Homework Assignment 2
Semantic Analysis via Text Classification

Data Processing

Reading a few records so that we don't explode the RAM.

```
import pandas as pd
```

```
amazon_df=pd.read_csv('./NLP/2/Reviews.csv',nrows=10)
```

```
amazon_df.head()
```

Id	ProductId	UserId	ProfileName	HelpfulnessNumerator	HelpfulnessDenominator	Score	Time	Summary	Text
1	B001E4KFG0	A3SGXH7AUHU8GW	delmartian	1	1	5	1303862400	Good Quality Dog Food	I have bought several of the Vitality canned d...
2	B00813GRG4	A1D87F6ZCVE5NK	dll pa	0	0	1	1346976000	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...
3	B000LQOCH0	ABXLMWJIXXAIN	Natalia Corres "Natalia Corres"	1	1	4	1219017600	"Delight" says it all	This is a confection that has been around a fe...
4	B000UA0QIQ	A395BORC6FGVXV	Karl	3	3	2	1307923200	Cough Medicine	If you are looking for the secret ingredient i...
5	B006K2ZZ7K	A1UQRSCLF8GW1T	Michael D. Bigham "M. Wassir"	0	0	5	1350777600	Great taffy	Great taffy at a great price. There was a wid...

Reading the required columns and splitting the data based on Score. The Review is tagged as positive if it is more than 3 and is negative if it is less than 3 and ignored when the score is 3.

```
amazon_df=pd.read_csv('./NLP/2/Reviews.csv',usecols=['UserId','Score','Text'])
```

```
amazon_df.shape
```

```
(568454, 3)
```

```
amazon_df_negative=amazon_df[amazon_df['Score']<3]
amazon_df_positive=amazon_df[amazon_df['Score']>3]
del amazon_df
print(f'{amazon_df_negative.shape=},{amazon_df_positive.shape=}')
```

```
amazon_df_negative.shape = (82037, 3), amazon_df_positive.shape = (443777, 3)
```

Checking if we have duplicate data points. We will remove if we have duplicates as it won't add more information to our traditional model and since we have a good number of samples we can still run Deep Learning models like BERT as well after removing the duplicates.

```
print('Duplicates in Negative DF : ',amazon_df_negative[amazon_df_negative.duplicated()].shape,'\nDuplicates in Positive DF : ',amazon_df_positive[amazon_df_positive.duplicated()].
```

```
Duplicates in Negative DF :  (24948, 3)
Duplicates in Positive DF :  (136946, 3)
```

Sanity checks if we still have duplicates. Then compare how much data we are removing from the original data.

```
amazon_df_negative=amazon_df_negative.drop_duplicates()
amazon_df_positive=amazon_df_positive.drop_duplicates()
print('Duplicates in Negative DF : ',amazon_df_negative[amazon_df_negative.duplicated()].shape,'\nDuplicates in Positive DF : ',amazon_df_positive[amazon_df_positive.duplicated()]
print(f'{amazon_df_negative.shape=},{amazon_df_positive.shape=}')
```

```
Duplicates in Negative DF :  (0, 3)
Duplicates in Positive DF :  (0, 3)
amazon_df_negative.shape = (57089, 3), amazon_df_positive.shape = (306831, 3)
```

```
print('Records Dropped in Positive Class : #',136946,' i.e ',int((136946/443777)*100),'% of total data',sep='')
print('Records Dropped in Negative Class : #',24948,' i.e ',int((24948/82037)*100),'% of total data',sep='')
```

```
Records Dropped in Positive Class : #136946 i.e 30% of total data
Records Dropped in Negative Class : #24948 i.e 30% of total data
```

Having too many reviews from a single user can bias the model so it's better to have a single review from each user so that we don't explicitly add bias to the model and at the same time assume that doing this will help deal with bias.

```
1  amazon_df_negative['UserId'].value_counts()
```

```
AKZKG2Z7CNV27     35
A2M9D9BDH0NV3Y    30
A2MUGFV2TDQ47K    29
A2TN9C5E4A0I3F    26
A2XNJJ9TF70P4J    25
                  ..
A3FG03RM9AZNKP     1
A238V1MWVLQMSS     1
A2GXHMP9KHS3IX     1
ARK3R8WOCZMI2      1
A3I8AFVPEE8KI5     1
Name: UserId, Length: 48889, dtype: int64
```

```
1  amazon_df_negative[amazon_df_negative['UserId']=='A2M9D9BDH0NV3Y']['Score'].value_counts()
```

```
1    16
2    14
Name: Score, dtype: int64
```

```
1  amazon_df_negative[amazon_df_negative['UserId']=='A2XNJJ9TF70P4J']['Score'].value_counts()
```

```
1    20
2     5
Name: Score, dtype: int64
```

```
1  amazon_df_positive['UserId'].value_counts()
```

```
AY12DBB0U420B     305
A30XHLG6DIBRW8    251
A281NPSIMI1C2R    224
A1YUL9PCJR3JTY    184
A1Z54EM24Y40LL    174
                 ...
A21D27IDM2MDP3      1
AT8D1BM87NCRV       1
A12EXIL2QAU5KV      1
A25NBFSL0NC04K      1
A3LGQPJCZVL9UC      1
Name: UserId, Length: 207065, dtype: int64
```

```
1  amazon_df_positive[amazon_df_positive['UserId']=='AY12DBB0U420B'].head()
```

	UserId	Score	Text
109	AY12DBB0U420B	5	I'm presently on a diet and I was at my Fresh ...
2196	AY12DBB0U420B	4	A nearby Fresh and Easy Neighborhood Market st...
3016	AY12DBB0U420B	4	When I'm working, I always carry my own lunch....
3528	AY12DBB0U420B	4	In my trip through the health area of my local...
3826	AY12DBB0U420B	5	I've always liked licorice candy and the best ...

```
1  amazon_df_positive[amazon_df_positive['UserId']=='AY12DBB0U420B']['Score'].value_counts()
```

```
5    246
4     59
Name: Score, dtype: int64
```

```
1  amazon_df_positive[amazon_df_positive['UserId']=='A1Z54EM24Y40LL']['Score'].value_counts()
```

```
5    135
4     39
Name: Score, dtype: int64
```

Here we are grouping based on UserId and then taking only 1 sample per user as we discussed the reasons above and then noting down how much data we have retained from the original dataset. Later we will be only retaining the required columns as it will not only reduce the size of the dataframe in our RAM but will also be helpful as having too many unnecessary columns can make the script confusing.

```
# Since all the reviews are positive we will consider only 1 review for each UserId
amazon_df_positive=amazon_df_positive.groupby(by=['UserId']).head(1)

# Since all the reviews are negative we will consider only 1 review for each UserId
amazon_df_negative=amazon_df_negative.groupby(by=['UserId']).head(1)

print(f'{amazon_df_negative.shape=},{amazon_df_positive.shape=}')
```

```
amazon_df_negative.shape = (48889, 3), amazon_df_positive.shape = (207065, 3)
```

```
print('Records Retained in Positive Class : #',207065,' i.e ',int((207065/443777)*100),'% of total data',sep='')
print('Records Retained in Negative Class : #',48889,' i.e ',int((48889/82037)*100),'% of total data',sep='')
```

```
Records Retained in Positive Class : #207065 i.e 46% of total data
Records Retained in Negative Class : #48889 i.e 59% of total data
```

```
amazon_df_positive=amazon_df_positive[['Text','Score']]
amazon_df_negative=amazon_df_negative[['Text','Score']]
```

Here we are trying to fetch the number of reviews per score. The aim is to have equal reviews per score so that we don't explicitly bias the dataset for a particular score as it may make the model lean more towards the score which has higher samples. For a better model, it's always better to have the dataset balanced.

```
amazon_df_positive['Score'].value_counts()
```

```
5    173629
4     33436
Name: Score, dtype: int64
```

```
amazon_df_negative['Score'].value_counts()
```

```
1    31568
2    17321
Name: Score, dtype: int64
```

Since we are allowed to use 10-20% of the original number of samples. This not only helps to get the script running quickly and easily but also saves a lot of hardware resources. Additionally, running deep learning models like BERT would be more challenging given the size of the dataset as we need to fine-tune the model. Lastly, we will be taking an equal number of samples from each score so that the data remains balanced.

```
[ ]  1  print('We are allowed to work on 10-20% of whole data which means #',int((568454*0.15)),' can be the total records we can use to work on out of #568,454!',sep='')
```

```
We are allowed to work on 10-20% of whole data which means #85268 can be the total records we can use to work on out of #568,454!
```

```
[ ]  1  print('#Records from Negative Class (1=17321 & 2=17321) and #Records from Positive Negative Class (4=17321 & 5=17321) gives a total of #',(17321*4),'!',sep='')
```

```
#Records from Negative Class (1 = 17321 & 2 = 17321) and #Records from Positive Negative Class (4 = 17321 & 5 = 17321) gives a total of #69284!
```

```
1  amazon_df_negative_1=amazon_df_negative[amazon_df_negative['Score']==1].sample(n=17321,random_state=123)
2  amazon_df_negative_2=amazon_df_negative[amazon_df_negative['Score']==2]
3
4  amazon_df_positive_4=amazon_df_positive[amazon_df_positive['Score']==4].sample(n=17321,random_state=123)
5  amazon_df_positive_5=amazon_df_positive[amazon_df_positive['Score']==5].sample(n=17321,random_state=123)
6
7  amazon_df_negative_1['Label']=-1
8  amazon_df_negative_2['Label']=-1
9  amazon_df_positive_4['Label']=1
10 amazon_df_positive_5['Label']=1
11
12 amazon_df_negative_1=amazon_df_negative_1[['Text','Label']]
13 amazon_df_negative_2=amazon_df_negative_2[['Text','Label']]
14 amazon_df_positive_4=amazon_df_positive_4[['Text','Label']]
15 amazon_df_positive_5=amazon_df_positive_5[['Text','Label']]
16
17 print(f'{amazon_df_negative_1.shape=},\n{amazon_df_negative_2.shape=},\n{amazon_df_positive_4.shape=},\n{amazon_df_positive_5.shape=}')
```

```
amazon_df_negative_1.shape = (17321, 2),
amazon_df_negative_2.shape = (17321, 2),
amazon_df_positive_4.shape = (17321, 2),
amazon_df_positive_5.shape = (17321, 2)
```

```
1  amazon_df=pd.concat(objs=[amazon_df_negative_1,amazon_df_negative_2,amazon_df_positive_4,amazon_df_positive_5])
2  amazon_df=amazon_df.sample(frac=1,random_state=123)
3  print(f'{amazon_df.shape=}')
```

```
amazon_df.shape = (69284, 2)
```

Here we are printing the top 5 records from the final dataset.

```
1  amazon_df.head(5).to_dict(orient='records')
```

```
[{'Text': 'This coffee is alright. The taste is rather sourish and a bit bitter. It just does not taste very good. Maybe its because I am used to Lavazza and Sturbucks beans, but thats how I feel about it. I tried making it in my drip coffee maker (I have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre. I ended up giving this coffee away.',
  'Label': -1},
 {'Text': "These caramels are insanely delicious, and unlike any I've tried before. Seriously. They are rich, buttery, the slight crunch from the salt just knocks them out of the ballpark, flavor-wise. I bought 3 bags, kept one and gave away 2...now I wish I had kept them all! They are fabulous!!!!",
  'Label': 1},
 {'Text': 'Kalvert Rose Syrup exceeded my expectations for flavor and aroma.  I use it in ice cream and in to sweeten some teas.',
  'Label': 1},
 {'Text': 'The package says it is a resealable zipper, but it does not fit well. You have to put it in the proper storage area to keep it fresh.',
  'Label': 1},
 {'Text': 'I stopped buying these when I noticed they now contain "PGPR".  (Google it)<br /><br />I now buy only Pearson\'s mint patties.',
  'Label': -1}]
```

Here we are printing the last 5 records from the final dataset.

```
1  amazon_df.tail(5).to_dict(orient='records')
```

```
[{'Text': 'My 3.5 yr old daughter with Autism loves these! She is on a special gluten-free, casein-free and soy-free diet and these are a wonderful snack for her.',
  'Label': 1},
 {'Text': 'The Kelloggs Muselix are delicious and the delivery was quick.<a href="http://www.amazon.com/gp/product/B001E6KBSK">Kellogg\'s Crispy Blend Mueslix Raisins, Dates & Almonds, 15.3-Ounce Unit (Pack of 5)</a>',
  'Label': 1},
 {'Text': 'I decided to give these a try.  The lids are VERY difficult to put on and I often ended up bending the cup while trying to affix the lid.  They do not seal perfectly and I often found unexpected puddles, but they do the job.  Since I have a Keurig permanant filter I thought of just buying the paper filters to save on cleanup.  For almost $11 from this vendor, you can get 50 (fifty) paper filters.....I would just assume use paper towels.  That works out to be just under 25 cents for a small piece of paper....DisposaKups, please re-evaluate your pricing!',
  'Label': -1},
 {'Text': 'I like McDougall\'s soups and was taken by the concept that they<br />are virtually non fat and provide fiber etc I am most disappointed<br />to find the manufacturer attempts deception of the consumer by<br />listing the nutritional values for one serving, whereas, in fact,<br />the values are for half a package. McDougall has lost all credibility<br />for me and I will not be purchasing again. It is offensive to treat<br />consumers in this way.<a href="http://www.amazon.com/gp/product/B0017U08M2">Dr. McDougall\'s Right Foods Vegan Split Pea Soup, Lower Sodium, 1.9-Ounce Cups (Pack of 6)</a>',
  'Label': -1},
 {'Text': 'My dog has still not figured out what to do with this.  He is a very smart dog with a vocab of about 50 words.  I was surprised how hard the plastic is and am also worried. Once he does figure out how it works I am afraid he will knock himself out when he shakes it from side to side! Let alone what will happen if it flies out of his mouth!',
  'Label': -1}]
```

Checking the distribution of Labels from the final dataset.

```
1   amazon_df['Label'].value_counts()
```

```
-1    34642
 1    34642
Name: Label, dtype: int64
```

Sanity checks in the final dataset before we proceed to modelling. We can see that we are only using 1.1MB of RAM space.

```
1   amazon_df.isnull().sum()
```

```
Text      0
Label     0
dtype: int64
```

```
1  amazon_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 69284 entries, 0 to 69283
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Text    69284 non-null  object
 1   Label   69284 non-null  int64 
dtypes: int64(1), object(1)
memory usage: 1.1+ MB
```

We will be using a great Python package BeautifulSoup to remove HTML tags. Additionally, we will be using a regular expression package (re) to clean text reviews.

```
1  from bs4 import BeautifulSoup
```

```
1  t1='I like McDougall\'s soups and was taken by the concept that they<br />are virtually non fat and provide fiber etc I am most disappointed<br />to find the manufacturer attempts
```

```
1  t1
```

```
'I like McDougall\'s soups and was taken by the concept that they<br />are virtually non fat and provide fiber etc I am most disappointed<br />to find the manufacturer attempts deceptio
n of the consumer by<br />listing the nutritional values for one serving, whereas, in fact,<br />the values are for half a package. McDougall has lost all credibility<br />for me and I
will not be purchasing again. It is offensive to treat<br />consumers in this way.<a href="http://www.amazon.com/gp/product/B0017U08M2">Dr. McDougall\'s Right Foods Vegan Split Pea Soup
, Lower Sodium, 1.9-Ounce Cups (Pack of 6)</a>'
```

```
1  BeautifulSoup(t1,"lxml").get_text()
```

```
'I like McDougall's soups and was taken by the concept that theyare virtually non fat and provide fiber etc I am most disappointedto find the manufacturer attempts deception of the cons
umer bylisting the nutritional values for one serving, whereas, in fact,the values are for half a package. McDougall has lost all credibilityfor me and I will not be purchasing again. I
t is offensive to treatconsumers in this way.Dr. McDougall's Right Foods Vegan Split Pea Soup, Lower Sodium, 1.9-Ounce Cups (Pack of 6)'
```

```
1  import re
```

```
1  t2=re.compile(pattern=r'([a-z]+)\.+').sub(repl=r'\1 . ',string=str(t1).lower()).strip()
2  t2=re.compile(pattern=r'\<.*?\>').sub(repl=r' ',string=str(t2).lower()).strip()
```

```
1  t2
```

```
'i like mcdougall's soups and was taken by the concept that they are virtually non fat and provide fiber etc i am most disappointed to find the manufacturer attempts deception of the co
nsumer by listing the nutritional values for one serving, whereas, in fact, the values are for half a package .  mcdougall has lost all credibility for me and i will not be purchasing a
gain .  it is offensive to treat consumers in this way .  dr .  mcdougall's right foods vegan split pea soup, lower sodium, 1.9-ounce cups (pack of 6)'
```

```
1  BeautifulSoup(t2,"lxml").get_text()
```

```
'i like mcdougall's soups and was taken by the concept that they are virtually non fat and provide fiber etc i am most disappointed to find the manufacturer attempts deception of the co
nsumer by listing the nutritional values for one serving, whereas, in fact, the values are for half a package .  mcdougall has lost all credibility for me and i will not be purchasing a
gain .  it is offensive to treat consumers in this way .  dr .  mcdougall's right foods vegan split pea soup, lower sodium, 1.9-ounce cups (pack of 6)'
```

We will define functions to process and the clean review column and then plot the word cloud for visualization purposes.

```
from collections import Counter

def join_pre_process_text(text:list=[])->str:
  return ' '.join(text)

def pre_process_text(text:str='',remove_punct=False)->list:
  text=re.compile(pattern=r'([a-z]+)\.+').sub(repl=r'\1 . ',string=str(text).lower())
  text=re.compile(pattern=r'\<.*?\>').sub(repl=r' ',string=str(text))
  if remove_punct:
    text=re.compile(pattern=r'[^a-z\d]+').sub(repl=r' ',string=str(text))
  text=re.compile(pattern=r'\s+').sub(repl=r' ',string=str(text)).strip()
  return text.split(' ')

def get_word_freq(df)->dict:
  all_tokens=[pre_process_text(text=review_1,remove_punct=True) for review_1 in df]
  all_tokens_flat=[token for sublist in all_tokens for token in sublist]
  word_freq=Counter(all_tokens_flat)
  return word_freq
```

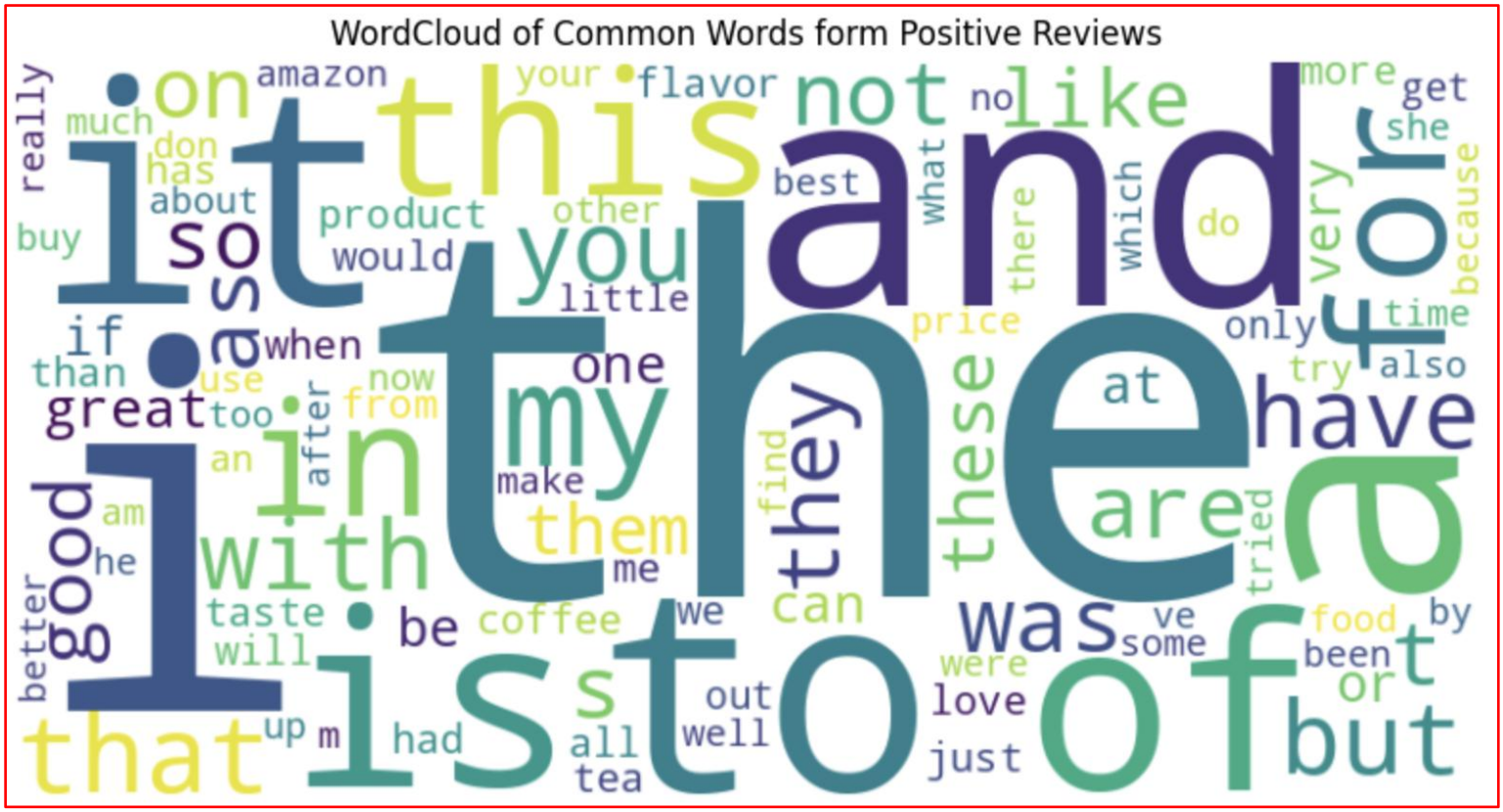
```
# for collection
word_freq_collection=get_word_freq(df=amazon_df['Text'])

# for positive reviews
word_freq_positive=get_word_freq(df=amazon_df[amazon_df['Label']==1]['Text'])

# for positive reviews
word_freq_negative=get_word_freq(df=amazon_df[amazon_df['Label']==-1]['Text'])

# filter only top 100 most common words
top_100_collection=word_freq_collection.most_common(100)
top_100_positive=word_freq_positive.most_common(100)
top_100_negative=word_freq_negative.most_common(100)
```

Creating a word cloud for the whole corpus using the top 100 words.

```
from wordcloud import WordCloud

# visualize common words using WordCloud for collection
wordcloud=WordCloud(width=800,height=400,background_color='white').generate_from_frequencies(dict(top_100_collection))

plt.figure(figsize=(10,5))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Common Words form Collection')
plt.show()
```

Creating a word cloud for the positive reviews using the top 100 words.

```
# visualize common words using WordCloud for Positive Reviews
wordcloud=WordCloud(width=800,height=400,background_color='white').generate_from_frequencies(dict(top_100_positive))

plt.figure(figsize=(10,5))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Common Words form Positive Reviews')
plt.show()
```



Creating a word cloud for the negative reviews using the top 100 words.

```
# visualize common words using WordCloud for Negative Reviews
wordcloud=WordCloud(width=800,height=400,background_color='white').generate_from_frequencies(dict(top_100_negative))

plt.figure(figsize=(10,5))
plt.imshow(wordcloud,interpolation='bilinear')
plt.axis('off')
plt.title('WordCloud of Common Words form Negative Reviews')
plt.show()
```

Testing the functions and then running it on the whole dataset and printing the first record.

```
amazon_df['Text'].iloc[0]
```

```
'This coffee is alright. The taste is rather sourish and a bit bitter. It just does not taste very good. Maybe its because I am used to Lavazza and Sturbucks beans, but thats how I feel about it. I tried making it in my drip coffee maker (I have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre. I ended up giving this coffee away.'
```

```
join_pre_process_text(text=pre_process_text(text=amazon_df['Text'].iloc[0],remove_punct=False))
```

```
'this coffee is alright . the taste is rather sourish and a bit bitter . it just does not taste very good . maybe its because i am used to lavazza and sturbucks beans, but thats how i feel about it . i tried making it in my drip coffee maker (i have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre . i ended up giving this coffee away .'
```

```
amazon_df['Text'].iloc[-2]
```

```
'I like McDougall\'s soups and was taken by the concept that they<br />are virtually non fat and provide fiber etc I am most disappointed<br />to find the manufacturer attempts deception of the consumer by<br />listing the nutritional values for one serving, whereas, in fact,<br />the values are for half a package. McDougall has lost all credibility<br />for me and I will not be purchasing again. It is offensive to treat<br />consumers in this way.<a href="http://www.amazon.com/gp/product/B0017U08M2">Dr. McDougall\'s Right Foods Vegan Split Pea Soup, Lower Sodium, 1.9-Ounce Cups (Pack of 6)</a>'
```

```
join_pre_process_text(text=pre_process_text(text=amazon_df['Text'].iloc[-2],remove_punct=False))
```

```
'i like mcdougall's soups and was taken by the concept that they are virtually non fat and provide fiber etc i am most disappointed to find the manufacturer attempts deception of the consumer by listing the nutritional values for one serving, whereas, in fact, the values are for half a package . mcdougall has lost all credibility for me and i will not be purchasing again . it is offensive to treat consumers in this way . dr . mcdougall's right foods vegan split pea soup, lower sodium, 1.9-ounce cups (pack of 6)'
```

```
amazon_df['Cleaned_Text']=amazon_df['Text'].apply(lambda x: join_pre_process_text(text=pre_process_text(text=x,remove_punct=True)))
```

```
amazon_df.head(1).to_dict(orient='records')
```

```
[{'Text': 'This coffee is alright. The taste is rather sourish and a bit bitter. It just does not taste very good. Maybe its because I am used to Lavazza and Sturbucks beans, but thats how I feel about it. I tried making it in my drip coffee maker (I have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre. I ended up giving this coffee away.',
  'Label': -1,
  'Cleaned_Text': 'this coffee is alright the taste is rather sourish and a bit bitter it just does not taste very good maybe its because i am used to lavazza and sturbucks beans but thats how i feel about it i tried making it in my drip coffee maker i have a very good coffee maker that releases most flavor and french press still it tasted mediocre i ended up giving this coffee away'}]
```

Splitting the dataset so that the same can be used for every model.

```
from sklearn.model_selection import train_test_split
```

```
# Split data into training and testing sets
X_train,X_test,y_train,y_test=train_test_split(amazon_df['Cleaned_Text'],amazon_df['Label'],test_size=0.2,random_state=1234)
del train_test_split
```

```
print(f'{X_train.shape=},\n{X_test.shape=},\n{y_train.shape=},\n{y_test.shape=}')
```

```
X_train.shape = (55427,),
X_test.shape = (13857,),
y_train.shape = (55427,),
y_test.shape = (13857,)
```

Task 1 - Re-do the TF-IDF approach for this task.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

Traditional TF-IDF Approach

```
# 1. Extraction of features using TF-IDF (Term Frequency & Inverse Document Frequency)
tfidf_vectorizer=TfidfVectorizer(use_idf=True,max_features=2000)
X_train_tfidf=tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf=tfidf_vectorizer.transform(X_test)
del tfidf_vectorizer,TfidfVectorizer
```

```
print(f'{X_train_tfidf.shape=},\n{X_test_tfidf.shape=},\n{type(X_train_tfidf)=},\n{type(X_test_tfidf)=}')
```

```
X_train_tfidf.shape = (55427, 2000),
X_test_tfidf.shape = (13857, 2000),
type(X_train_tfidf) = <class 'scipy.sparse._csr.csr_matrix'>,
type(X_test_tfidf) = <class 'scipy.sparse._csr.csr_matrix'>
```

```
from sklearn.linear_model import LogisticRegression,SGDClassifier,RidgeClassifier,PassiveAggressiveClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
import time
```

Defining evaluation function.

```
def get_evaluation_metrics(true_pred,test_pred):
  precision=precision_score(true_pred,test_pred)
  recall=recall_score(true_pred,test_pred)
  accuracy=accuracy_score(true_pred,test_pred)
  f1_scr=f1_score(true_pred,test_pred)
  return precision,recall,accuracy,f1_scr
```

Defining function to train the traditional model.

```
def train_model(xtrain_records,ytrain_records,xtest_records,ytest_records)->list:
  # training different models

  nb_model=MultinomialNB()
  logistic_model=LogisticRegression()
  ridge_classifier_model=RidgeClassifier()
  passive_aggressive_model=PassiveAggressiveClassifier()
  nearest_centroid_model=NearestCentroid()
  sgd_model=SGDClassifier()
  rf_model=RandomForestClassifier()

  method_names=list()
  precs=list()
  recal=list()
  acc=list()
  f1s=list()

  print('Models Running : ',time.ctime())
  for model1,model_name in [
    (nb_model,"Naive Bayes"),
    (logistic_model,"Logistic Regression"),
    (ridge_classifier_model,"Ridge Classifier"),
    (passive_aggressive_model,"Passive Aggressive Classifier"),
    (nearest_centroid_model,"Nearest Centroid"),
    (sgd_model,"Stochastic Gradient Descent"),
    (rf_model,"Random Forest"),
    ]:
      try:
        model1.fit(xtrain_records,ytrain_records)
        p,r,a,f=get_evaluation_metrics(true_pred=ytest_records,test_pred=model1.predict(xtest_records))
        method_names.append(model_name)
        precs.append(p)
        recal.append(r)
        acc.append(a)
        f1s.append(f)
        print(model_name,time.ctime())
      except Exception as e:
        print('Error -> ',e,'===',model_name,time.ctime())
  return method_names,precs,recal,acc,f1s
```

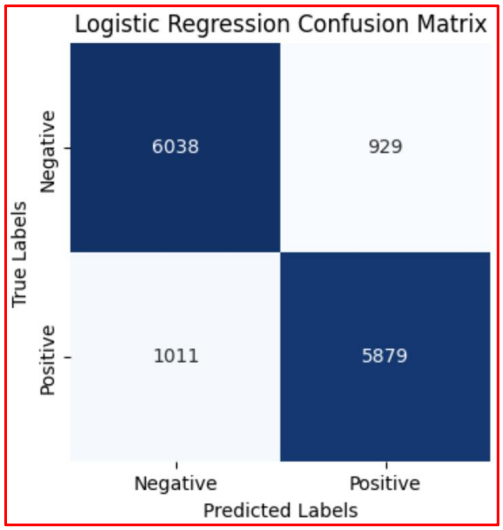
Running traditional models that were used in the last assignment and printing the results.

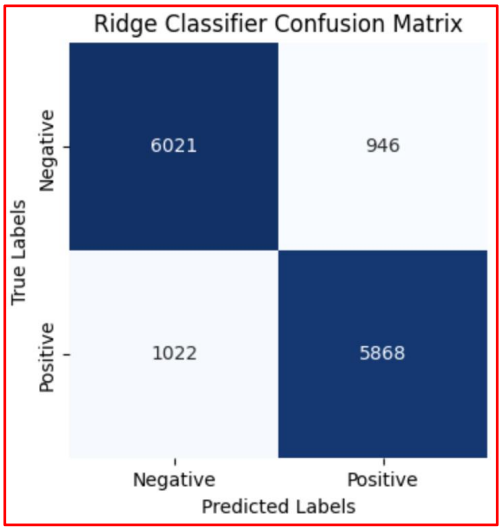
```
method_names_list,precision_list,recall_list,accuracy_list,f1_list=train_model(xtrain_records=X_train_tfidf,ytrain_records=y_train,xtest_records=X_test_tfidf,ytest_records=y_test)
```

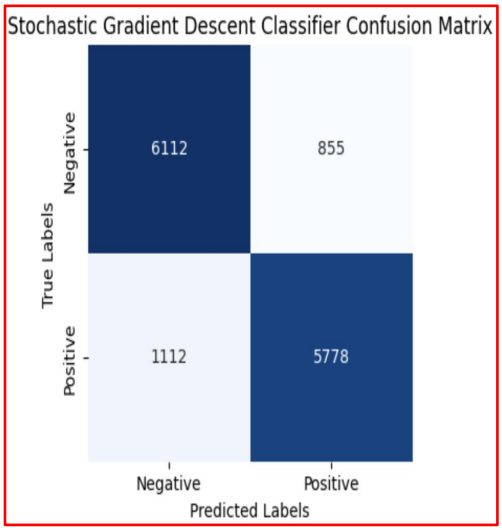
```
Naive Bayes Sat Mar 16 20:57:04 2024
Logistic Regression Sat Mar 16 20:57:04 2024
Ridge Classifier Sat Mar 16 20:57:05 2024
Passive Aggressive Classifier Sat Mar 16 20:57:06 2024
Nearest Centroid Sat Mar 16 20:57:06 2024
Stochastic Gradient Descent Sat Mar 16 20:57:06 2024
Random Forest Sat Mar 16 20:57:07 2024
```

	Method	Precision	Recall	Accuracy	F1
1	TF-IDF -> Logistic Regression	0.863543	0.853266	0.859999	0.858373
2	TF-IDF -> Ridge Classifier	0.861168	0.851669	0.857978	0.856392
5	TF-IDF -> Stochastic Gradient Descent	0.860223	0.851234	0.857256	0.855705
0	TF-IDF -> Naive Bayes	0.833406	0.832075	0.833802	0.83274
6	TF-IDF -> Random Forest	0.838896	0.802612	0.825215	0.820353
3	TF-IDF -> Passive Aggressive Classifier	0.874486	0.771553	0.831349	0.819801
4	TF-IDF -> Nearest Centroid	0.778441	0.775617	0.778668	0.777027

Printing the confusion matrix for the top 3 methods where the results are sorted based on the F1 score.







Task 2 - Review classification by using word2vec.

```
import gensim
```

```
# load Google's pre-trained Word2Vec model.
pre_w2v_model=gensim.models.KeyedVectors.load_word2vec_format('./snow_white_database/text_generation/GoogleNews-vectors-negative300.bin.gz',binary=True)
```

```
print(type(pre_w2v_model))
```

```
<class 'gensim.models.keyedvectors.KeyedVectors'>
```

```
amazon_df.head(1).to_dict(orient='records')
```

```
[{'Text': 'This coffee is alright. The taste is rather sourish and a bit bitter. It just does not taste very good. Maybe its because I am used to Lavazza and Sturbucks beans, but thats how I feel about it. I tried making it in my drip coffee maker (I have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre. I ended up giving this coffee away.',
  'Label': -1,
  'Cleaned_Text': 'this coffee is alright the taste is rather sourish and a bit bitter it just does not taste very good maybe its because i am used to lavazza and sturbucks beans but thats how i feel about it i tried making it in my drip coffee maker i have a very good coffee maker that releases most flavor and french press still it tasted mediocre i ended up giving this coffee away'}]
```

```
'customizeddsfsdfsd' in pre_w2v_model,'custom' in pre_w2v_model
```

```
(False, True)
```

We can see that we don't have embedding for the letter "a" but have all the remaining letters in the Google News vector model and the vector size is 300 dimensions.

```
'A' in pre_w2v_model,'a' in pre_w2v_model
```

```
(True, False)
```

```
pre_w2v_model['coffee'].shape
```

```
(300,)
```

```
for i in range(97,123):
    l1=chr(i)
    print(l1,l1 in pre_w2v_model)
```

```
a False
b True
c True
d True
e True
f True
g True
h True
i True
j True
k True
l True
m True
n True
o True
p True
q True
r True
s True
t True
u True
v True
w True
x True
y True
z True
```

Here we defined a function to get embedding for our text. Since we don't have all the words from our review in our model we are getting character level embedding and then taking an average of the embedding which gets embedding for each word. For example for the text "tea is life" we will get the embedding for the letter "t" then "e" then "a" and since each of these are 300 dimensions we will take an average of this and thus we get the embedding for tea and the process continues for other words and then finally once we have embedding for all the words we again take average so that our final embedding vector for the text is also 300 dimension as this makes sure that even if we have a short or a very long text the final embedding for each text will be 300 dimension only.

```
def get_mean_embedding(array_for_mean:list)->np.array:
  return np.mean(array_for_mean,axis=0)

def get_w2v_embedding(model_name,text)->list:
  word_embedding=list()
  for word1 in text.split(' '):
    if word1 not in model_name:
      char_level_embed=list()
      for char1 in word1:
        if char1=='a':
          char1='A'
        char_level_embed.append(model_name[char1])
      word_embedding.append(get_mean_embedding(array_for_mean=char_level_embed))
    else:
      word_embedding.append(model_name[word1])
  return get_mean_embedding(array_for_mean=word_embedding)
```

We tested a sample to check if the final output is 300 dimensions or not before we apply it to data else the time and memory would be wasted if the function doesn't work as we discussed above.

```
1  amazon_df.iloc[0]['Cleaned_Text']
```

```
'this coffee is alright the taste is rather sourish and a bit bitter it just does not taste very good maybe its because i am used to lavazza and sturbucks beans but thats how i feel abo
ut it i tried making it in my drip coffee maker i have a very good coffee maker that releases most flavor and french press still it tasted mediocre i ended up giving this coffee away'
```

```
1  get_w2v_embedding(model_name=pre_w2v_model,text='this coffee is alright the taste').shape
```

```
(300,)
```

```
1  get_w2v_embedding(model_name=pre_w2v_model,text=amazon_df.iloc[0]['Cleaned_Text']).shape
```

```
(300,)
```

Here we apply our model and generate a train and test matrix.

```
1  X_train_w2v=get_w2v_for_df(df=X_train,w2v_model=pre_w2v_model)
2  X_test_w2v=get_w2v_for_df(df=X_test,w2v_model=pre_w2v_model)
3  print(f'{X_train_w2v.shape=},\n{X_test_w2v.shape=},\n{type(X_train_w2v)=},\n{type(X_test_w2v)=}')
```

```
X_train_w2v.shape = (55427, 300),
X_test_w2v.shape = (13857, 300),
type(X_train_w2v) = <class 'numpy.ndarray'>,
type(X_test_w2v) = <class 'numpy.ndarray'>
```

Here we are running all the traditional models using the embedding vectors as features.

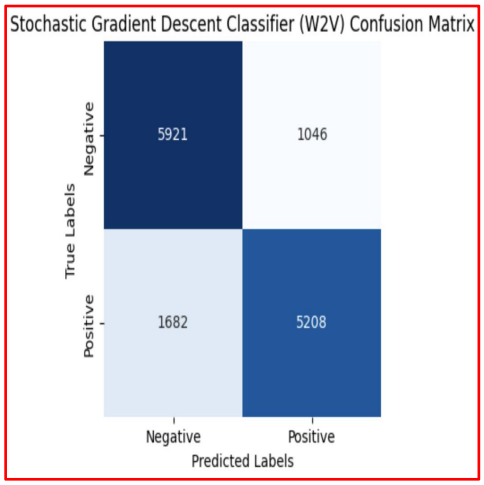
```
method_names_list_w2v,precision_list_w2v,recall_list_w2v,accuracy_list_w2v,f1_list_w2v=train_model(xtrain_records=X_train_w2v,ytrain_records=y_train,xtest_records=X_test_w2v,ytest_
```

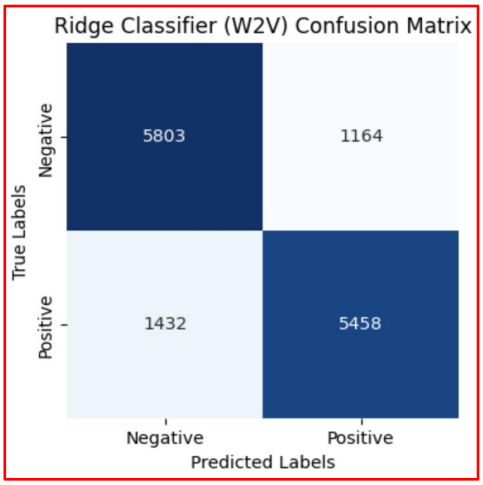
```
Models Running :  Sat Mar 16 21:46:11 2024
Error ->  Negative values in data passed to MultinomialNB (input X) === Naive Bayes Sat Mar 16 21:46:11 2024
Logistic Regression Sat Mar 16 21:46:17 2024
Ridge Classifier Sat Mar 16 21:46:19 2024
Passive Aggressive Classifier Sat Mar 16 21:46:20 2024
Nearest Centroid Sat Mar 16 21:46:20 2024
Stochastic Gradient Descent Sat Mar 16 21:46:21 2024
Random Forest Sat Mar 16 21:49:34 2024
```

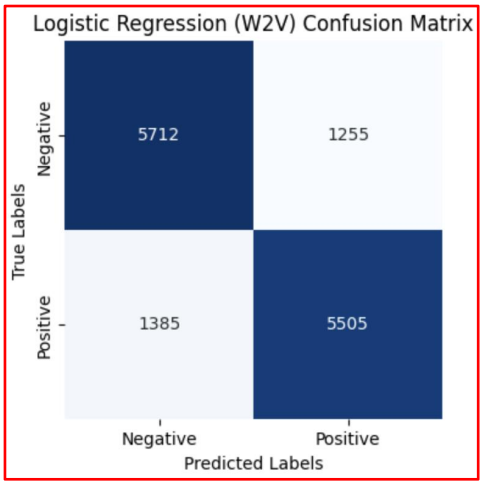
The final results for the above model are printed.

	Method	Precision	Recall	Accuracy	F1
4	Word2Vec -> Stochastic Gradient Descent	0.786134	0.834398	0.804792	0.809547
1	Word2Vec -> Ridge Classifier	0.824222	0.792163	0.812658	0.807874
0	Word2Vec -> Logistic Regression	0.814349	0.798984	0.809483	0.806593
2	Word2Vec -> Passive Aggressive Classifier	0.819223	0.78926	0.808617	0.803962
5	Word2Vec -> Random Forest	0.777251	0.761684	0.772967	0.769389
3	Word2Vec -> Nearest Centroid	0.730082	0.682293	0.716605	0.705379

Printing the confusion matrix for the top 3 methods where the results are sorted based on the F1 score.







Task 3 - BERT (without fine-tuning) for review classification.

BERT Approach without fine tuning

```
def check_max_length_of_review(df)->int:
  temp_df=df.copy().to_frame()
  temp_df['Len']=temp_df['Cleaned_Text'].str.len()
  return max(temp_df['Len'])
```

```
print('Max Length of Review in Xtrain : ',check_max_length_of_review(df=X_train),' & Max Length of Review in X_test : ',check_max_length_of_review(df=X_test))
```

```
Max Length of Review in Xtrain :  9027  & Max Length of Review in X_test :  10874
```

```
del check_max_length_of_review
```

```
!pip install -qq transformers
```

```
from transformers import pipeline
```

```
bert_classifier=pipeline("sentiment-analysis")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision af0f99b (https://huggingface.co/distilbert/distilbert-ba
Using a pipeline without specifying a model name and revision in production is not recommended.
config.json: 100% 629/629 [00:00<00:00, 30.7kB/s]
model.safetensors: 100% 268M/268M [00:02<00:00, 99.8MB/s]
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 2.66kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 3.27MB/s]
```

```
bert_classifier("We are very happy to show you the 🤗 Transformers library.")
```

```
[{'label': 'POSITIVE', 'score': 0.9997795224189758}]
```

```
amazon_df.head(3).to_dict(orient='records')
```

```
[{'Text': 'This coffee is alright. The taste is rather sourish and a bit bitter. It just does not taste very good. Maybe its because I am used to Lavazza and Sturbucks beans, but thats how I feel about it. I tried making it in my drip coffee maker (I have a very good coffee maker that releases most flavor) and french press, still it tasted mediocre. I ended up giving this coffee away.',
  'Label': -1,
  'Cleaned_Text': 'this coffee is alright the taste is rather sourish and a bit bitter it just does not taste very good maybe its because i am used to lavazza and sturbucks beans but thats how i feel about it i tried making it in my drip coffee maker i have a very good coffee maker that releases most flavor and french press still it tasted mediocre i ended up giving this coffee away'},
 {'Text': "These caramels are insanely delicious, and unlike any I've tried before. Seriously. They are rich, buttery, the slight crunch from the salt just knocks them out of the ballpark, flavor-wise. I bought 3 bags, kept one and gave away 2...now I wish I had kept them all! They are fabulous!!!!",
  'Label': 1,
  'Cleaned_Text': 'these caramels are insanely delicious and unlike any i ve tried before seriously they are rich buttery the slight crunch from the salt just knocks them out of the ballpark flavor wise i bought 3 bags kept one and gave away 2 now i wish i had kept them all they are fabulous'},
 {'Text': 'Kalvert Rose Syrup exceeded my expectations for flavor and aroma.  I use it in ice cream and in to sweeten some teas.',
  'Label': 1,
  'Cleaned_Text': 'kalvert rose syrup exceeded my expectations for flavor and aroma i use it in ice cream and in to sweeten some teas'}]
```

```
bert_classifier(amazon_df.iloc[0]['Cleaned_Text'])
```

```
[{'label': 'NEGATIVE', 'score': 0.9956425428390503}]
```

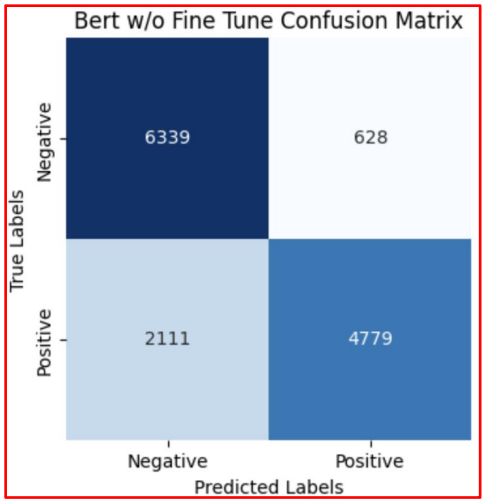
```
bert_classifier(amazon_df.iloc[1]['Cleaned_Text'])
```

```
[{'label': 'POSITIVE', 'score': 0.9991422891616821}]
```

```
bert_classifier(amazon_df.iloc[1]['Cleaned_Text'])[0]['label']
```

```
'POSITIVE'
```

Printing the confusion matrix.



Task 4 - BERT (with fine-tuning) for review classification.

BERT Approach with fine tuning

```
!pip install -qq transformers tensorflow
```

```
!pip install -qq tabulate
from tabulate import tabulate
# https://github.com/astanin/python-tabulate/blob/master/tabulate/__init__.py
```

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

```
X_train=pd.read_csv('./NLP/2/X_train.csv')
X_test=pd.read_csv('./NLP/2/X_test.csv')
y_train=pd.read_csv('./NLP/2/y_train.csv')
y_test=pd.read_csv('./NLP/2/y_test.csv')
y_train_bert_f_tune=np.where(y_train['Label']==-1,0,1)
y_test_bert_f_tune=np.where(y_test['Label']==-1,0,1)
```

```
import tensorflow as tf
from transformers import TFDistilBertForSequenceClassification,DistilBertTokenizer

bert_model=TFDistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased-finetuned-sst-2-english")
bert_tokenizer=DistilBertTokenizer.from_pretrained("distilbert-base-uncased-finetuned-sst-2-english")
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 629/629 [00:00<00:00, 18.9kB/s]
model.safetensors: 100% 268M/268M [00:03<00:00, 87.7MB/s]
All PyTorch model weights were used when initializing TFDistilBertForSequenceClassification.

All the weights of TFDistilBertForSequenceClassification were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDistilBertForSequenceClassification for predictions without further training.
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 720B/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 2.97MB/s]
```

```
# https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english
# max_seq_length
max_review_length=128
train_encodings=bert_tokenizer(list(X_train['Cleaned_Text'].values),truncation=True,padding=True,max_length=max_review_length)
test_encodings=bert_tokenizer(list(X_test['Cleaned_Text'].values),truncation=True,padding=True,max_length=max_review_length)
```

```
train_dataset=tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    # y_train.values
    y_train_bert_f_tune
))
test_dataset=tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    # y_test.values
    y_test_bert_f_tune
))
```

```
print(len(train_dataset),len(test_dataset))
```

```
55427 13857
```

```
# 85% of the data for training
train_size=int(0.85 * len(train_dataset))
train_dataset_2=train_dataset.take(train_size)
val_dataset=train_dataset.skip(train_size)
```

```
print(len(train_dataset),len(test_dataset),len(train_dataset_2),len(val_dataset))
```

```
55427 13857 47112 8315
```

```
bert_batch_size=16
bert_epochs=4
bert_model.compile(optimizer='Adam',loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),metrics=[tf.metrics.SparseCategoricalAccuracy('accuracy')])
```

```python
# Fine-tune the model
bert_history=bert_model.fit(train_dataset_2.shuffle(len(train_dataset_2)).batch(bert_batch_size),
          epochs=bert_epochs,
          batch_size=bert_batch_size,
          validation_data=val_dataset.shuffle(len(val_dataset)).batch(bert_batch_size),
                            )
```

```
Epoch 1/4
WARNING:tensorflow:AutoGraph could not transform <function infer_framework at 0x7b00e2564040> and will run it as-is.
Cause: for/else statement not yet supported
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
WARNING: AutoGraph could not transform <function infer_framework at 0x7b00e2564040> and will run it as-is.
Cause: for/else statement not yet supported
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert
2945/2945 [==============================] - 779s 238ms/step - loss: 0.6960 - accuracy: 0.4940 - val_loss: 0.6947 - val_accuracy: 0.4987
Epoch 2/4
2945/2945 [==============================] - 679s 230ms/step - loss: 0.6939 - accuracy: 0.4991 - val_loss: 0.6940 - val_accuracy: 0.4988
Epoch 3/4
2945/2945 [==============================] - 671s 228ms/step - loss: 0.6934 - accuracy: 0.4987 - val_loss: 0.6933 - val_accuracy: 0.4992
Epoch 4/4
2945/2945 [==============================] - 672s 228ms/step - loss: 0.6970 - accuracy: 0.4992 - val_loss: 0.6931 - val_accuracy: 0.4996
```

```python
test_loss,test_accuracy=bert_model.evaluate(test_dataset.batch(bert_batch_size))
print(f"Test Loss: {test_loss},Test Accuracy: {test_accuracy}")
```

```
4/4 [==============================] - 22s 5s/step - loss: 0.6578 - accuracy: 0.5834
Test Loss: 0.6578419208526611, Test Accuracy: 0.5833970308303833
```

```python
pred_list=bert_model.predict(test_dataset.batch(bert_batch_size))
```

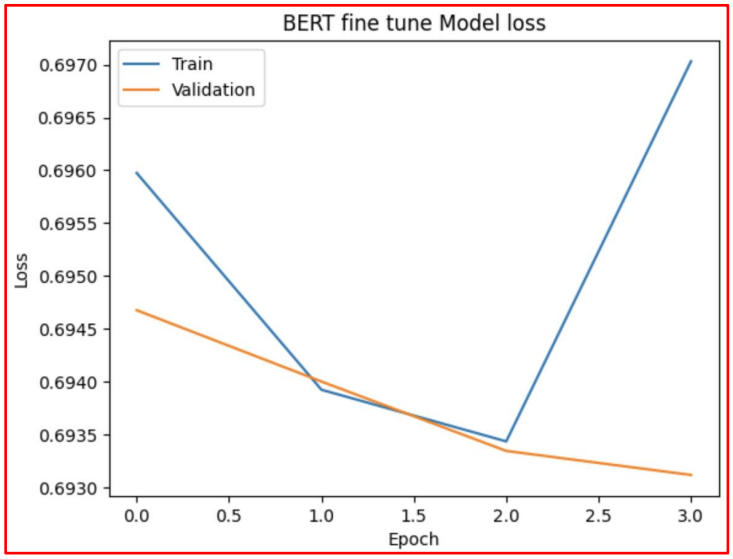
```
4/4 [==============================] - 14s 2s/step
```

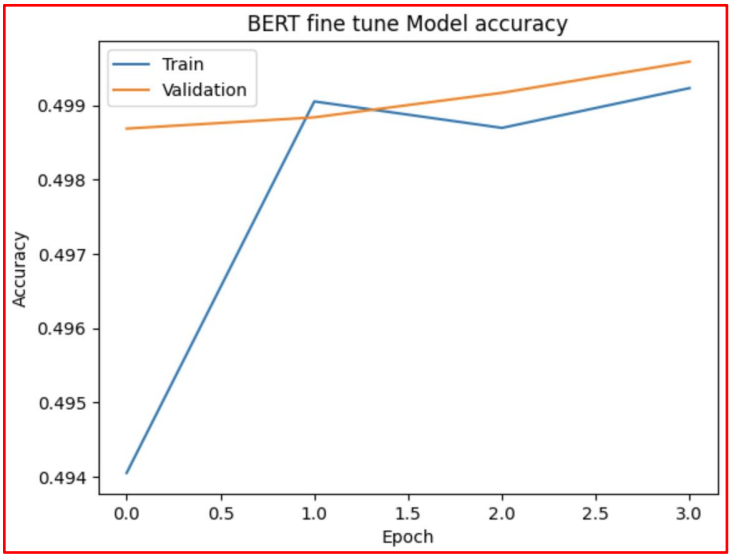
```python
np.where(pred_list.logits[:,0] > 0,-1,1)
```

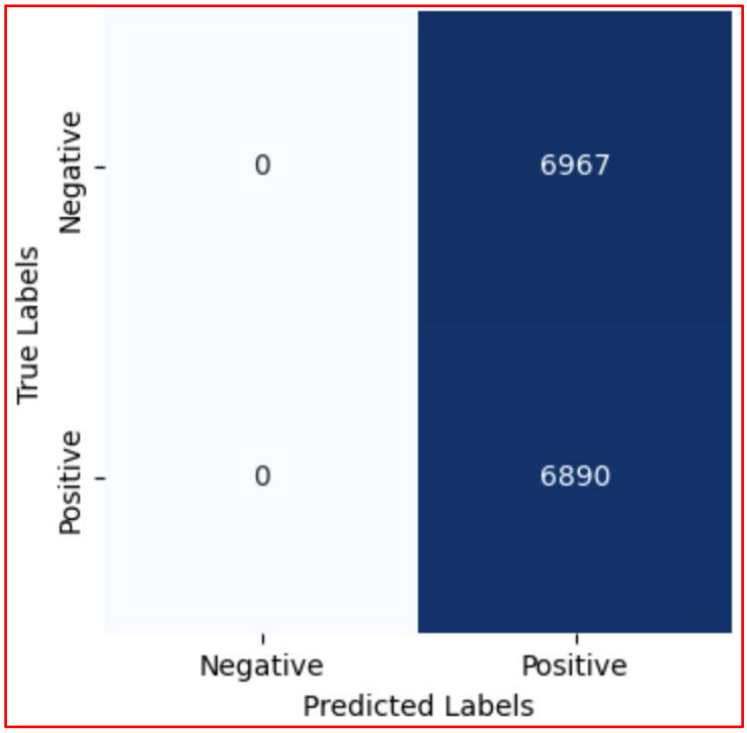
```
array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])
```

```python
bert_classifier_pred_with_fine_tune=bert_model.predict(test_dataset.batch(bert_batch_size))
bert_classifier_pred_with_fine_tune=np.argmax(bert_classifier_pred_with_fine_tune.logits,axis=1)
```

```
867/867 [==============================] - 67s 74ms/step
```






True Labels
Negative
Positive
0
6967
0
6890
Negative
Positive
Predicted Labels

Task 5 - BERT (with LoRA) for review classification.

```
bert_model=BertForSequenceClassification.from_pretrained("distilbert-base-uncased",num_labels=2)
bert_tokenizer=AutoTokenizer.from_pretrained("distilbert-base-uncased")
bert_model=get_peft_model(bert_model,LoraConfig(task_type=TaskType.SEQ_CLS,inference_mode=False,r=1,lora_alpha=1,lora_dropout=0.1,target_modules=["key","query","value"],))
print(bert_model.print_trainable_parameters())
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:88: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
config.json: 100% 483/483 [00:00<00:00, 24.3kB/s]
You are using a model of type distilbert to instantiate a model of type bert. This is not supported for all configurations of models and can yield errors.
model.safetensors: 100% 268M/268M [00:00<00:00, 303MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.bias', 'classifie
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 1.23kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 7.17MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 21.3MB/s]
trainable params: 56,834 || all params: 109,540,612 || trainable%: 0.05188395332317479
None
```

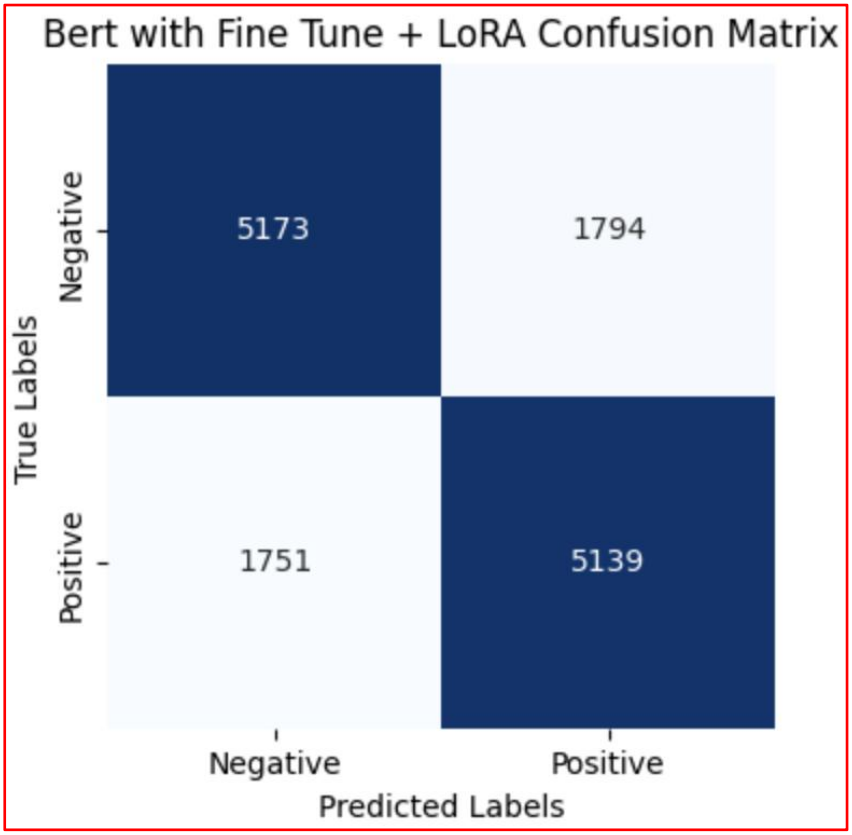
```
X_train=pd.read_csv('./NLP/2/X_train.csv')
X_test=pd.read_csv('./NLP/2/X_test.csv')
y_train=pd.read_csv('./NLP/2/y_train.csv')
y_test=pd.read_csv('./NLP/2/y_test.csv')
y_train=np.where(y_train['Label']==-1,0,1)
y_test=np.where(y_test['Label']==-1,0,1)

# Split data into training and evaluation datasets
X_train,X_eval,y_train,y_eval=train_test_split(X_train,y_train,test_size=0.15,random_state=123)
print(f'{X_train.shape=},\n{y_train.shape=},\n{X_eval.shape=},\n{y_eval.shape=},\n{X_test.shape=},\n{y_test.shape=}')

# Tokenize input data
X_train=bert_tokenizer(list(X_train['Cleaned_Text'].values),padding=True,truncation=True,return_tensors="pt")
X_eval=bert_tokenizer(list(X_eval['Cleaned_Text'].values),padding=True,truncation=True,return_tensors="pt")
X_test=bert_tokenizer(list(X_test['Cleaned_Text'].values),padding=True,truncation=True,return_tensors="pt")

# Convert labels to PyTorch tensors
y_train=torch.tensor(y_train)
y_eval=torch.tensor(y_eval)
y_test=torch.tensor(y_test)

# Define training dataset
train_dataset=torch.utils.data.TensorDataset(X_train['input_ids'],X_train['attention_mask'],y_train)
eval_dataset=torch.utils.data.TensorDataset(X_eval['input_ids'],X_eval['attention_mask'],y_eval)
test_dataset=torch.utils.data.TensorDataset(X_test['input_ids'],X_test['attention_mask'],y_test)

# Define training arguments
training_args=TrainingArguments(
    output_dir='./NLP/2/',
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    logging_dir='/NLP/2/logs/',
)
```

```python
# Define Trainer with custom data collator
class CustomDataCollator:
    def __call__(self,features):
        input_ids=torch.stack([f[0] for f in features])
        attention_mask=torch.stack([f[1] for f in features])
        labels=torch.tensor([f[2] for f in features])
        return {"input_ids": input_ids,"attention_mask": attention_mask,"labels": labels}

trainer=Trainer(
    model=bert_model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
    data_collator=CustomDataCollator()
)
```

```
X_train.shape = (47112, 1),
y_train.shape = (47112,),
X_eval.shape = (8315, 1),
y_eval.shape = (8315,),
X_test.shape = (13857, 1),
y_test.shape = (13857,)
/usr/local/lib/python3.10/dist-packages/accelerate/accelerator.py:432: FutureWarning: Passing t
dataloader_config = DataLoaderConfiguration(dispatch_batches=None, split_batches=False, even_ba
  warnings.warn(
```

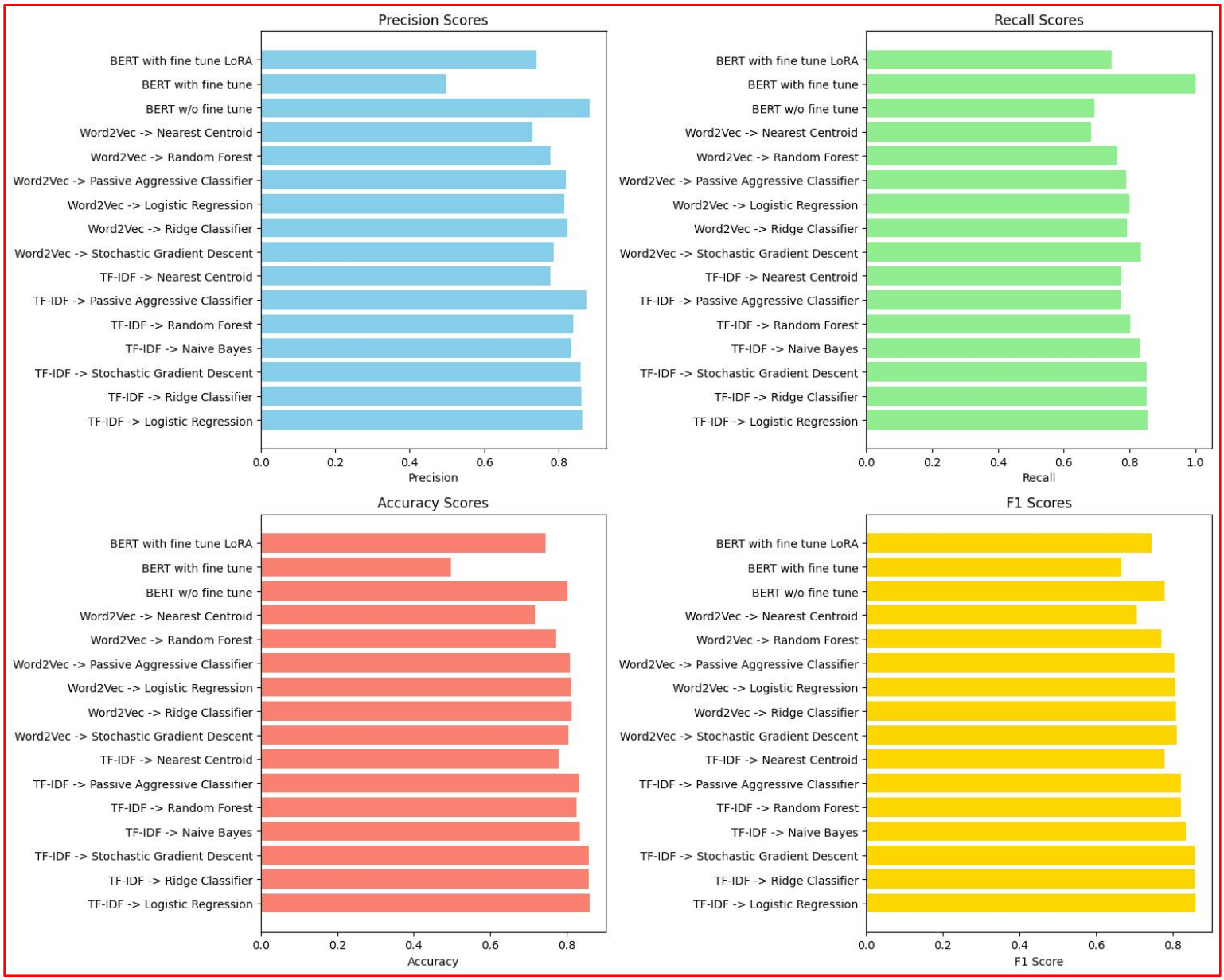
```python
# Fine-tune the model
trainer.train()
```

13000	0.535400
13500	0.517600
14000	0.535200
14500	0.524100
15000	0.546800
15500	0.505000
16000	0.505300
16500	0.527000
17000	0.522600
17500	0.515600



	Method	Precision	Recall	Accuracy	F1	Method Rank
0	TF-IDF -> Logistic Regression	0.8635	0.8533	0.86	0.8584	0
1	TF-IDF -> Ridge Classifier	0.8612	0.8517	0.858	0.8564	0
2	TF-IDF -> Stochastic Gradient Descent	0.8602	0.8512	0.8573	0.8557	0
3	TF-IDF -> Naive Bayes	0.8334	0.8321	0.8338	0.8327	0
4	TF-IDF -> Random Forest	0.8389	0.8026	0.8252	0.8204	0
5	TF-IDF -> Passive Aggressive Classifier	0.8745	0.7716	0.8313	0.8198	0
6	TF-IDF -> Nearest Centroid	0.7784	0.7756	0.7787	0.777	0
7	Word2Vec -> Stochastic Gradient Descent	0.7861	0.8344	0.8048	0.8095	1
8	Word2Vec -> Ridge Classifier	0.8242	0.7922	0.8127	0.8079	1
9	Word2Vec -> Logistic Regression	0.8143	0.799	0.8095	0.8066	1
10	Word2Vec -> Passive Aggressive Classifier	0.8192	0.7893	0.8086	0.804	1
11	Word2Vec -> Random Forest	0.7773	0.7617	0.773	0.7694	1
12	Word2Vec -> Nearest Centroid	0.7301	0.6823	0.7166	0.7054	1
13	BERT w/o fine tune	0.8839	0.6936	0.8023	0.7773	2
14	BERT with fine tune	0.4972	1	0.4972	0.6642	3
15	BERT with fine tune LoRA	0.7412	0.7459	0.7442	0.7435	4

Task 6 - Results Analysis.



The evaluation of different models based on various scoring metrics reveals some insightful patterns

- ✓ ***Precision Score:*** The BERT model exhibits impressive precision, even without fine-tuning, closely followed by the TF-IDF feature approach with classifiers like Passive Aggressive, Logistic Regression, and Ridge Classifier. However, when BERT undergoes fine-tuning, its precision score drops comparatively.
- ✓ ***Recall Score:*** Fine-tuned BERT stands out with the highest recall score, indicating its effectiveness in identifying relevant instances. Additionally, the TF-IDF approach combined with Logistic Regression, Ridge Classifier, and Stochastic Gradient Descent performs commendably. Conversely, recall scores are notably lower for BERT without fine-tuning and for the Word2Vec feature approach alongside Nearest Centroid.
- ✓ ***Accuracy Score:*** The TF-IDF feature approach, especially when paired with Logistic Regression, Ridge Classifier, and Stochastic Gradient Descent, yields the highest accuracy scores. Conversely, the Word2Vec approach with Nearest Centroid and BERT without fine-tuning exhibit relatively lower accuracy scores.
- ✓ ***F1 Score:*** Similar to recall, fine-tuned BERT achieves the highest F1 score, indicating a harmonious balance between precision and recall. Furthermore, TF-IDF with Logistic Regression, Ridge Classifier, and Stochastic Gradient Descent consistently demonstrates robust F1 scores. However, BERT without fine-tuning and Word2Vec with Nearest Centroid show comparatively weaker F1 scores.

So, while BERT models showcase varying performances based on fine-tuning, TF-IDF paired with different classifiers consistently performs well across precision, recall, accuracy, and F1 scores.