaybr', 'br',
'truli', 'disgrac', 'far', 'show', 'fallen', 'write', 'pain', 'bad', 'perform', 'almost', 'bad', 'mildli', 'entertai
n', 'respit', 'guesthost', 'show', 'probabl', 'wouldnt', 'still', 'air', 'find', 'hard', 'believ', 'creator', 'hands
elect', 'origin', 'cast', 'also', 'chose', 'band', 'hack', 'follow', 'one', 'recogn', 'brillianc', 'see', 'fit', 're
plac', 'mediocr', 'felt', 'must', 'give', '2', 'star', 'respect', 'origin', 'cast', 'made', 'show', 'huge', 'succes
s', 'show', 'aw', 'cant', 'believ', 'still', 'air']
```

In [31]:
```
imdb_2.head()
```

Out[31]:

	review	sentiment	punctuation_count	Stopword_count	new_review	after_removing_punctuation_count	after_Stopword_c
0	i thought this was a wonderful way to spend ti...	positive	40	83	[thought, wonderful, way, spend, time, hot, su...	0	
1	probably my all-time favorite movie, a story o...	positive	28	69	[probably, alltime, favorite, movie, story, se...	0	
2	i sure would like to see a resurrection of a u...	positive	12	86	[sure, would, like, see, resurrection, dated, ...	0	
3	this show was an amazing, fresh & innovative i...	negative	33	96	[show, amazing, fresh, innovative, idea, 70s, ...	0	
4	encouraged by the positive comments about this...	negative	31	64	[encouraged, positive, comments, film, looking...	0	

- In the context of movie review classification, where the primary goal is often to identify sentiment or opinions expressed in the reviews, lemmatization might be preferred over stemming due to its ability to produce more meaningful tokens that better capture the semantics of the words.
- However, it's essential to experiment with both approaches and evaluate their performance using appropriate metrics to determine which one works best for a specific dataset and classification task. Additionally, the choice of lemmatization or stemming can also depend on the specific requirements of the downstream classification algorithm and the computational resources available.

Perform TFIDF Vectorization

In [32]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

clean_data = [' '.join(doc) for doc in imdb_2['lemmatized_text']]

tfidf_vectorizer = TfidfVectorizer()

X_tfidf = tfidf_vectorizer.fit_transform(clean_data)


print(f'Output: {(X_tfidf.shape)}(number_of_samples, vocabulary_size)')
```

```
Output: (25000, 113304)(number_of_samples, vocabulary_size)
```

- Converting the Tokenization and lemmatized data to the numeric value to perform the next operations.

Exploring parameter settings using GridSearchCV on Random Forest & Gradient Boosting Classifier. Use Xgboost instead of Gradient Boosting if it's taking a very long time in GridSearchCV

Evaluation before reducing the dataset :

In [72]:
```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report
```

In [73]:
```
cleaned_data = {'cleaned_text': imdb_2['lemmatized_text'], 'target': imdb_2['sentiment']}
main_df = pd.DataFrame(cleaned_data)
main_df.head()
```

Out[73]:

	cleaned_text	target
0	[thought, wonderful, way, spend, time, hot, su...	positive
1	[probably, alltime, favorite, movie, story, se...	positive
2	[sure, would, like, see, resurrection, dated, ...	positive
3	[show, amazing, fresh, innovative, idea, 70, f...	negative
4	[encouraged, positive, comment, film, looking,...	negative

In [74]:

```
print(main_df.groupby('target').size())
```

```
target
negative    12500
positive    12500
dtype: int64
```

- Checking the count of the each target column values, model will not perform bias as both the values are same.

Mapping the Tareget Column :

In [75]:

```
name_mapping = { 'negative':0,'positive':1 }
main_df["target"] = main_df['target'].map(name_mapping)
display(main_df)
```

	cleaned_text	target
0	[thought, wonderful, way, spend, time, hot, su...	1
1	[probably, alltime, favorite, movie, story, se...	1
2	[sure, would, like, see, resurrection, dated, ...	1
3	[show, amazing, fresh, innovative, idea, 70, f...	0
4	[encouraged, positive, comment, film, looking,...	0
...	...	...
24995	[first, tuned, morning, news, thought, wow, fi...	0
24996	[got, one, week, ago, love, modern, light, fil...	1
24997	[bad, plot, bad, dialogue, bad, acting, idioti...	0
24998	[catholic, taught, parochial, elementary, scho...	0
24999	[one, expects, star, trek, movie, high, art, f...	0

25000 rows × 2 columns

- Mapping the target column converting the str to int value to enabling better decision-making, analysis, and operations.

Splitting data into (60, 40) ratio 60 for training and rest for testing the model.

In [76]:

```
X_train, X_test, Y_train, Y_test = train_test_split(main_df['cleaned_text'], main_df['target'], test_size=0.4, stra

print('Train Test split ratio is : [60, 40]','\n')
print("Training set:")
print("X_train shape:", X_train.shape)
print("y_train shape:", Y_train.shape)


print("\nTesting set:")
print("X_test shape:", X_test.shape)
print("y_test shape:", Y_test.shape)
```

```
Train Test split ratio is : [60, 40]

Training set:
X_train shape: (15000,)
y_train shape: (15000,)

Testing set:
X_test shape: (10000,)
y_test shape: (10000,)
```

- We have 15000 rows for training the model and 10000 rows for the testing.

Applying the TFIDF Vectorization method to the X_test and X_train

In [77]:
```
X_train = [' '.join(doc) for doc in X_train]
X_test = [' '.join(doc) for doc in X_test]

tfidf_vectorizer = TfidfVectorizer()

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

Defining the classifier models :

In [78]:
```
rf_classifier = RandomForestClassifier()
gb_classifier = GradientBoostingClassifier()
xgb_classifier = xgb.XGBClassifier()
```

Defining parameters for the the model to perform Grid Search CV :

In [62]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
param_grid_gb = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

param_grid_xgb = {
        'n_estimators': [100, 200, 300],
        'learning_rate': [0.05, 0.1, 0.2],
        'max_depth': [3, 4, 5],
        'min_child_weight': [1, 3, 5],
        'gamma': [0, 0.1, 0.2]
    }
```

Perform GridSearchCV for Random Forest :

In [39]:
```
grid_search_rf = GridSearchCV(rf_classifier, param_grid_rf, cv=5, n_jobs=5)
start_time1 = time.time()
grid_search_rf.fit(X_train_tfidf, Y_train)
latency = time.time() - start_time1
print("Latency for Random Forest: ", latency )
```

```
Latency for Random Forest:  1600.5208892822266
```

Converting the latency time to minutes:

In [70]:
```
tak_time1 = latency/60
print(f"Random Forest took {tak_time1} minutes to perform the Grid search CV")
```

```
Random Forest took 26.675348154703777 minutes to perform the Grid search CV
```

Best Parameters and score for Random Forest :

In [69]:
```
best_params_rf = grid_search_rf.best_params_
best_score_rf = grid_search_rf.best_score_
print("Best Parameters obtained after performing the GSVC :", best_params_rf, '\n')
print("Model Best score :", round(best_score_rf, 4)*100)
```

```
Best Parameters obtained after performing the GSVC : {'max_depth': None, 'min_samples_leaf': 4, 'min_samples_split':
2, 'n_estimators': 300}

Model Best score : 85.55
```

- Perform the Grid search cv for Random forest classifier which almost took 26 minutes to obtain the best parameters.
- Model best score is 85.55 %, which is not really bad in this case.

Perform GridSearchCV for Gradient Boosting

grid_search_gb = GridSearchCV(gb_classifier, param_grid_gb, cv=5, n_jobs=-5) start_time2 = time.time() grid_search_gb.fit(X_train_tfidf, Y_train) print(grid_search_gb.refit_time_, '\n') latency2 = time.time() - start_time2 print("Latency for Gradient Boosting: ", latency2) best_params = grid_search_xgb.best_params_ best_score = grid_search_xgb.best_score_ print("Best performing model parameters:") print(best_params) print("Best score:", best_score)

Going to use the XGB, reason because the Gradient boosting was taking so long to perform GSCV I have almost wait for around 8-10 hours for the result but i did not received the output. So later i

decide to move with XGB

Perform GridSearchCV for XGB :

In [63]:
```
grid_search_xgb = GridSearchCV(xgb_classifier, param_grid_xgb, cv=5, n_jobs=-5)

start_time2 = time.time()

grid_search_xgb.fit(X_train_tfidf, Y_train)

latency2 = time.time() - start_time2
print("Latency for Random Forest: ", latency2)
```

```
Latency for Random Forest:  7599.761258840561
```

Converting the latency time to minutes:

In [66]:
```
tak_time = latency2/60
print(f"XGB took {tak_time} minutes to perform the Grid search CV")
```

```
XGB took 126.66268764734268 minutes to perform the Grid search CV
```

Time required to perform GSCV using XGB classifier is 2 hours and around 6 minutes to complete the process.

In [68]:
```
best_params_xgb = grid_search_xgb.best_params_
best_score_xgb = grid_search_xgb.best_score_

print("Best Parameters obtained after performing the GSVC :", best_params_xgb, '\n')
print("Model Best score :", round(best_score_xgb, 4)*100)
```

```
Best Parameters obtained after performing the GSVC : {'gamma': 0.2, 'learning_rate': 0.2, 'max_depth': 5, 'min_child
_weight': 3, 'n_estimators': 300}

Model Best score : 85.89
```

- Perform the Grid search cv for Random forest classifier which almost took 2 hours to complete the process.
- Model best score is 85.89 %, which is not really bad in this case.

Perform Final evaluation of models on the best parameter settings using the evaluation metrics

In [118…
```
best_rf_model = RandomForestClassifier(**best_params_rf)

best_rf_model.fit(X_train_tfidf, Y_train)

y_pred_rf = best_rf_model.predict(X_test_tfidf)

accuracy_rf = accuracy_score(Y_test, y_pred_rf)
print("Accuracy of Random Forest model:", round(accuracy_rf, 2)*100)

print("Random Forest Classification Report:")
print(classification_report(Y_test, y_pred_rf))
```

```
Accuracy of Random Forest model: 86.0
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86      5000
           1       0.86      0.86      0.86      5000

    accuracy                           0.86     10000
   macro avg       0.86      0.86      0.86     10000
weighted avg       0.86      0.86      0.86     10000
```

In [85]:
```
best_XGB_model = xgb.XGBClassifier(**best_params_xgb)

best_XGB_model.fit(X_train_tfidf, Y_train)

y_pred_xgb = best_XGB_model.predict(X_test_tfidf)

accuracy_xgb = accuracy_score(Y_test, y_pred_xgb)
print("Accuracy of XGB model:", round(accuracy_xgb, 4)*100)

print(" XGB Classification Report:")
print(classification_report(Y_test, y_pred_xgb))
```

```
Accuracy of XGB model: 86.59
 XGB Classification Report:
               precision    recall  f1-score   support

           0       0.88      0.85      0.86      5000
           1       0.85      0.88      0.87      5000

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000
```

Comparing both the model performance :

- While performing the Grid search cv Random forest model took less time than the XGB. With the time differance of 1 hour and around 36 minutes.

Evaluation based on testing data set for both the models :

- Both model perform best also the accuracy is almost same for both.
- Accuracy difference is around 0.59 percent.
- But considering the classification report results XGB perform quite better than Randon forest.

In my opinion i will go with the Random Forest model as the model did not take too long to perform Grid search cv compare to XGB as already mention above that the model took almost 2 hours and 36 minutes. Also I don't see major differance in accuracy.

```
- Also i want to mention one thing that while performing the same task with the Gradient Boosting
i wait for around 8-10 hours for receiving the output so i decided to go with XGB. But that also
take long time perform the same operation.
- So considering the time factor the Random Forest is best than XGB and Gradient Boosting.
```