EDA_NLP_sentiment_analysis_of_movie_reviews

March 4, 2023

1 Sentiment Analysis of IMDB Movie Reviews (Part 1: Exploratory Data Analysis)

Problem Statement:

In this, we have to predict the number of positive and negative reviews based on sentiments by using different classification models. We start with processing the data and explore the data through visualization.

Side note for profs: I have done NLP analysis before,so this work is using as a this template as a baseline.

1.1 Data Processing

Import necessary libraries

[ ]:
```
#Load the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re,string,unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
# from textblob import TextBlob
```

```
# from textblob import Word
from sklearn.metrics import␣
 ↪classification_report,confusion_matrix,accuracy_score
```

Import the training dataset

[ ]:
```
#importing the training data
imdb_data=pd.read_csv('IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

[ ]:
```
                                              review sentiment
0  One of the other reviewers has mentioned that …  positive
1  A wonderful little production. <br /><br />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
5  Probably my all-time favorite movie, a story o…  positive
6  I sure would like to see a resurrection of a u…  positive
7  This show was an amazing, fresh & innovative i…  negative
8  Encouraged by the positive comments about this…  negative
9  If you like original gut wrenching laughter yo…  positive
```

[ ]:
```
#Summary of the dataset
imdb_data.describe()
```

[ ]:
```
                                                   review sentiment
count                                               50000     50000
unique                                              49582         2
top     Loved today's show!!! It was a variety and not…  positive
freq                                                    5     25000
```

Sentiment count

[ ]:
```
#sentiment count
imdb_data['sentiment'].value_counts()
```

[ ]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

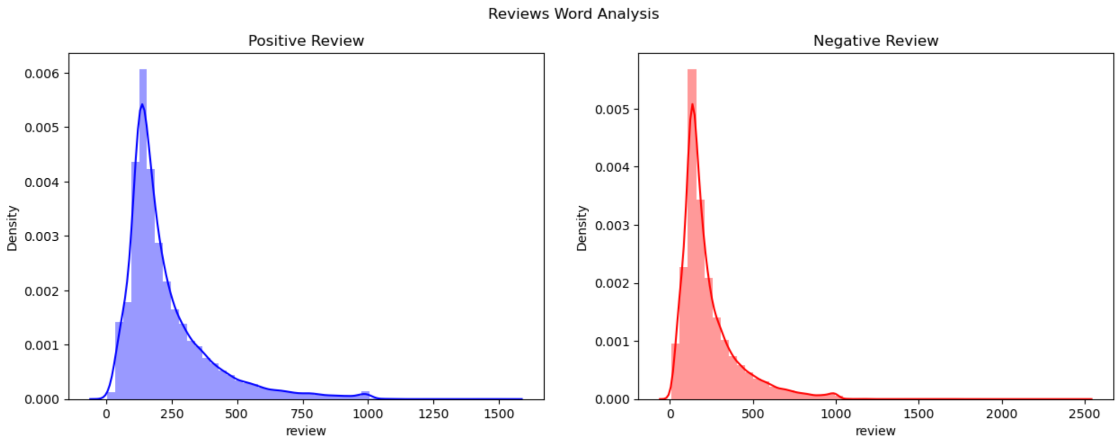
We can see that the dataset is balanced.

1.2 Exploratory Data Analysis

[ ]:
```
#Analyse the count of words in each segment- both positive and negative reviews
#Function for checking word length
def cal_len(data):
    return len(data)

#Create generic plotter with Seaborn
def plot_count(count_ones,count_zeros,title_1,title_2,subtitle):
    fig,(ax1,ax2)=plt.subplots(1,2,figsize=(15,5))
    sns.distplot(count_zeros,ax=ax1,color='Blue')
    ax1.set_title(title_1)
    sns.distplot(count_ones,ax=ax2,color='Red')
    ax2.set_title(title_2)
    fig.suptitle(subtitle)
    plt.show()



count_good=imdb_data[imdb_data['sentiment']=='positive']
count_bad=imdb_data[imdb_data['sentiment']=='negative']
count_good_words=count_good['review'].str.split().apply(lambda z:cal_len(z))
count_bad_words=count_bad['review'].str.split().apply(lambda z:cal_len(z))
# print("Positive Review Words:" + str(count_good_words))
# print("Negative Review Words:" + str(count_bad_words))
plot_count(count_good_words,count_bad_words,"Positive Review","Negative␣
↪Review","Reviews Word Analysis")
```

```
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_14986/539831338.py:9:
UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(count_zeros,ax=ax1,color='Blue')
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_14986/539831338.py:11
: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
```

```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(count_ones,ax=ax2,color='Red')
```



Analyzing word count for positive and negative reviews, it seens that negative reviews have a much longer tails, which might result from angry viewers write long critics for bad movies. It shows us a potential of using word count to classify positive or negative sentiments.

Spliting the training dataset

[ ]:
```python
#split the dataset
#train dataset
train_reviews=imdb_data.review[:40000]
train_sentiments=imdb_data.sentiment[:40000]
#test dataset
test_reviews=imdb_data.review[40000:]
test_sentiments=imdb_data.sentiment[40000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)
```

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

Removing html strips and noise text

[ ]:
```python
#Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)
```

```
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)
```

```
/Users/swimmingcircle/opt/anaconda3/lib/python3.9/site-
packages/bs4/__init__.py:435: MarkupResemblesLocatorWarning: The input looks
more like a filename than markup. You may want to open this file and pass the
filehandle into Beautiful Soup.
  warnings.warn(
```

Removing special characters

[ ]:
```
#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characters)
```

Text stemming

Removing the suffix from a word and reduce it to its root word (e.g. Flying" is a word and its suffix is "ing", if we remove "ing" from "Flying" then we will get base word or root word which is "Fly".)

[ ]:
```
#Stemming the text
def simple_stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)
```

Removing stopwords

[ ]:
```
#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nltk.corpus.stopwords.words('english')
```

[ ]:
```
#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
```

```python
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in␣
↪stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in␣
↪stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)
```

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

Normalized train reviews

[ ]:
```python
#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
```

[ ]:
```
'one review ha mention watch 1 oz episod youll hook right thi exactli happen
meth first thing struck oz wa brutal unflinch scene violenc set right word go
trust thi show faint heart timid thi show pull punch regard drug sex violenc
hardcor classic use wordit call oz nicknam given oswald maximum secur state
penitentari focus mainli emerald citi experiment section prison cell glass front
face inward privaci high agenda em citi home manyaryan muslim gangsta latino
christian italian irish moreso scuffl death stare dodgi deal shadi agreement
never far awayi would say main appeal show due fact goe show wouldnt dare forget
```

```
pretti pictur paint mainstream audienc forget charm forget romanceoz doesnt mess
around first episod ever saw struck nasti wa surreal couldnt say wa readi watch
develop tast oz got accustom high level graphic violenc violenc injustic crook
guard wholl sold nickel inmat wholl kill order get away well manner middl class
inmat turn prison bitch due lack street skill prison experi watch oz may becom
comfort uncomfort viewingthat get touch darker side'
```

Normalized test reviews

```
[ ]: #Normalized test reviews
norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
```

```
[ ]: 'read review watch thi piec cinemat garbag took least 2 page find somebodi els
didnt think thi appallingli unfunni montag wasnt acm humour 70 inde ani era thi
isnt least funni set sketch comedi ive ever seen itll till come along half skit
alreadi done infinit better act monti python woodi allen wa say nice piec anim
last 90 second highlight thi film would still get close sum mindless
drivelridden thi wast 75 minut semin comedi onli world semin realli doe mean
semen scatolog humour onli world scat actual fece precursor joke onli mean thi
handbook comedi tit bum odd beaver niceif pubesc boy least one hand free havent
found playboy exist give break becaus wa earli 70 way sketch comedi go back
least ten year prior onli way could even forgiv thi film even made wa gunpoint
retro hardli sketch clown subtli pervert children may cut edg circl could actual
funni come realli quit sad kept go throughout entir 75 minut sheer belief may
save genuin funni skit end gave film 1 becaus wa lower scoreand onli recommend
insomniac coma patientsor perhap peopl suffer lockjawtheir jaw would final drop
open disbelief'
```

Bags of words model

It is used to convert text documents to numerical vectors or bag of words.

```
[ ]: #Count vectorizer for bag of words
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
#transformed train reviews
cv_train_reviews=cv.fit_transform(norm_train_reviews)
#transformed test reviews
cv_test_reviews=cv.transform(norm_test_reviews)

print('BOW_cv_train:',cv_train_reviews.shape)
print('BOW_cv_test:',cv_test_reviews.shape)
#vocab=cv.get_feature_names()-toget feature names
```

```
BOW_cv_train: (40000, 6209089)
BOW_cv_test: (10000, 6209089)
```

Term Frequency-Inverse Document Frequency model (TFIDF)

It is used to convert text documents to matrix of tfidf features.

[ ]:
```
#Tfidf vectorizer
tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
#transformed train reviews
tv_train_reviews=tv.fit_transform(norm_train_reviews)
#transformed test reviews
tv_test_reviews=tv.transform(norm_test_reviews)
print('Tfidf_train:',tv_train_reviews.shape)
print('Tfidf_test:',tv_test_reviews.shape)
```

```
Tfidf_train: (40000, 6209089)
Tfidf_test: (10000, 6209089)
```

1.3 Word cloud analysis

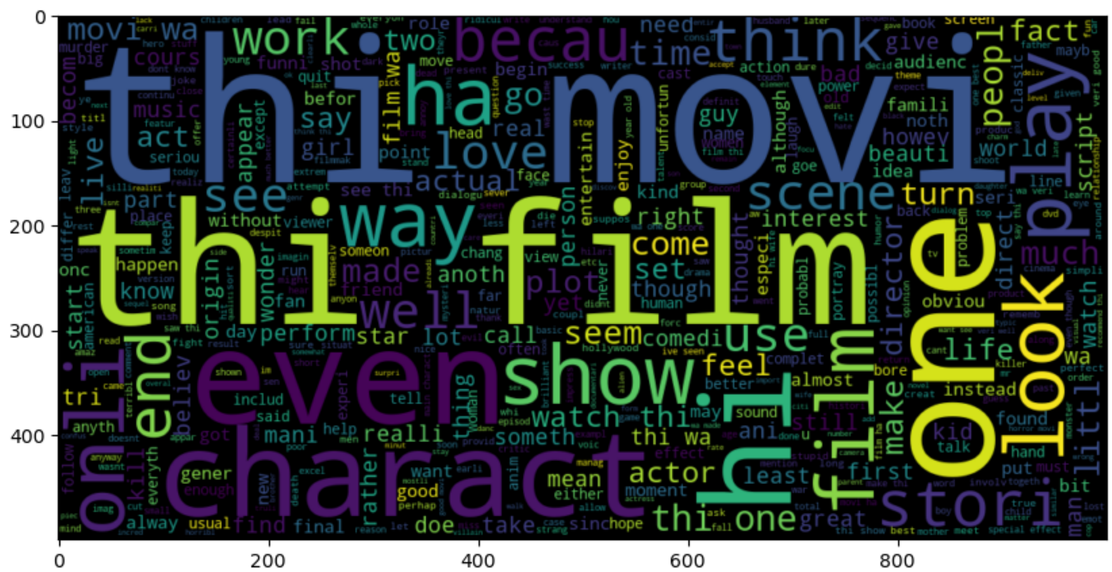
1.3.1 Bags of words

[ ]:
```
string_texts = ' '.join(norm_train_reviews.tolist())
```

[ ]:
```
# imdb_data.review[:40000]
pos_reviews = count_good['review']
neg_reviews = count_bad['review']

pos_string_texts = ' '.join(pos_reviews.tolist())
neg_string_texts = ' '.join(neg_reviews.tolist())
```

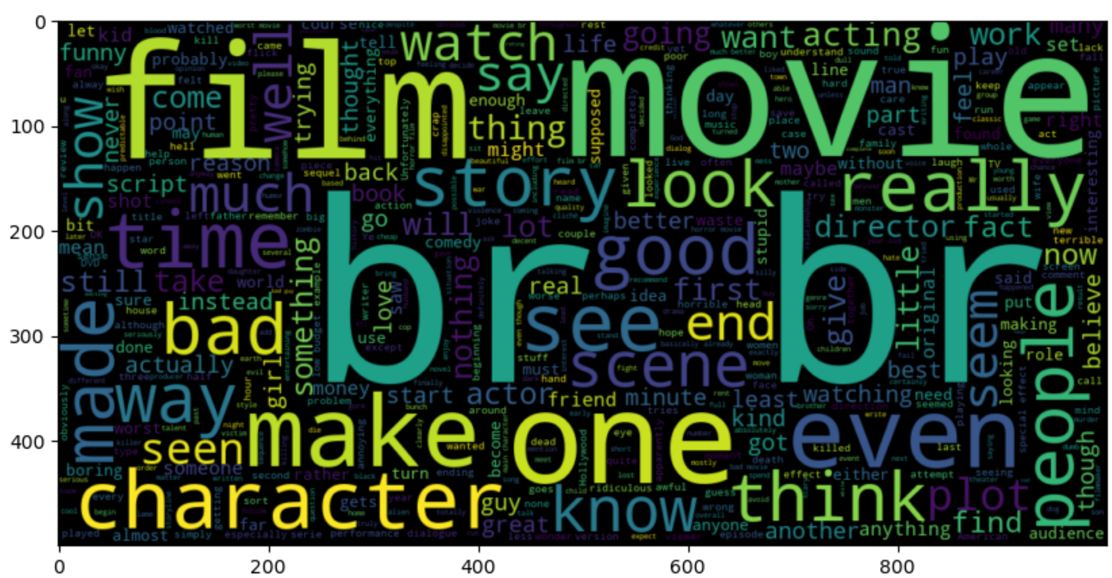
[ ]:
```
# For all reivews
plt.figure(figsize=(10,10))
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
all_words =WC.generate(string_texts)
plt.imshow(all_words,interpolation='bilinear')
plt.show
```

[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




```
[ ]: # For positive reviews
plt.figure(figsize=(10,10))
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
all_words =WC.generate(pos_string_texts)
plt.imshow(all_words,interpolation='bilinear')
plt.show

# For negative reviews
plt.figure(figsize=(10,10))
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
all_words =WC.generate(neg_string_texts)
plt.imshow(all_words,interpolation='bilinear')
plt.show
```

```
[ ]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Word clouds show nothing informative when plotting bags of words. We believe plotting tf-idf will be more informative if we have enough computational power.

[ ]:
```
# #Attempt to convert Tf-idf vector back to word for wordcloud, but it exceeds
 ↪its computational capacity

# # Get the feature names from the TfidfVectorizer
# feature_names = tv.get_feature_names_out()
```

```
# # Create a dictionary mapping feature indices to feature names
# feature_dict = dict(zip(range(len(feature_names)), feature_names))

# # Create a dense matrix of the tf-idf values
# dense_matrix = tv_train_reviews.todense()

# # Get the tf-idf scores for each feature
# tfidf_scores = dense_matrix.mean(axis=0).tolist()[0]

# # Create a dictionary mapping feature names to tf-idf scores
# tfidf_dict = dict(zip(feature_names, tfidf_scores))

# # Create a WordCloud object with the desired parameters
# wordcloud = WordCloud(width=800, height=800, background_color='white',␣
↪colormap='inferno', stopwords=None, min_font_size=10)

# # Generate the word cloud from the tf-idf dictionary
# wordcloud.generate_from_frequencies(tfidf_dict)

# # Display the word cloud
# plt.figure(figsize=(8,8), facecolor=None)
# plt.imshow(wordcloud)
# plt.axis("off")
# plt.tight_layout(pad=0)
# plt.show()
```

1.4 Unigram, Bigram, Trigram analysis

[ ]:
```
count_good=imdb_data[imdb_data['sentiment']=='positive']
count_bad=imdb_data[imdb_data['sentiment']=='negative']
```

[ ]:
```
from nltk.util import ngrams
from nltk import FreqDist

count_good['review'] = count_good['review'].apply(denoise_text)
count_bad['review'] = count_bad['review'].apply(denoise_text)



def get_corpus(text):
    words = []
    for i in text:
        for j in i.split():
            words.append(j.strip())

    return words
corpus = get_corpus(imdb_data.review)
```

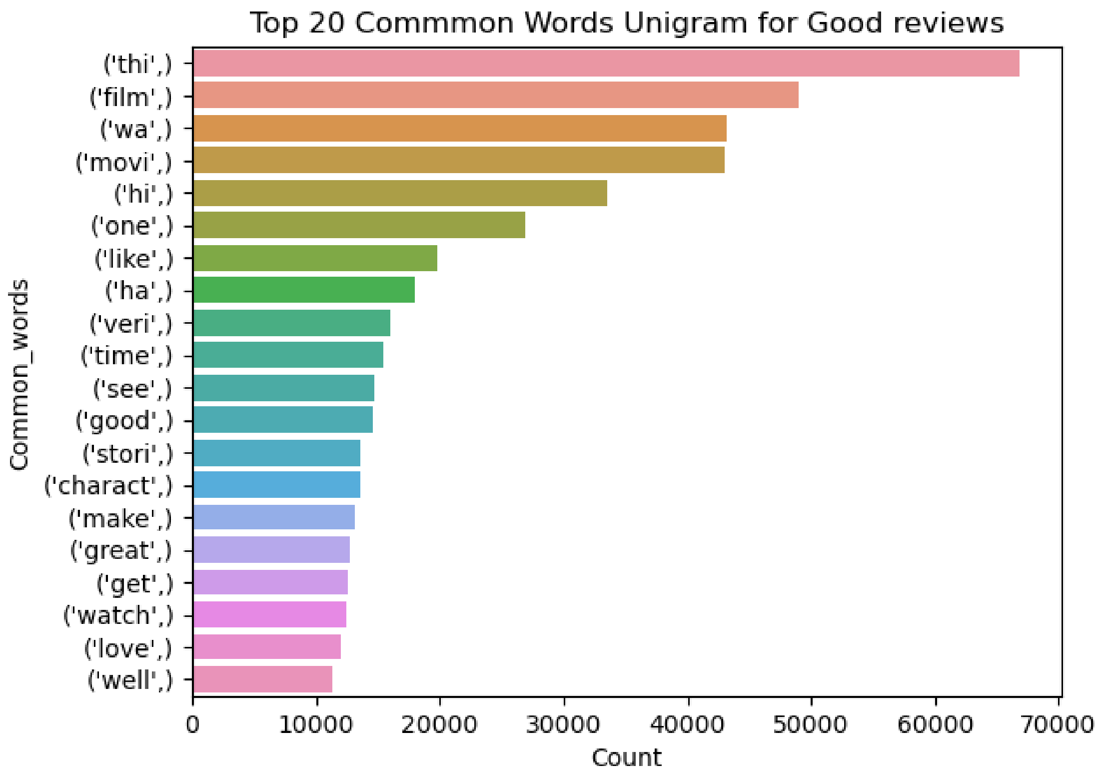
```
pos_corpus = get_corpus(count_good.review)
neg_corpus = get_corpus(count_bad.review)

def top_n_grams(corpus, n):
    p = ' '.join(corpus)
    # Tokenize the text into bigrams
    bigrams = list(ngrams(p.split(), n))

    # Calculate the frequency of each bigram
    freq_dist = FreqDist(bigrams)

    # Sort the bigrams based on their frequency
    sorted_ngrams = sorted(freq_dist.items(), key=lambda x: x[1], reverse=True)

    # Get the top 10 bigrams
    top_n_bigrams = sorted_ngrams[:20]


    return top_n_bigrams
```

```
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_14986/920115542.py:4:
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_guide/indexing.html#returning-a-view-versus-a-copy
  count_good['review'] = count_good['review'].apply(denoise_text)
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_14986/920115542.py:5:
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_guide/indexing.html#returning-a-view-versus-a-copy
  count_bad['review'] = count_bad['review'].apply(denoise_text)
```

1.5 Unigram

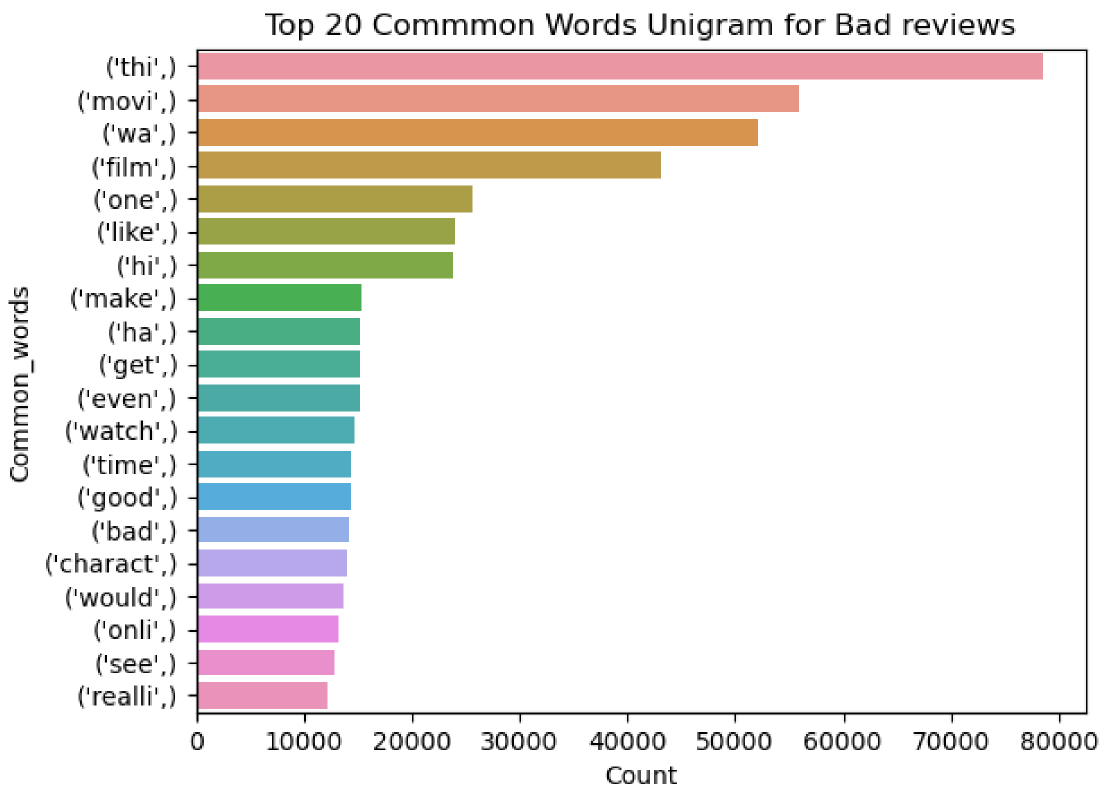
[ ]:
```
most_common_uni = top_n_grams(pos_corpus,1)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Unigram for Good reviews")
```

[ ]: Text(0.5, 1.0, 'Top 20 Commmon Words Unigram for Good reviews')



[ ]:
```
most_common_uni = top_n_grams(neg_corpus,1)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Unigram for Bad reviews")
```

[ ]: Text(0.5, 1.0, 'Top 20 Commmon Words Unigram for Bad reviews')

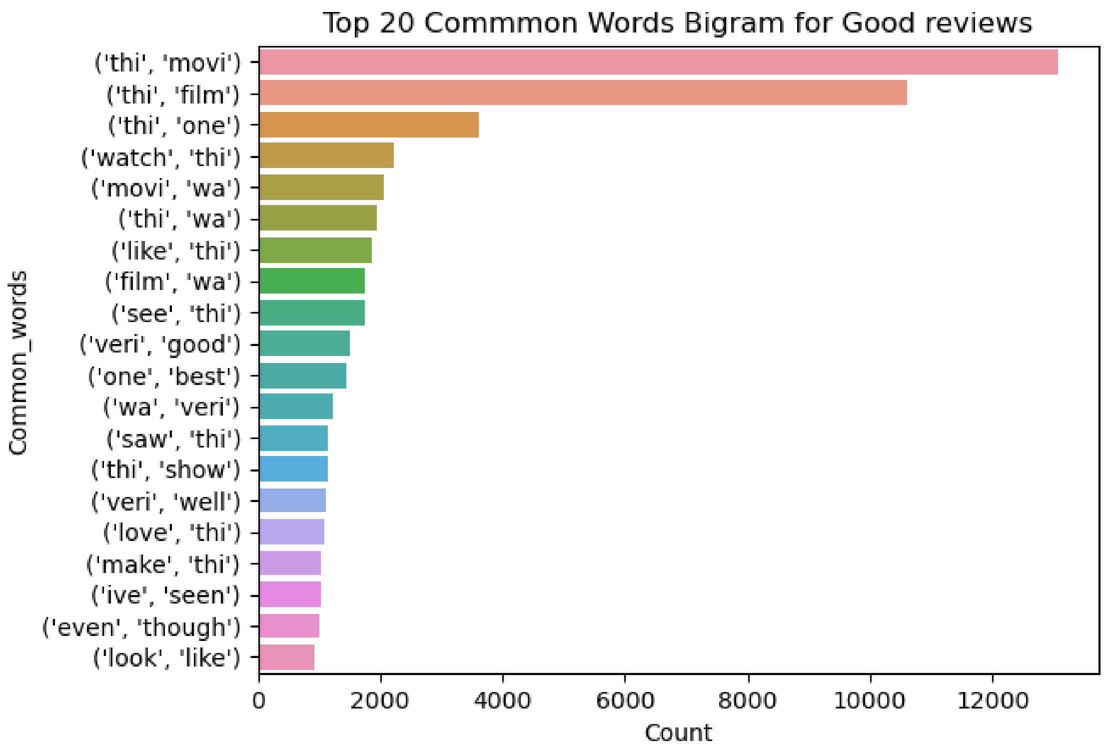


For unigram, we can see that there isn't much difference between good and bad reviews.

1.6 Bigram

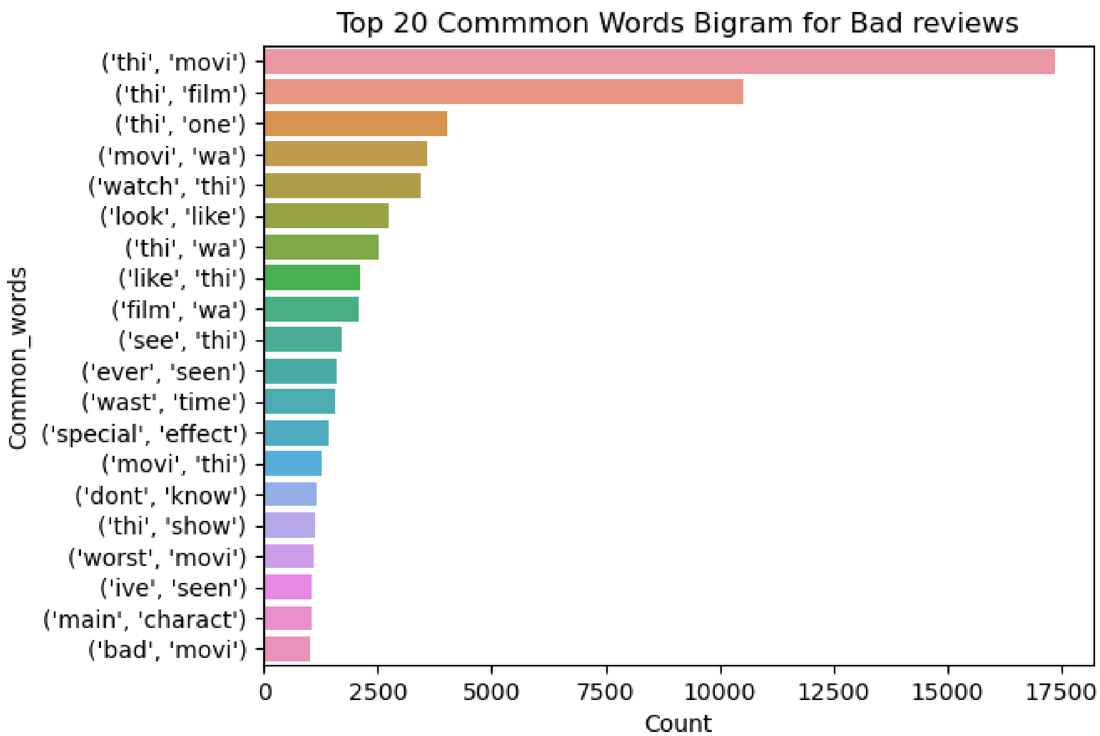
[ ]:
```
most_common_uni = top_n_grams(pos_corpus,2)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Bigram for Good reviews")
```

[ ]:
```
Text(0.5, 1.0, 'Top 20 Commmon Words Bigram for Good reviews')
```



[ ]:
```
most_common_uni = top_n_grams(neg_corpus,2)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Bigram for Bad reviews")
```

[ ]:
```
Text(0.5, 1.0, 'Top 20 Commmon Words Bigram for Bad reviews')
```



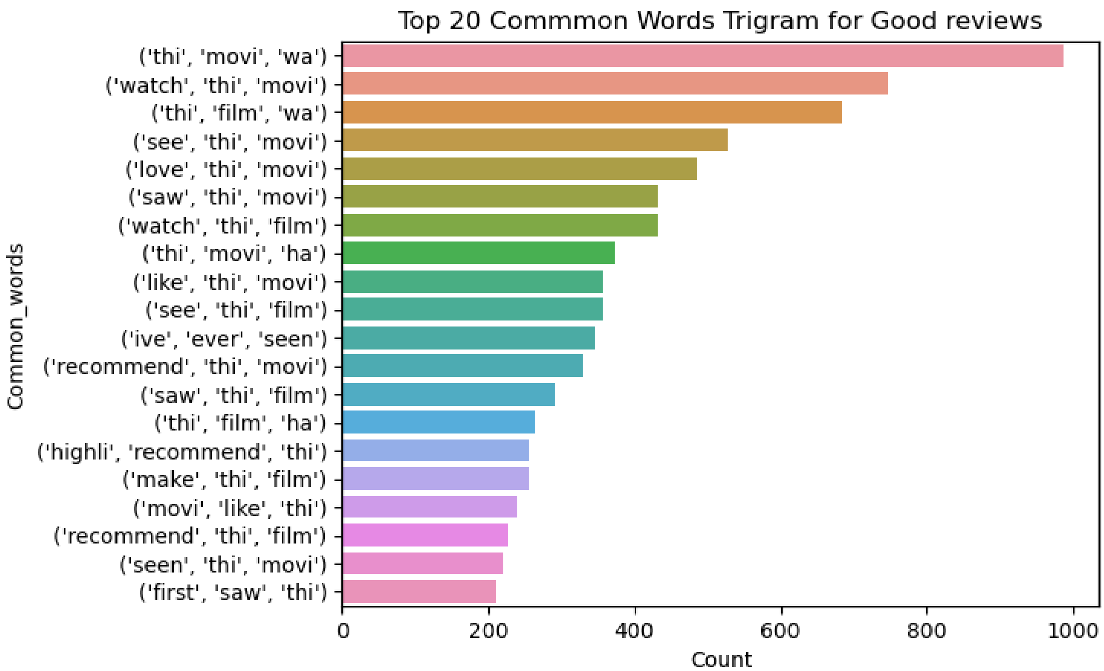
In Bigram, though the top 9 common words are the same for both good and bad reviews, but we can see some indicators to show negative and positive comments. - Example good reviews bigrams: (very, good), (very, well), (love, this) - Example bad reviews bigrams: (waste time), (worst, movie), (bad, movie)

We can see that 'good' and 'bad' are the key words.

1.7 Trigram

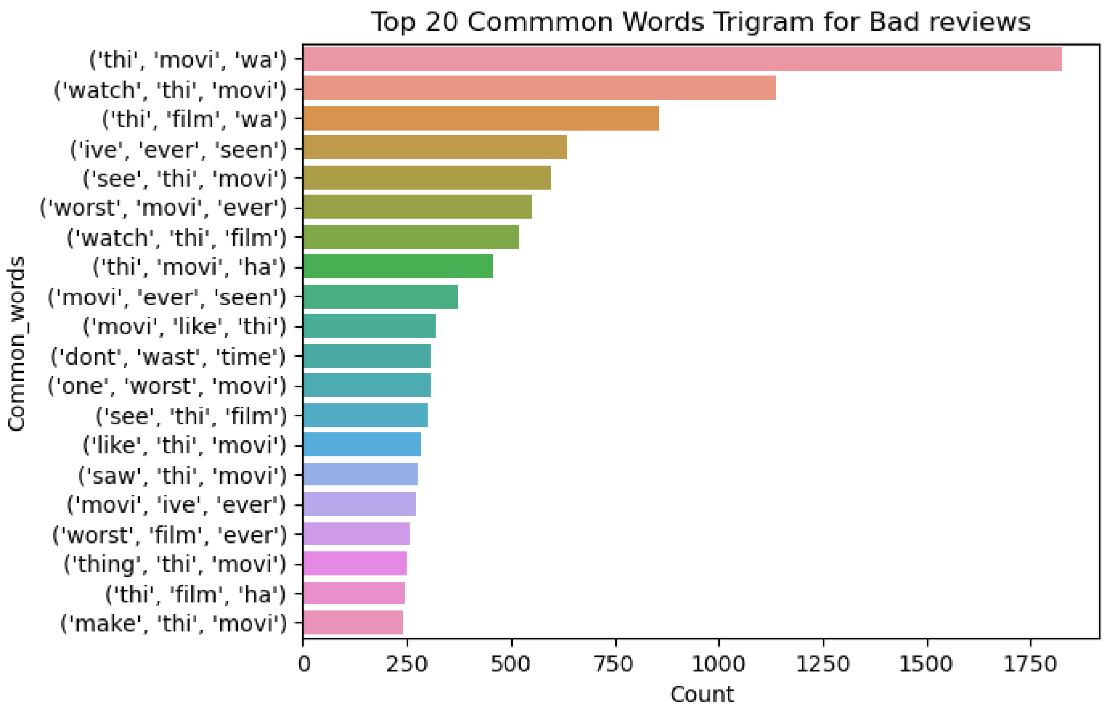
[ ]:
```
most_common_uni = top_n_grams(pos_corpus,3)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Trigram for Good reviews")
```

[ ]:
```
Text(0.5, 1.0, 'Top 20 Commmon Words Trigram for Good reviews')
```



[ ]:
```
most_common_uni = top_n_grams(neg_corpus,3)
most_common_uni = dict(most_common_uni)
temp = pd.DataFrame(columns = ["Common_words" , 'Count'])
temp["Common_words"] = list(most_common_uni.keys())
temp["Count"] = list(most_common_uni.values())

fig = sns.barplot(temp, x="Count", y="Common_words", orientation='horizontal')
plt.title("Top 20 Commmon Words Trigram for Bad reviews")
```

[ ]:
```
Text(0.5, 1.0, 'Top 20 Commmon Words Trigram for Bad reviews')
```



Trigram even shows even more difference between good and bad reviews. - Example good reviews bigrams: (love, this, movie),(like, this movie), (recommend, this movie), (highly, recommend, this), (recommend, this film) - Example bad reviews bigrams: (worst, movie, ever), (don't, waste, time), (one, worst, movie), (worst, film, ever)

Rather than using keywords 'good' and 'bad', we see more information of such as strong recommendation for good reviews. For bad reviews, it still maintains 'worst' as the main keyword.

It shows that we might be able to consider using n_grams > 3 if it occurs to be a parameter.

```
[ ]:
```

ML-NLP-sentiment-analysis-of-movie-reviews

March 4, 2023

1 Sentiment Analysis of IMDB Movie Reviews (Part 2: Machine Learning models)

Problem Statement:

In this, we have to predict the number of positive and negative reviews based on sentiments by using different classification models.

Side note for profs - I have done NLP analysis before,so this work is using as a this template as a baseline. - Feel free to ignore the BERT model code. It works but taking too long to train and often crashes the kernal. Hence, I switch to use HuggingFace training package in Sentiment Analysis of IMDB Movie Reviews (Part 3: BERT model)

1.1 Data Processing

Import necessary libraries

[ ]:
```
#Load the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import spacy
import re,string,unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
```

```
# from textblob import TextBlob
# from textblob import Word
from sklearn.metrics import␣
↪classification_report,confusion_matrix,accuracy_score
```

Import the training dataset

[ ]:
```
#importing the training data
imdb_data=pd.read_csv('IMDB Dataset.csv')
print(imdb_data.shape)
imdb_data.head(10)
```

```
(50000, 2)
```

[ ]:
```
                                              review sentiment
0  One of the other reviewers has mentioned that …  positive
1  A wonderful little production. <br /><br />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
5  Probably my all-time favorite movie, a story o…  positive
6  I sure would like to see a resurrection of a u…  positive
7  This show was an amazing, fresh & innovative i…  negative
8  Encouraged by the positive comments about this…  negative
9  If you like original gut wrenching laughter yo…  positive
```

Exploratery data analysis

[ ]:
```
#Summary of the dataset
imdb_data.describe()
```

[ ]:
```
                                                   review sentiment
count                                               50000     50000
unique                                              49582         2
top     Loved today's show!!! It was a variety and not…  positive
freq                                                    5     25000
```

Sentiment count

[ ]:
```
#sentiment count
imdb_data['sentiment'].value_counts()
```

[ ]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

We can see that the dataset is balanced.

Spliting the training dataset

```
[ ]: #split the dataset
#train dataset
train_reviews=imdb_data.review[:40000]
train_sentiments=imdb_data.sentiment[:40000]
#test dataset
test_reviews=imdb_data.review[40000:]
test_sentiments=imdb_data.sentiment[40000:]
print(train_reviews.shape,train_sentiments.shape)
print(test_reviews.shape,test_sentiments.shape)
```

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

Removing html strips and noise text

```
[ ]: #Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()

#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)

#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(denoise_text)
```

```
/Users/swimmingcircle/opt/anaconda3/lib/python3.9/site-
packages/bs4/__init__.py:435: MarkupResemblesLocatorWarning: The input looks
more like a filename than markup. You may want to open this file and pass the
filehandle into Beautiful Soup.
  warnings.warn(
```

Removing special characters

```
[ ]: #Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_special_characters)
```

Text stemming

```
[ ]: #Stemming the text
def simple_stemmer(text):
    ps=nltk.porter.PorterStemmer()
    text= ' '.join([ps.stem(word) for word in text.split()])
    return text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(simple_stemmer)
```

Removing stopwords

```
[ ]: #Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nltk.corpus.stopwords.words('english')
```

```
[ ]: #set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in␣
 ↪stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in␣
 ↪stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
imdb_data['review']=imdb_data['review'].apply(remove_stopwords)
```

```
{'until', 'all', 'didn', 'when', 'before', "mightn't", "you're", 'herself',
'that', "you'd", 'doing', 't', 'while', 'needn', 'ma', "isn't", 'their', 'has',
'him', 'mustn', 'an', 'd', 'in', 'same', 'then', 'being', 'both', 'itself',
'very', "shouldn't", 'hasn', "wouldn't", 'having', 'some', "weren't",
'ourselves', 'ours', 'against', 's', 'through', "haven't", "wasn't", "doesn't",
'ain', 'been', "she's", 'a', 'why', 'because', 'down', 'further', 'hadn',
'wasn', 'theirs', 'its', 'themselves', 'were', 'between', "you've", 'from',
'doesn', 've', 'should', 'so', "mustn't", 'himself', 'nor', 'more', 'couldn',
'had', "didn't", 'can', 'them', 'shouldn', 'for', 'o', 'don', 'which', 'to',
'shan', 'again', "it's", 'isn', 'of', 'yourselves', "shan't", 'no', 'yours',
'under', 'you', 'up', 'are', 'i', 'on', 'aren', 'and', 'about', 'y', 'll',
"needn't", 'once', 'will', 'at', 'her', 'by', 'wouldn', 're', "won't", 'other',
'there', 'what', 'she', 'or', 'haven', "hasn't", 'the', 'me', 'if', 'mightn',
'they', 'but', 'now', "hadn't", 'who', 'won', 'off', 'each', 'too', 'he', 'his',
```

```
"don't", 'it', 'how', 'does', 'is', 'few', 'be', 'below', 'not', 'into', 'am',
'as', 'over', "should've", 'most', 'hers', 'm', 'weren', 'after', 'do', 'our',
'only', 'these', 'those', 'own', 'have', 'did', 'any', "couldn't", "that'll",
'just', 'than', 'myself', 'yourself', 'whom', 'your', 'my', 'with', "you'll",
'we', 'this', 'where', 'such', "aren't", 'above', 'was', 'here', 'out',
'during'}
```

Normalized train reviews

[ ]:
```
#normalized train reviews
norm_train_reviews=imdb_data.review[:40000]
norm_train_reviews[0]
```

[ ]:
```
'one review ha mention watch 1 oz episod youll hook right thi exactli happen
meth first thing struck oz wa brutal unflinch scene violenc set right word go
trust thi show faint heart timid thi show pull punch regard drug sex violenc
hardcor classic use wordit call oz nicknam given oswald maximum secur state
penitentari focus mainli emerald citi experiment section prison cell glass front
face inward privaci high agenda em citi home manyaryan muslim gangsta latino
christian italian irish moreso scuffl death stare dodgi deal shadi agreement
never far awayi would say main appeal show due fact goe show wouldnt dare forget
pretti pictur paint mainstream audienc forget charm forget romanceoz doesnt mess
around first episod ever saw struck nasti wa surreal couldnt say wa readi watch
develop tast oz got accustom high level graphic violenc violenc injustic crook
guard wholl sold nickel inmat wholl kill order get away well manner middl class
inmat turn prison bitch due lack street skill prison experi watch oz may becom
comfort uncomfort viewingthat get touch darker side'
```

Normalized test reviews

[ ]:
```
#Normalized test reviews
norm_test_reviews=imdb_data.review[40000:]
norm_test_reviews[45005]
```

[ ]:
```
'read review watch thi piec cinemat garbag took least 2 page find somebodi els
didnt think thi appallingli unfunni montag wasnt acm humour 70 inde ani era thi
isnt least funni set sketch comedi ive ever seen itll till come along half skit
alreadi done infinit better act monti python woodi allen wa say nice piec anim
last 90 second highlight thi film would still get close sum mindless
drivelridden thi wast 75 minut semin comedi onli world semin realli doe mean
semen scatolog humour onli world scat actual fece precursor joke onli mean thi
handbook comedi tit bum odd beaver niceif pubesc boy least one hand free havent
found playboy exist give break becaus wa earli 70 way sketch comedi go back
least ten year prior onli way could even forgiv thi film even made wa gunpoint
retro hardli sketch clown subtli pervert children may cut edg circl could actual
funni come realli quit sad kept go throughout entir 75 minut sheer belief may
save genuin funni skit end gave film 1 becaus wa lower scoreand onli recommend
insomniac coma patientsor perhap peopl suffer lockjawtheir jaw would final drop
open disbelief'
```

Bags of words model

It is used to convert text documents to numerical vectors or bag of words.

[ ]:
```
#Count vectorizer for bag of words
cv=CountVectorizer(min_df=0,max_df=1,binary=False,ngram_range=(1,3))
#transformed train reviews
cv_train_reviews=cv.fit_transform(norm_train_reviews)
#transformed test reviews
cv_test_reviews=cv.transform(norm_test_reviews)

print('BOW_cv_train:',cv_train_reviews.shape)
print('BOW_cv_test:',cv_test_reviews.shape)
#vocab=cv.get_feature_names()-toget feature names
```

```
BOW_cv_train: (40000, 6209089)
BOW_cv_test: (10000, 6209089)
```

Term Frequency-Inverse Document Frequency model (TFIDF)

It is used to convert text documents to matrix of tfidf features.

[ ]:
```
#Tfidf vectorizer
tv=TfidfVectorizer(min_df=0,max_df=1,use_idf=True,ngram_range=(1,3))
#transformed train reviews
tv_train_reviews=tv.fit_transform(norm_train_reviews)
#transformed test reviews
tv_test_reviews=tv.transform(norm_test_reviews)
print('Tfidf_train:',tv_train_reviews.shape)
print('Tfidf_test:',tv_test_reviews.shape)
```

```
Tfidf_train: (40000, 6209089)
Tfidf_test: (10000, 6209089)
```

1.2 Logistic Regression

[ ]:
```
#training the model
lr=LogisticRegression(penalty='l2',max_iter=500,C=1,random_state=42)
#Fitting the model for Bag of words
lr_bow=lr.fit(cv_train_reviews,train_sentiments)
print(lr_bow)
#Fitting the model for tfidf features
lr_tfidf=lr.fit(tv_train_reviews,train_sentiments)
print(lr_tfidf)
```

```
LogisticRegression(C=1, max_iter=500, random_state=42)
LogisticRegression(C=1, max_iter=500, random_state=42)
```

[ ]:
```
#Predicting the model for bag of words
lr_bow_predict=lr.predict(cv_test_reviews)
```

```
print(lr_bow_predict)
##Predicting the model for tfidf features
lr_tfidf_predict=lr.predict(tv_test_reviews)
print(lr_tfidf_predict)
```

```
['negative' 'negative' 'negative' … 'negative' 'positive' 'positive']
['negative' 'negative' 'negative' … 'negative' 'positive' 'positive']
```

[ ]:
```
#Accuracy score for bag of words
lr_bow_score=accuracy_score(test_sentiments,lr_bow_predict)
print("lr_bow_score :",lr_bow_score)
#Accuracy score for tfidf features
lr_tfidf_score=accuracy_score(test_sentiments,lr_tfidf_predict)
print("lr_tfidf_score :",lr_tfidf_score)
```

```
lr_bow_score : 0.7512
lr_tfidf_score : 0.75
```

[ ]:
```
#Classification report for bag of words
lr_bow_report=classification_report(test_sentiments,lr_bow_predict,target_names=['Positive','N
print(lr_bow_report)

#Classification report for tfidf features
lr_tfidf_report=classification_report(test_sentiments,lr_tfidf_predict,target_names=['Positive
print(lr_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.75      0.75      0.75      4993
    Negative       0.75      0.75      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000

              precision    recall  f1-score   support

    Positive       0.74      0.77      0.75      4993
    Negative       0.76      0.73      0.75      5007

    accuracy                           0.75     10000
   macro avg       0.75      0.75      0.75     10000
weighted avg       0.75      0.75      0.75     10000
```

Logistic analysis shows around 75% accuracy for both bags of words and tf-idf data which is quite impressive!

1.3 SDGC: Stochastic gradient descent or Linear support vector machines for bag of words and tfidf features

SGDClassifier is a linear classifier in scikit-learn that uses stochastic gradient descent (SGD) as the optimization algorithm. It is a type of online learning algorithm that can handle large-scale datasets efficiently, by processing one instance at a time and updating the model parameters incrementally.

SGDClassifier can be used for binary classification, multi-class classification, and regression tasks. It supports a variety of loss functions, such as hinge loss (for linear SVM), log loss (for logistic regression), and squared loss (for linear regression). It also supports various regularization methods, such as L1 and L2 regularization, to prevent overfitting.

SGDClassifier can be a good choice for large datasets or streaming data, where batch learning algorithms may not be suitable due to memory constraints or processing time. However, it may require more hyperparameter tuning and preprocessing than other classifiers, as it is more sensitive to the scaling and distribution of the features.

[ ]:
```
#training the linear svm
svm=SGDClassifier(loss='hinge',max_iter=500,random_state=42)
#fitting the svm for bag of words
svm_bow=svm.fit(cv_train_reviews,train_sentiments)
print(svm_bow)
#fitting the svm for tfidf features
svm_tfidf=svm.fit(tv_train_reviews,train_sentiments)
print(svm_tfidf)
```

```
SGDClassifier(max_iter=500, random_state=42)
SGDClassifier(max_iter=500, random_state=42)
```

[ ]:
```
#Predicting the model for bag of words
svm_bow_predict=svm.predict(cv_test_reviews)
print(svm_bow_predict)
#Predicting the model for tfidf features
svm_tfidf_predict=svm.predict(tv_test_reviews)
print(svm_tfidf_predict)
```

```
['positive' 'positive' 'negative' … 'positive' 'positive' 'positive']
['positive' 'positive' 'positive' … 'positive' 'positive' 'positive']
```

[ ]:
```
#Accuracy score for bag of words
svm_bow_score=accuracy_score(test_sentiments,svm_bow_predict)
print("svm_bow_score :",svm_bow_score)
#Accuracy score for tfidf features
svm_tfidf_score=accuracy_score(test_sentiments,svm_tfidf_predict)
print("svm_tfidf_score :",svm_tfidf_score)
```

```
svm_bow_score : 0.5829
svm_tfidf_score : 0.5112
```

```
[ ]: #Classification report for bag of words
svm_bow_report=classification_report(test_sentiments,svm_bow_predict,target_names=['Positive',
print(svm_bow_report)
#Classification report for tfidf features
svm_tfidf_report=classification_report(test_sentiments,svm_tfidf_predict,target_names=['Positi
print(svm_tfidf_report)
```

```
              precision    recall  f1-score   support

    Positive       0.94      0.18      0.30      4993
    Negative       0.55      0.99      0.70      5007

    accuracy                           0.58     10000
   macro avg       0.74      0.58      0.50     10000
weighted avg       0.74      0.58      0.50     10000

              precision    recall  f1-score   support

    Positive       1.00      0.02      0.04      4993
    Negative       0.51      1.00      0.67      5007

    accuracy                           0.51     10000
   macro avg       0.75      0.51      0.36     10000
weighted avg       0.75      0.51      0.36     10000
```

Since random guess will have 50% accuracy, we can conclude that SDGC (bags of word accuracy: 58%, tf-idf accuracy: 51%) isn't a good model to implement.

1.4 XGboost

```
[ ]: train_sentiments_label = train_sentiments.apply(lambda x: 1 if x == 'positive'␣
    ↪else 0)
test_sentiments_label = test_sentiments.apply(lambda x: 1 if x == 'positive'␣
    ↪else 0)
train_sentiments_label.value_counts(), test_sentiments_label.value_counts()
```

```
[ ]: (0    20007
 1    19993
 Name: sentiment, dtype: int64,
 1    5007
 0    4993
 Name: sentiment, dtype: int64)
```

```
[ ]: import xgboost as xgb
```

```
[ ]: cv_classifier = xgb.XGBClassifier(max_depth = 7, eta = 0.9, objective= 'binary:
    ↪hinge', n_estimators = 200,
```

```
                                    use_label_encoder=False, eval_metric = 'auc')
tv_classifier = xgb.XGBClassifier(max_depth = 10, eta = 0.2, objective= 'binary:
 ↪hinge', n_estimators = 200,
                                    use_label_encoder=False, eval_metric = 'auc')

cv_bow = cv_classifier.fit(cv_train_reviews, train_sentiments_label)
cv_tfidf = tv_classifier.fit(tv_train_reviews, train_sentiments_label)
```

[ ]:
```
xgb_bow_pred = cv_classifier.predict(cv_test_reviews)
xgb_tfidf_pred = tv_classifier.predict(tv_test_reviews)

# evaluate predictions
cv_score = classification_report(xgb_bow_pred, test_sentiments_label,␣
 ↪target_names=['Positive','Negative'])
print(cv_score)

tv_score = classification_report(xgb_tfidf_pred, test_sentiments_label,␣
 ↪target_names=['Positive','Negative'])
print(tv_score)
```

```
              precision    recall  f1-score   support

    Positive       1.00      0.50      0.67      9972
    Negative       0.00      0.61      0.01        28

    accuracy                           0.50     10000
   macro avg       0.50      0.55      0.34     10000
weighted avg       1.00      0.50      0.66     10000

              precision    recall  f1-score   support

    Positive       0.00      0.00      0.00         0
    Negative       1.00      0.50      0.67     10000

    accuracy                           0.50     10000
   macro avg       0.50      0.25      0.33     10000
weighted avg       1.00      0.50      0.67     10000


/Users/swimmingcircle/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/swimmingcircle/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
```

```
  _warn_prf(average, modifier, msg_start, len(result))
/Users/swimmingcircle/opt/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1318: UndefinedMetricWarning: Recall
and F-score are ill-defined and being set to 0.0 in labels with no true samples.
Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

XGboost doesn't seem to be good at handling language transformed texts. It is very sensitive to parameter changes. It often predict the all the data into one class or another when changing the parameters. We conclude that it isn't a good model for our sentiment classification.

1.5 BERT model

1.5.1 What's special about BERT?

- Context-free models:generate a single word embedding representation for each word in the vocabulary, such as word2vec or GloVe. For example, the word "bank" would have the same representation in "bank deposit" and in "riverbank"
- Contextual models instead generate a representation of each word that is based on the other words in the sentence, such as BERT.

1.5.2 Understand how BERT works

1. Token embeddings: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. Segment embeddings: A marker indicating Sentence A or Sentence B is added to each token. **This allows the encoder to distinguish between sentences.**
3. Positional embeddings: A positional embedding is added to each token to indicate its position in the sentence.

1. Masked LM (MLM) The idea here is "simple": Randomly mask out 15% of the words in the input — replacing them with a [MASK] token. Loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones.

2. Next Sentence Prediction (NSP) In order to understand relationship between two sentences, BERT training process also uses next sentence prediction, BERT separates sentences with a special [SEP] token. During training the model is fed with two input sentences at a time such that:

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a a random sentence from the full corpus.

Example: predict if the next sentence is random or not

Important note: BERT does not try to predict the next word in the sentence!!

1.5.3 Tokenizer for BERT

BERT uses what is called a WordPiece tokenizer. It works by splitting words either into the full forms (e.g., one word becomes one token) or into word pieces — where one word can be broken into multiple tokens.

Word	Token(s)
surf	[‘surf’]
surfing	[‘surf’, ‘##ing’]
surfboarding	[‘surf’, ‘##board’, ‘##ing’]
surfboard	[‘surf’, ‘##board’]
snowboard	[‘snow’, ‘##board’]
snowboarding	[‘snow’, ‘##board’, ‘##ing’]
snow	[‘snow’]
snowing	[‘snow’, ‘##ing’]

By splitting words into word pieces, we have already identified that the words “surfboard” and “snowboard” share meaning through the wordpiece “##board” We have done this without even encoding our tokens or processing them in any way through BERT.

1.5.4 BERT model choice

BERT model we choose **DistilBERT** vs BERT - DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT’s performances as measured on the GLUE language understanding benchmark.

BERT-base vs BERT-large: BERT-based - BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters - BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

BERT-based-case vs **BERT-base-uncased**: - We don’t differentiate between cased and uncased data (english vs English)

1.5.5 BERT input

Input IDs – The input ids are often the only required parameters to be passed to the model as input. Token indices, numerical representations of tokens building the sequences that will be used as input by the model.

Attention mask – Attention Mask is used to avoid performing attention on padding token indices. Mask value can be either 0 or 1, 1 for tokens that are NOT MASKED, 0 for MASKED tokens.

Token type ids – It is used in use cases like sequence classification or question answering. As these require two different sequences to be encoded in the same input IDs. Special tokens, such as the classifier[CLS] and separator[SEP] tokens are used to separate the sequences.

Note: Padding is a special form of masking where the masked steps are at the start or the end of a sequence. Padding comes from the need to encode sequence data into contiguous batches: in order to make all sequences in a batch fit a given standard length, it is necessary to pad or truncate some sequences

1.5.6 BERT tokens

`CLS`: The [CLS] token, short for “classification,” is a special token used in BERT to represent the entire input sequence for classification tasks.

When training a classification model using BERT, the [CLS] token is added to the beginning of the input sequence, and the final hidden state corresponding to this token is used as the input to a classifier. This allows the model to make a prediction for the entire input sequence.

`SEP`: The [SEP] token, short for "separator," is used to separate two different segments of a sentence or document.

In BERT, the [SEP] token is used to separate the two segments when performing tasks like question answering or natural language inference, where the model needs to understand the relationship between two different segments of text.

`MASK`: [MASK] is used during pre-training to randomly mask some of the input tokens, forcing the model to learn to predict the masked tokens based on the surrounding context.

1.5.7 Understanding the parameters

`max_length` is a parameter used to define the maximum length of an input sequence.

`pad_to_max_length` is a Boolean parameter used to indicate whether sequences shorter than the max_length should be padded with a special token, usually [PAD], to make them the same length as the longest sequence in the batch.

`return_tensors` parameter specifies that we want the encoded data to be returned as TensorFlow tensor

`attention_mask`: 1 indicates a value that should be attended to, while 0 indicates a padded value.

Example:

`sequence_a = "This is a short sequence."     sequence_b = "This is a rather long sequence. It is at least longer than the sequence A."     len(encoded_sequence_a), len(encoded_sequence_b)`

(8, 19)

`padded_sequences = tokenizer([sequence_a, sequence_b], padding=True) padded_sequences["input_ids"]`

[[101, 1188, 1110, 170, 1603, 4954, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 1188, 1110, 170, 1897, 1263, 4954, 119, 1135, 1110, 1120, 1655, 2039, 1190, 1103, 4954, 138, 119, 102]]

`padded_sequences["attention_mask"]`

[[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

```
[ ]: #test with a smaller dataset
     imdb_data = imdb_data[:300]
```

```
[ ]: import transformers
     from tokenizers import BertWordPieceTokenizer
     # First load the real tokenizer
     tokenizer = transformers.DistilBertTokenizer.
      ↪from_pretrained('distilbert-base-uncased' , lower = True)
     # Save the loaded tokenizer locally
```

```
tokenizer.save_pretrained('.')
# Reload it with the huggingface tokenizers library
```

[ ]: Tokenizer(vocabulary_size=30522, model=BertWordPiece, unk_token=[UNK], sep_token=[SEP], cls_token=[CLS], pad_token=[PAD], mask_token=[MASK], clean_text=True, handle_chinese_chars=True, strip_accents=None, lowercase=True, wordpieces_prefix=##)

Resources - BERT Explained: A Complete Guide with Theory and Tutorial

[ ]:
```
# Encode the training data
encoded_train_data = tokenizer(train_reviews.values.tolist(), padding=True,␣
 ↪truncation=True, return_tensors='pt')
```

[ ]:
```
from torch.utils.data import TensorDataset, DataLoader
import torch
from tqdm import tqdm
from transformers import BertModel

# Load the pre-trained BERT model
bert_model = BertModel.from_pretrained('bert-base-uncased')

# Convert data to PyTorch tensors
input_ids = torch.tensor(encoded_train_data['input_ids'])
attention_masks = torch.tensor(encoded_train_data['attention_mask'])
train_sentiments = train_sentiments.apply(lambda x: 1 if x == 'positive' else 0)
labels = torch.tensor(train_sentiments.values.tolist())

# Create a TensorDataset
dataset = TensorDataset(input_ids, attention_masks, labels)

# Define batch size
batch_size = 8

# Create a DataLoader
dataloader = DataLoader(
    dataset,
    batch_size=batch_size,
    shuffle=True
)

# Iterate over batches
for batch in tqdm(dataloader):
    batch_input_ids = batch[0]
    batch_attention_masks = batch[1]
    batch_labels = batch[2]
    outputs = bert_model(batch_input_ids, attention_mask=batch_attention_masks)
```

```
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_8024/2238400034.py:7:
UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  input_ids = torch.tensor(encoded_train_data['input_ids'])
/var/folders/0h/xyv81g2n7sj6zr0c9cw30gkc0000gn/T/ipykernel_8024/2238400034.py:8:
UserWarning: To copy construct from a tensor, it is recommended to use
sourceTensor.clone().detach() or
sourceTensor.clone().detach().requires_grad_(True), rather than
torch.tensor(sourceTensor).
  attention_masks = torch.tensor(encoded_train_data['attention_mask'])
100%|     | 38/38 [05:53<00:00,  9.30s/it]
```

[ ]:

```

```

[ ]:

```

```

BERT_NLP_sentiment_analysis_of_movie_reviews_ipynb

March 4, 2023

1 Sentiment Analysis of IMDB Movie Reviews (Part 3: BERT model)

1.0.1 Important: Pre-trained transformer models like BERT from Hugging Face are designed to handle text in its raw form!

Side note: I was using pytorch BERT transfomer(`from transformers import BertModel`, `from transformers import BertTokenizer`) with data processing (e.g. stemming, strip html…). Yet, the model either doesn't run or take more than 1 hr because of the big dataset. Hence, I follow this tutorial to implement a BERT model.

1.1 BERT model

1.1.1 What's special about BERT?

- Context-free models:generate a single word embedding representation for each word in the vocabulary, such as word2vec or GloVe. For example, the word "bank" would have the same representation in "bank deposit" and in "riverbank"
- Contextual models instead generate a representation of each word that is based on the other words in the sentence, such as BERT.

1.1.2 Understand how BERT works

1. Token embeddings: A [CLS] token is added to the input word tokens at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
2. Segment embeddings: A marker indicating Sentence A or Sentence B is added to each token. **This allows the encoder to distinguish between sentences.**
3. Positional embeddings: A positional embedding is added to each token to indicate its position in the sentence.

1. Masked LM (MLM) The idea here is "simple": Randomly mask out 15% of the words in the input — replacing them with a [MASK] token. Loss function considers only the prediction of the masked tokens and ignores the prediction of the non-masked ones.

2. Next Sentence Prediction (NSP) In order to understand relationship between two sentences, BERT training process also uses next sentence prediction, BERT separates sentences with a special [SEP] token. During training the model is fed with two input sentences at a time such that:

- 50% of the time the second sentence comes after the first one.
- 50% of the time it is a a random sentence from the full corpus.

Example: predict if the next sentence is random or not

Important note: BERT does not try to predict the next word in the sentence!!

1.1.3 Tokenizer for BERT

BERT uses what is called a WordPiece tokenizer. It works by splitting words either into the full forms (e.g., one word becomes one token) or into word pieces — where one word can be broken into multiple tokens.

Word	Token(s)
surf	[‘surf’]
surfing	[‘surf’, ‘##ing’]
surfboarding	[‘surf’, ‘##board’, ‘##ing’]
surfboard	[‘surf’, ‘##board’]
snowboard	[‘snow’, ‘##board’]
snowboarding	[‘snow’, ‘##board’, ‘##ing’]
snow	[‘snow’]
snowing	[‘snow’, ‘##ing’]

By splitting words into word pieces, we have already identified that the words “surfboard” and “snowboard” share meaning through the wordpiece “##board” We have done this without even encoding our tokens or processing them in any way through BERT.

1.1.4 BERT model choice

BERT model we choose **DistilBERT** vs BERT - DistilBERT is a small, fast, cheap and light Transformer model trained by distilling BERT base. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT’s performances as measured on the GLUE language understanding benchmark.

BERT-base vs BERT-large: BERT-based - BERT-Base: 12-layer, 768-hidden-nodes, 12-attention-heads, 110M parameters - BERT-Large: 24-layer, 1024-hidden-nodes, 16-attention-heads, 340M parameters

BERT-based-case vs **BERT-base-uncased**: - We don’t differentiate between cased and uncased data (english vs English)

1.1.5 BERT input

Input IDs – The input ids are often the only required parameters to be passed to the model as input. Token indices, numerical representations of tokens building the sequences that will be used as input by the model.

Attention mask – Attention Mask is used to avoid performing attention on padding token indices. Mask value can be either 0 or 1, 1 for tokens that are NOT MASKED, 0 for MASKED tokens.

Token type ids – It is used in use cases like sequence classification or question answering. As these require two different sequences to be encoded in the same input IDs. Special tokens, such as the classifier[CLS] and separator[SEP] tokens are used to separate the sequences.

Note: Padding is a special form of masking where the masked steps are at the start or the end of a sequence. Padding comes from the need to encode sequence data into contiguous batches: in order to make all sequences in a batch fit a given standard length, it is necessary to pad or truncate some sequences

1.1.6 BERT tokens

`CLS`: The [CLS] token, short for "classification," is a special token used in BERT to represent the entire input sequence for classification tasks.

When training a classification model using BERT, the [CLS] token is added to the beginning of the input sequence, and the final hidden state corresponding to this token is used as the input to a classifier. This allows the model to make a prediction for the entire input sequence.

`SEP`: The [SEP] token, short for "separator," is used to separate two different segments of a sentence or document.

In BERT, the [SEP] token is used to separate the two segments when performing tasks like question answering or natural language inference, where the model needs to understand the relationship between two different segments of text.

`MASK`: [MASK] is used during pre-training to randomly mask some of the input tokens, forcing the model to learn to predict the masked tokens based on the surrounding context.

1.1.7 Understanding the parameters

`max_length` is a parameter used to define the maximum length of an input sequence.

`pad_to_max_length` is a Boolean parameter used to indicate whether sequences shorter than the max_length should be padded with a special token, usually [PAD], to make them the same length as the longest sequence in the batch.

`return_tensors` parameter specifies that we want the encoded data to be returned as TensorFlow tensor

`attention_mask`: 1 indicates a value that should be attended to, while 0 indicates a padded value.

Example:

```
sequence_a = "This is a short sequence."      sequence_b = "This is a rather long
sequence. It is at least longer than the sequence A."     len(encoded_sequence_a),
len(encoded_sequence_b)
```

(8, 19)

```
padded_sequences = tokenizer([sequence_a, sequence_b], padding=True)
padded_sequences["input_ids"]
```

[[101, 1188, 1110, 170, 1603, 4954, 119, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [101, 1188, 1110, 170, 1897, 1263, 4954, 119, 1135, 1110, 1120, 1655, 2039, 1190, 1103, 4954, 138, 119, 102]]

```
padded_sequences["attention_mask"]
```

[[1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

Note: We use huggingface model and setup packeages to have a clear view on the tranining process.

Resources - BERT Explained: A Complete Guide with Theory and Tutorial

1.1.8 BERT vs RoBERTa

RoBERTa model shares the same architecture as the BERT model. It is a reimplementation of BERT with some modifications to the key hyperparameters and minor embedding tweaks.

The key differences between RoBERTa and BERT can be summarized as follows:

- RoBERTa is a reimplementation of BERT with some modifications to the key hyperparameters and minor embedding tweaks. It uses a byte-level BPE as a tokenizer (similar to GPT-2) and a different pretraining scheme.
- RoBERTa is trained for longer sequences, too, i.e. the number of iterations is increased from 100K to 300K and then further to 500K.
- RoBERTa uses larger byte-level BPE vocabulary with 50K subword units instead of character-level BPE vocabulary of size 30K used in BERT.
- In the Masked Language Model (MLM) training objective, RoBERTa employs dynamic masking to generate the masking pattern every time a sequence is fed to the model.
- RoBERTa doesn't use token_type_ids, and we don't need to define which token belongs to which segment. Just separate segments with the separation token tokenizer.sep_token (or).
- The next sentence prediction (NSP) objective is removed from the training procedure.
- Larger mini-batches and learning rates are used in RoBERTa's training.

In the future, we can consider using RoBERTa because it is supposed to have better results.

[ ]:
```
import locale
def getpreferredencoding(do_setlocale = True):
    return "UTF-8"
locale.getpreferredencoding = getpreferredencoding
```

[ ]:
```
#make sure we are using GPU to run

import torch
torch.cuda.is_available()
```

[ ]: True

[ ]:
```
!pip install datasets
```

[ ]:
```
from datasets import load_dataset
imdb = load_dataset("imdb")
```

```
WARNING:datasets.builder:Found cached dataset imdb (/root/.cache/huggingface/dat
asets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f74830d1100e171db75bbdd
b80b3345c9c0)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
```

[ ]:
```
#take only 1000 training data but all of the testing data
```

```
small_train_dataset = imdb["train"].shuffle(seed=42).select([i for i in␣
↪list(range(1000))])
small_test_dataset = imdb["test"]
```

```
WARNING:datasets.arrow_dataset:Loading cached shuffled indices for dataset at /r
oot/.cache/huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded371
3f1f74830d1100e171db75bbddb80b3345c9c0/cache-9c48ce5d173413c7.arrow
```

[ ]:
```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-
packages (4.26.1)
ERROR: Operation cancelled by user
```

[ ]:
```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("distilbert-base-uncased")
def preprocess_function(examples):
   return tokenizer(examples["text"], truncation=True)

tokenized_train = small_train_dataset.map(preprocess_function, batched=True)
tokenized_test = small_test_dataset.map(preprocess_function, batched=True)
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
```

```
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
```

```
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}

loading file vocab.txt from cache at /root/.cache/huggingface/hub/models--
distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/vocab.txt
loading file tokenizer.json from cache at /root/.cache/huggingface/hub/models--
distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/tokenizer.json
loading file added_tokens.json from cache at None
loading file special_tokens_map.json from cache at None
loading file tokenizer_config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/tokenizer_config.json
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}

WARNING:datasets.arrow_dataset:Loading cached processed dataset at /root/.cache/
huggingface/datasets/imdb/plain_text/1.0.0/d613c88cf8fa3bab83b4ded3713f1f74830d1
100e171db75bbddb80b3345c9c0/cache-916b6147aa954289.arrow

Map:   0%|          | 0/25000 [00:00<?, ? examples/s]
```

[ ]:
```
from transformers import DataCollatorWithPadding
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

from transformers import AutoModelForSequenceClassification
model = AutoModelForSequenceClassification.
 ↪from_pretrained("distilbert-base-uncased", num_labels=2)


import numpy as np
from datasets import load_metric

def compute_metrics(eval_pred):
   load_accuracy = load_metric("accuracy")
   load_f1 = load_metric("f1")

   logits, labels = eval_pred
   predictions = np.argmax(logits, axis=-1)
   accuracy = load_accuracy.compute(predictions=predictions,␣
 ↪references=labels)["accuracy"]
   f1 = load_f1.compute(predictions=predictions, references=labels)["f1"]
   return {"accuracy": accuracy, "f1": f1}
```

```
loading configuration file config.json from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/config.json
Model config DistilBertConfig {
  "_name_or_path": "distilbert-base-uncased",
  "activation": "gelu",
  "architectures": [
    "DistilBertForMaskedLM"
  ],
  "attention_dropout": 0.1,
  "dim": 768,
  "dropout": 0.1,
  "hidden_dim": 3072,
  "initializer_range": 0.02,
  "max_position_embeddings": 512,
  "model_type": "distilbert",
  "n_heads": 12,
  "n_layers": 6,
  "pad_token_id": 0,
  "qa_dropout": 0.1,
  "seq_classif_dropout": 0.2,
  "sinusoidal_pos_embds": false,
  "tie_weights_": true,
  "transformers_version": "4.26.1",
  "vocab_size": 30522
}
```

```
loading weights file pytorch_model.bin from cache at
/root/.cache/huggingface/hub/models--distilbert-base-
uncased/snapshots/1c4513b2eedbda136f57676a34eea67aba266e5c/pytorch_model.bin
Some weights of the model checkpoint at distilbert-base-uncased were not used
when initializing DistilBertForSequenceClassification:
['vocab_layer_norm.weight', 'vocab_projector.bias', 'vocab_layer_norm.bias',
'vocab_projector.weight', 'vocab_transform.weight', 'vocab_transform.bias']
- This IS expected if you are initializing DistilBertForSequenceClassification
from the checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model from a
BertForPreTraining model).
- This IS NOT expected if you are initializing
DistilBertForSequenceClassification from the checkpoint of a model that you
expect to be exactly identical (initializing a BertForSequenceClassification
model from a BertForSequenceClassification model).
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.weight', 'pre_classifier.weight', 'classifier.bias',
'pre_classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[ ]:
```python
from transformers import TrainingArguments, Trainer

repo_name = "finetuning-sentiment-model-1000-samples"

training_args = TrainingArguments(
   output_dir=repo_name,
   learning_rate=2e-5,
   per_device_train_batch_size=16,
   per_device_eval_batch_size=16,
   num_train_epochs=2,
   weight_decay=0.01,
   save_strategy="epoch",
)

trainer = Trainer(
   model=model,
   args=training_args,
   train_dataset=tokenized_train,
   eval_dataset=tokenized_test,
   tokenizer=tokenizer,
   data_collator=data_collator,
   compute_metrics=compute_metrics,
)
```

```
PyTorch: setting up devices
```

```
The default value for the training argument `--report_to` will change in v5
(from all installed integrations to none). In v5, you will need to use
`--report_to all` to get the same behavior as now. You should start updating
your code and make this info disappear :-).
```

It seems that the defaul loss function is already cross entropy loss: here.

```
[ ]: trainer.train()
```

```
The following columns in the training set don't have a corresponding argument in
`DistilBertForSequenceClassification.forward` and have been ignored: text. If
text are not expected by `DistilBertForSequenceClassification.forward`,  you can
safely ignore this message.
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306:
FutureWarning: This implementation of AdamW is deprecated and will be removed in
a future version. Use the PyTorch implementation torch.optim.AdamW instead, or
set `no_deprecation_warning=True` to disable this warning
  warnings.warn(
***** Running training *****
  Num examples = 1000
  Num Epochs = 2
  Instantaneous batch size per device = 16
  Total train batch size (w. parallel, distributed & accumulation) = 16
  Gradient Accumulation steps = 1
  Total optimization steps = 126
  Number of trainable parameters = 66955010
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.
```

```
<IPython.core.display.HTML object>
```

```
Saving model checkpoint to finetuning-sentiment-model-3000-samples/checkpoint-63
Configuration saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/config.json
Model weights saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-
model-3000-samples/checkpoint-63/special_tokens_map.json
Saving model checkpoint to finetuning-sentiment-
model-3000-samples/checkpoint-126
Configuration saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/config.json
Model weights saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/pytorch_model.bin
tokenizer config file saved in finetuning-sentiment-
model-3000-samples/checkpoint-126/tokenizer_config.json
Special tokens file saved in finetuning-sentiment-
```

```
model-3000-samples/checkpoint-126/special_tokens_map.json

Training completed. Do not forget to share your model on huggingface.co/models
=)
```

[ ]:
```
TrainOutput(global_step=126, training_loss=0.4624747018965464,
metrics={'train_runtime': 105.1295, 'train_samples_per_second': 19.024,
'train_steps_per_second': 1.199, 'total_flos': 263009880425280.0, 'train_loss':
0.4624747018965464, 'epoch': 2.0})
```

[ ]:
```
trainer.evaluate()
```

```
The following columns in the evaluation set don't have a corresponding argument
in `DistilBertForSequenceClassification.forward` and have been ignored: text. If
text are not expected by `DistilBertForSequenceClassification.forward`,  you can
safely ignore this message.
***** Running Evaluation *****
  Num examples = 25000
  Batch size = 16

<IPython.core.display.HTML object>
```

[ ]:
```
{'eval_loss': 0.308576762676239,
 'eval_accuracy': 0.87956,
 'eval_f1': 0.8788476240292923,
 'eval_runtime': 441.2864,
 'eval_samples_per_second': 56.653,
 'eval_steps_per_second': 3.542,
 'epoch': 2.0}
```

Even though we only run 1000 training data with 2 epochs, when we test on all the testing dataset, we achieve 88% accuracy! We can try to use more datapoints when we have sufficient computational power or customize the BERT model to improve accuracy.

Resources: - Sentiment Analysis of IMDB Movie Reviews - Sentiment Analysis - Cleaning,EDA & BERT(88% Acc) - Text Classification with Movie Reviews - NLP - Data Preprocessing and Cleaning - Getting Started with Sentiment Analysis using Python - BERT Explained: A Complete Guide with Theory and Tutorial