**MINOR PROJECT CODE/ALGORITHM**

Submitted in partial fulfillment for the award of the degree of

**BACHELOR OF TECHNOLOGY**

**(Department of Computer Science and Engineering)**

Submitted to

**INDIAN INSTITUTE OF INFORMATION TECHNOLOGY**

**BHOPAL (M.P.)**



**Submitted by**

SANSKAR ASATI (20U02010)

HARSH VARDHAN BADERIYA (20U02055)

**Under the supervision of**

Dr. GAGAN VISHWAKARMA

(Department of Computer Science and Engineering, IIIT Bhopal)

Dr. YATENDRA SAHU

(Department of Computer Science and Engineering, IIIT Bhopal)

**IMDB SENTIMENT ANALYSIS**

**https://drive.google.com/drive/folders/10rkLhJcDr3Iv8mTW1KP8vVxSsPLl8HUm?usp=share\_link**

1. Open the IMDB Sentimental Analysis folder.
2. The a1\_IMDB\_dataset file contains the IMDB dataset.
3. The b2\_preprocessing\_functuion.py and cleaning.py are the Data cleaning supporting files. Translator.py helps us to convert text in any language into English.
4. The a2\_glove.6B.100d file contains the GloVe word embeddings.
5. The app.py is the main flask file which helps us to integrate front end with backend.
6. All files with extension h5 are the trained models of Deep Learning used for sentiment analysis.
7. The a3\_IMDB\_Unseen\_Reviews file has some unseen review on which testing and prediction will be done.
8. The most important file is the b1\_SentimentAnalysis\_NeuralNetwork.ipynb It contains the code for actual sentiment analysis. It helps us to preprocess(cleaning) the reviews, Tokenize and embed them, run Machine Learning and Deep Learning Algorithms on it and finally predict the Sentiments with the help of training, testing and validation of these models.

**REQUIRED**

To run these files, one must have access to either Google Collab, Jupyter Lab or Kaggle. Also, Python should be installed properly and also added to the environment variable of the system. Flask should also be installed over the PC. Another requirement is Android studio with all the Flutter Plugins installed.

**HOW TO RUN**

One must run the b1\_Sentimentanalysis\_neuralNetwork.ipynb file to start training the models. On the other hand, in order to test the model, one can open the file app.py and wait for the website address to pop up with Activation status turned on. Copy the website address Then run the main.dart file in lib folder. Paste the website address in home\_page.dart file and that’s it your model is ready to run. Try to insert any movie review and click on predict to know the sentiment of the review along with its probable (predicted) rating. It has Multilanguage Support too, Do check that out!**PREREQUISITE PYTHON LIBRARIES TO BE INSTALLED:**

1. Tensorflow
2. Pandas
3. Numpy
4. RE
5. NLTK
6. Pickle
7. nltk.corpus
8. matplotlib
9. matplotlib.pyplot
10. random
11. keras
12. keras.preprocessing.text
13. keras.models
14. keras.layers.core
15. keras.layers
16. sklearn.model\_selection
17. keras\_preprocessing
18. seaborn
19. wordcloud
20. googletrans3.0.0
21. nltk.stem
22. nltk.tokenize.toktok
23. sklearn.feature\_extraction
24. sklearn.preprocessing
25. sklearn.metrics
26. tensorflow.keras.initializers
27. io
28. json
29. chardet

# b1\_SentimentAnalysis\_NeuralNetwork.ipynb

# Plan of Action

1. Load **IMDb Movie Reviews dataset (50,000 reviews)**
2. **Pre-process dataset** by removing special characters, numbers, emojis, spell check, html tags etc. from user reviews + convert **sentiment labels** positive & negative to numbers 1 & 0, respectively
3. Preparing the dataset for **Bag of Words(BOW) and TF-IDF** and using **Linear Regression Model, SVM Model and Naive Bayes Model**
4. **Import GloVe Word Embedding** to build Embedding Dictionary + Use this to build Embedding Matrix for our Corpus
5. Model Training using **Deep Learning in Keras** for separate: **Simple Neural Net, CNN and LSTM Models** and analyse model performance and results
6. Along with these Models performing the Deep Learning with **Hybrid Models** such as **BI-LSTM, CNN-LSTM, CNN-BI LSTM and HYBRID NEURAL NETWORK(HNN)**
7. Last, perform **predictions on real IMDb movie reviews**

# Setting the environment

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

# # Setting toolkit folder as working directory  
  
# %cd /content/drive/My Drive/Project8\_SentimentAnalysis\_with\_NeuralNetwork  
# ! ls

!pip install tensorflow

# Importing essential libraries and functions  
  
import pandas as pd  
import numpy as np  
import re  
import nltk  
import pickle  
from nltk.corpus import stopwords  
from numpy import array  
import tensorflow as tf  
import matplotlib.pyplot as plt  
import random

from keras.preprocessing.text import one\_hot, Tokenizer  
from keras.models import Sequential  
from keras.layers.core import Activation, Dropout, Dense  
from keras.layers import Flatten, GlobalMaxPooling1D, Embedding, Conv1D, LSTM,Bidirectional  
from sklearn.model\_selection import train\_test\_split

from keras\_preprocessing.sequence import pad\_sequences

# Loading dataset

# Importing IMDb Movie Reviews dataset  
  
movie\_reviews = pd.read\_csv("a1\_IMDB\_Dataset.csv")  
  
# dataset source: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

# Dataset exploration  
  
movie\_reviews.shape

(50000, 2)

movie\_reviews.head(5)

# Checking for missing values  
  
movie\_reviews.isnull().values.any()

# Let's observe distribution of positive / negative sentiments in dataset  
  
import seaborn as sns  
import matplotlib.pyplot as plt  
ax=sns.countplot(x='sentiment', data=movie\_reviews)  
for p in ax.patches:  
 ax.annotate(format(p.get\_height()),   
 (p.get\_x() + p.get\_width() / 2., p.get\_height()),   
 ha = 'center', va = 'center',   
 xytext = (0, -20),   
 textcoords = 'offset points')

# Choose randomly a positive review and a negative review  
ind\_positive = random.choice(list(movie\_reviews[movie\_reviews['sentiment'] == 'positive'].index))  
ind\_negative = random.choice(list(movie\_reviews[movie\_reviews['sentiment'] == 'negative'].index))  
  
review\_positive = movie\_reviews['review'][ind\_positive]  
review\_negative = movie\_reviews['review'][ind\_negative]  
  
print('Positive review: ', review\_positive)  
print('\n')  
print('Negative review: ', review\_negative)  
print('\n')  
  
from wordcloud import WordCloud  
cloud\_positive = WordCloud().generate(review\_positive)  
cloud\_negative = WordCloud().generate(review\_negative)  
  
plt.figure(figsize = (20,15))  
plt.subplot(1,2,1)  
plt.imshow(cloud\_positive)  
plt.title('Positive review')  
  
plt.subplot(1,2,2)  
plt.imshow(cloud\_negative)  
plt.title('Negative review')  
plt.show()

# Data Preprocessing

1. Translating into English
2. Removing URL
3. Removing HTML tags
4. Converting Chat Words eg.ICU, IDK, ASAP etc.
5. Converting Abbreviations
6. Removing special Characters
7. Removing Emojis
8. Removing Emoticons
9. Removing Punctuations
10. Removing extra characters and multiple spaces
11. Lowering down the characters
12. Removing Stopwords
13. Stemming
14. Lemmetization
15. Spell Check

movie\_reviews["review"][2]  
  
# You can see that our text contains punctuations, brackets, HTML tags and numbers   
# We will preprocess this text in the next section

# import b2\_preprocessing\_function  
from b2\_preprocessing\_function import CustomPreprocess  
# import translator  
from translator import translate

custom = CustomPreprocess()  
custom.preprocess\_text("Those were the best days of my life!")

convert = translate()  
convert.conversion("नमस्ते")

convert.conversion("amaaazingg rod car")

# import nltk  
# nltk.download('stopwords')

from googletrans import Translator,constants

def conversion(sen):  
   
 translator = Translator()  
 detection = translator.detect(sen)  
   
 if (constants.LANGUAGES[detection.lang]=='english'):  
 return sen  
 else:  
 translations = translator.translate(sen)  
 sen = translations.text  
 return sen

# Removing chat words  
  
chat\_words\_str = """  
AFAIK=As Far As I Know  
AFK=Away From Keyboard  
ASAP=As Soon As Possible  
ATK=At The Keyboard  
ATM=At The Moment  
A3=Anytime, Anywhere, Anyplace  
BAK=Back At Keyboard  
BBL=Be Back Later  
BBS=Be Back Soon  
BFN=Bye For Now  
B4N=Bye For Now  
BRB=Be Right Back  
BRT=Be Right There  
BTW=By The Way  
B4=Before  
B4N=Bye For Now  
CU=See You  
CUL8R=See You Later  
CYA=See You  
FAQ=Frequently Asked Questions  
FC=Fingers Crossed  
FWIW=For What It's Worth  
FYI=For Your Information  
GAL=Get A Life  
GG=Good Game  
GN=Good Night  
GMTA=Great Minds Think Alike  
GR8=Great!  
G9=Genius  
IC=I See  
ICQ=I Seek you (also a chat program)  
ILU=ILU: I Love You  
IMHO=In My Honest/Humble Opinion  
IMO=In My Opinion  
IOW=In Other Words  
IRL=In Real Life  
KISS=Keep It Simple, Stupid  
LDR=Long Distance Relationship  
LMAO=Laugh My A.. Off  
LOL=Laughing Out Loud  
LTNS=Long Time No See  
L8R=Later  
MTE=My Thoughts Exactly  
M8=Mate  
NRN=No Reply Necessary  
OIC=Oh I See  
PITA=Pain In The A..  
PRT=Party  
PRW=Parents Are Watching  
ROFL=Rolling On The Floor Laughing  
ROFLOL=Rolling On The Floor Laughing Out Loud  
ROTFLMAO=Rolling On The Floor Laughing My A.. Off  
SK8=Skate  
STATS=Your sex and age  
ASL=Age, Sex, Location  
THX=Thank You  
TTFN=Ta-Ta For Now!  
TTYL=Talk To You Later  
U=You  
U2=You Too  
U4E=Yours For Ever  
WB=Welcome Back  
WTF=What The F...  
WTG=Way To Go!  
WUF=Where Are You From?  
W8=Wait...  
7K=Sick:-D Laugher  
"""  
chat\_words\_list = []  
chat\_words\_map\_dict = {}  
for line in chat\_words\_str.split("\n"):  
 if line != "":  
 cw = line.split("=")[0]  
 cw\_expanded = line.split("=")[1]  
 chat\_words\_list.append(cw)  
 chat\_words\_map\_dict[cw] = cw\_expanded  
chat\_words\_list = set(chat\_words\_list)  
  
  
def chat\_words\_conversion(text):  
 new\_text = []  
 for w in text.split():  
 if w.upper() in chat\_words\_list:  
 new\_text.append(chat\_words\_map\_dict[w.upper()])  
 else:  
 new\_text.append(w)  
 return " ".join(new\_text)

## In this case, we will be replacing some abbreviated pronouns with full forms (example:"you've"->you have")  
abbreviations = {  
 u"he's": "he is",   
 u"there's": "there is",   
 u"We're": "We are",   
 u"That's": "That is",   
 u"won't": "will not",   
 u"they're": "they are",   
 u"Can't": "Cannot",   
 u"wasn't": "was not",   
 u"don\x89Ûªt": "do not",   
 u"aren't": "are not",   
 u"isn't": "is not",   
 u"What's": "What is",   
 u"haven't": "have not",   
 u"hasn't": "has not",   
 u"There's": "There is",   
 u"He's": "He is",   
 u"It's": "It is",   
 u"You're": "You are",   
 u"I'M": "I am",   
 u"shouldn't": "should not",   
 u"wouldn't": "would not",   
 u"i'm": "I am",   
 u"I\x89Ûªm": "I am",   
 u"I'm": "I am",   
 u"Isn't": "is not",   
 u"Here's": "Here is",   
 u"you've": "you have",   
 u"you\x89Ûªve": "you have",   
 u"we're": "we are",   
 u"what's": "what is",   
 u"couldn't": "could not",   
 u"we've": "we have",   
 u"it\x89Ûªs": "it is",   
 u"doesn\x89Ûªt": "does not",   
 u"It\x89Ûªs": "It is",   
 u"Here\x89Ûªs": "Here is",   
 u"who's": "who is",   
 u"I\x89Ûªve": "I have",   
 u"y'all": "you all",   
 u"can\x89Ûªt": "cannot",   
 u"would've": "would have",   
 u"it'll": "it will",   
 u"we'll": "we will",   
 u"wouldn\x89Ûªt": "would not",   
 u"We've": "We have",   
 u"he'll": "he will",   
 u"Y'all": "You all",   
 u"Weren't": "Were not",   
 u"Didn't": "Did not",   
 u"they'll": "they will",   
 u"they'd": "they would",   
 u"DON'T": "DO NOT",   
 u"That\x89Ûªs": "That is",   
 u"they've": "they have",   
 u"i'd": "I would",   
 u"should've": "should have",   
 u"You\x89Ûªre": "You are",   
 u"where's": "where is",   
 u"Don\x89Ûªt": "Do not",   
 u"we'd": "we would",   
 u"i'll": "I will",   
 u"weren't": "were not",   
 u"They're": "They are",   
 u"Can\x89Ûªt": "Cannot",   
 u"you\x89Ûªll": "you will",   
 u"I\x89Ûªd": "I would",   
 u"let's": "let us",   
 u"it's": "it is",   
 u"can't": "cannot",   
 u"don't": "do not",   
 u"you're": "you are",   
 u"i've": "I have",   
 u"that's": "that is",   
 u"i'll": "I will",   
 u"doesn't": "does not",  
 u"i'd": "I would",   
 u"didn't": "did not",   
 u"ain't": "am not",   
 u"you'll": "you will",   
 u"I've": "I have",   
 u"Don't": "do not",   
 u"I'll": "I will",   
 u"I'd": "I would",   
 u"Let's": "Let us",   
 u"you'd": "You would",   
 u"It's": "It is",   
 u"Ain't": "am not",   
 u"Haven't": "Have not",   
 u"Could've": "Could have",   
 u"youve": "you have",   
 u"donå«t": "do not",   
}  
   
def convert\_abb(text):  
 for emot in abbreviations:  
 text = re.sub(u'('+emot+')', "\_".join(abbreviations[emot].replace(",","").split()), text)  
 return text

# Spell Check  
from textblob import TextBlob  
def spell\_check(text):  
 textBlb = TextBlob(text)  
 return textBlb.correct().string

# Removing emotions  
  
EMOTICONS = {  
 u":‑\)":"Happy face or smiley",  
 u":\)":"Happy face or smiley",  
 u":-\]":"Happy face or smiley",  
 u":\]":"Happy face or smiley",  
 u":-3":"Happy face smiley",  
 u":3":"Happy face smiley",  
 u":->":"Happy face smiley",  
 u":>":"Happy face smiley",  
 u"8-\)":"Happy face smiley",  
 u":o\)":"Happy face smiley",  
 u":-\}":"Happy face smiley",  
 u":\}":"Happy face smiley",  
 u":-\)":"Happy face smiley",  
 u":c\)":"Happy face smiley",  
 u":\^\)":"Happy face smiley",  
 u"=\]":"Happy face smiley",  
 u"=\)":"Happy face smiley",  
 u":‑D":"Laughing, big grin or laugh with glasses",  
 u":D":"Laughing, big grin or laugh with glasses",  
 u"8‑D":"Laughing, big grin or laugh with glasses",  
 u"8D":"Laughing, big grin or laugh with glasses",  
 u"X‑D":"Laughing, big grin or laugh with glasses",  
 u"XD":"Laughing, big grin or laugh with glasses",  
 u"=D":"Laughing, big grin or laugh with glasses",  
 u"=3":"Laughing, big grin or laugh with glasses",  
 u"B\^D":"Laughing, big grin or laugh with glasses",  
 u":-\)\)":"Very happy",  
 u":‑\(":"Frown, sad, andry or pouting",  
 u":-\(":"Frown, sad, andry or pouting",  
 u":\(":"Frown, sad, andry or pouting",  
 u":‑c":"Frown, sad, andry or pouting",  
 u":c":"Frown, sad, andry or pouting",  
 u":‑<":"Frown, sad, andry or pouting",  
 u":<":"Frown, sad, andry or pouting",  
 u":‑\[":"Frown, sad, andry or pouting",  
 u":\[":"Frown, sad, andry or pouting",  
 u":-\|\|":"Frown, sad, andry or pouting",  
 u">:\[":"Frown, sad, andry or pouting",  
 u":\{":"Frown, sad, andry or pouting",  
 u":@":"Frown, sad, andry or pouting",  
 u">:\(":"Frown, sad, andry or pouting",  
 u":'‑\(":"Crying",  
 u":'\(":"Crying",  
 u":'‑\)":"Tears of happiness",  
 u":'\)":"Tears of happiness",  
 u"D‑':":"Horror",  
 u"D:<":"Disgust",  
 u"D:":"Sadness",  
 u"D8":"Great dismay",  
 u"D;":"Great dismay",  
 u"D=":"Great dismay",  
 u"DX":"Great dismay",  
 u":‑O":"Surprise",  
 u":O":"Surprise",  
 u":‑o":"Surprise",  
 u":o":"Surprise",  
 u":-0":"Shock",  
 u"8‑0":"Yawn",  
 u">:O":"Yawn",  
 u":-\\*":"Kiss",  
 u":\\*":"Kiss",  
 u":X":"Kiss",  
 u";‑\)":"Wink or smirk",  
 u";\)":"Wink or smirk",  
 u"\\*-\)":"Wink or smirk",  
 u"\\*\)":"Wink or smirk",  
 u";‑\]":"Wink or smirk",  
 u";\]":"Wink or smirk",  
 u";\^\)":"Wink or smirk",  
 u":‑,":"Wink or smirk",  
 u";D":"Wink or smirk",  
 u":‑P":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u":P":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u"X‑P":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u"XP":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u":‑Þ":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u":Þ":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u":b":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u"d:":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u"=p":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u">:P":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u":‑/":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":/":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":-[.]":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u">:[(\\\)]":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u">:/":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":[(\\\)]":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u"=/":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u"=[(\\\)]":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":L":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u"=L":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":S":"Skeptical, annoyed, undecided, uneasy or hesitant",  
 u":‑\|":"Straight face",  
 u":\|":"Straight face",  
 u":$":"Embarrassed or blushing",  
 u":‑x":"Sealed lips or wearing braces or tongue-tied",  
 u":x":"Sealed lips or wearing braces or tongue-tied",  
 u":‑#":"Sealed lips or wearing braces or tongue-tied",  
 u":#":"Sealed lips or wearing braces or tongue-tied",  
 u":‑&":"Sealed lips or wearing braces or tongue-tied",  
 u":&":"Sealed lips or wearing braces or tongue-tied",  
 u"O:‑\)":"Angel, saint or innocent",  
 u"O:\)":"Angel, saint or innocent",  
 u"0:‑3":"Angel, saint or innocent",  
 u"0:3":"Angel, saint or innocent",  
 u"0:‑\)":"Angel, saint or innocent",  
 u"0:\)":"Angel, saint or innocent",  
 u":‑b":"Tongue sticking out, cheeky, playful or blowing a raspberry",  
 u"0;\^\)":"Angel, saint or innocent",  
 u">:‑\)":"Evil or devilish",  
 u">:\)":"Evil or devilish",  
 u"\}:‑\)":"Evil or devilish",  
 u"\}:\)":"Evil or devilish",  
 u"3:‑\)":"Evil or devilish",  
 u"3:\)":"Evil or devilish",  
 u">;\)":"Evil or devilish",  
 u"\|;‑\)":"Cool",  
 u"\|‑O":"Bored",  
 u":‑J":"Tongue-in-cheek",  
 u"#‑\)":"Party all night",  
 u"%‑\)":"Drunk or confused",  
 u"%\)":"Drunk or confused",  
 u":-###..":"Being sick",  
 u":###..":"Being sick",  
 u"<:‑\|":"Dump",  
 u"\(>\_<\)":"Troubled",  
 u"\(>\_<\)>":"Troubled",  
 u"\(';'\)":"Baby",  
 u"\(\^\^>``":"Nervous or Embarrassed or Troubled or Shy or Sweat drop",  
 u"\(\^\_\^;\)":"Nervous or Embarrassed or Troubled or Shy or Sweat drop",  
 u"\(-\_-;\)":"Nervous or Embarrassed or Troubled or Shy or Sweat drop",  
 u"\(~\_~;\) \(・\.・;\)":"Nervous or Embarrassed or Troubled or Shy or Sweat drop",  
 u"\(-\_-\)zzz":"Sleeping",  
 u"\(\^\_-\)":"Wink",  
 u"\(\(\+\_\+\)\)":"Confused",  
 u"\(\+o\+\)":"Confused",  
 u"\(o\|o\)":"Ultraman",  
 u"\^\_\^":"Joyful",  
 u"\(\^\_\^\)/":"Joyful",  
 u"\(\^O\^\)／":"Joyful",  
 u"\(\^o\^\)／":"Joyful",  
 u"\(\_\_\)":"Kowtow as a sign of respect, or dogeza for apology",  
 u"\_\(\.\_\.\)\_":"Kowtow as a sign of respect, or dogeza for apology",  
 u"<\(\_ \_\)>":"Kowtow as a sign of respect, or dogeza for apology",  
 u"<m\(\_\_\)m>":"Kowtow as a sign of respect, or dogeza for apology",  
 u"m\(\_\_\)m":"Kowtow as a sign of respect, or dogeza for apology",  
 u"m\(\_ \_\)m":"Kowtow as a sign of respect, or dogeza for apology",  
 u"\('\_'\)":"Sad or Crying",  
 u"\(/\_;\)":"Sad or Crying",  
 u"\(T\_T\) \(;\_;\)":"Sad or Crying",  
 u"\(;\_;":"Sad of Crying",  
 u"\(;\_:\)":"Sad or Crying",  
 u"\(;O;\)":"Sad or Crying",  
 u"\(:\_;\)":"Sad or Crying",  
 u"\(ToT\)":"Sad or Crying",  
 u";\_;":"Sad or Crying",  
 u";-;":"Sad or Crying",  
 u";n;":"Sad or Crying",  
 u";;":"Sad or Crying",  
 u"Q\.Q":"Sad or Crying",  
 u"T\.T":"Sad or Crying",  
 u"QQ":"Sad or Crying",  
 u"Q\_Q":"Sad or Crying",  
 u"\(-\.-\)":"Shame",  
 u"\(-\_-\)":"Shame",  
 u"\(一一\)":"Shame",  
 u"\(；一\_一\)":"Shame",  
 u"\(=\_=\)":"Tired",  
 u"\(=\^\·\^=\)":"cat",  
 u"\(=\^\·\·\^=\)":"cat",  
 u"=\_\^= ":"cat",  
 u"\(\.\.\)":"Looking down",  
 u"\(\.\_\.\)":"Looking down",  
 u"\^m\^":"Giggling with hand covering mouth",  
 u"\(\・\・?":"Confusion",  
 u"\(?\_?\)":"Confusion",  
 u">\^\_\^<":"Normal Laugh",  
 u"<\^!\^>":"Normal Laugh",  
 u"\^/\^":"Normal Laugh",  
 u"\（\\*\^\_\^\\*）" :"Normal Laugh",  
 u"\(\^<\^\) \(\^\.\^\)":"Normal Laugh",  
 u"\(^\^\)":"Normal Laugh",  
 u"\(\^\.\^\)":"Normal Laugh",  
 u"\(\^\_\^\.\)":"Normal Laugh",  
 u"\(\^\_\^\)":"Normal Laugh",  
 u"\(\^\^\)":"Normal Laugh",  
 u"\(\^J\^\)":"Normal Laugh",  
 u"\(\\*\^\.\^\\*\)":"Normal Laugh",  
 u"\(\^—\^\）":"Normal Laugh",  
 u"\(#\^\.\^#\)":"Normal Laugh",  
 u"\（\^—\^\）":"Waving",  
 u"\(;\_;\)/~~~":"Waving",  
 u"\(\^\.\^\)/~~~":"Waving",  
 u"\(-\_-\)/~~~ \($\·\·\)/~~~":"Waving",  
 u"\(T\_T\)/~~~":"Waving",  
 u"\(ToT\)/~~~":"Waving",  
 u"\(\\*\^0\^\\*\)":"Excited",  
 u"\(\\*\_\\*\)":"Amazed",  
 u"\(\\*\_\\*;":"Amazed",  
 u"\(\+\_\+\) \(@\_@\)":"Amazed",  
 u"\(\\*\^\^\)v":"Laughing,Cheerful",  
 u"\(\^\_\^\)v":"Laughing,Cheerful",  
 u"\(\(d[-\_-]b\)\)":"Headphones,Listening to music",  
 u'\(-"-\)':"Worried",  
 u"\(ーー;\)":"Worried",  
 u"\(\^0\_0\^\)":"Eyeglasses",  
 u"\(\＾ｖ\＾\)":"Happy",  
 u"\(\＾ｕ\＾\)":"Happy",  
 u"\(\^\)o\(\^\)":"Happy",  
 u"\(\^O\^\)":"Happy",  
 u"\(\^o\^\)":"Happy",  
 u"\)\^o\^\(":"Happy",  
 u":O o\_O":"Surprised",  
 u"o\_0":"Surprised",  
 u"o\.O":"Surpised",  
 u"\(o\.o\)":"Surprised",  
 u"oO":"Surprised",  
 u"\(\\*￣m￣\)":"Dissatisfied",  
 u"\(‘A`\)":"Snubbed or Deflated"  
}  
  
def remove\_emoticons(text):  
 emoticon\_pattern = re.compile(u'(' + u'|'.join(k for k in EMOTICONS) + u')')  
 return emoticon\_pattern.sub(r'', text)

# Stemming  
  
from nltk.stem.porter import PorterStemmer  
ps = PorterStemmer()  
def stem\_words(text):  
 return " ".join([ps.stem(word) for word in text.split()])

# URL Removal  
  
def remove\_url(text):  
 url\_tag = re.compile(r'https://\S+|www\.\S+')  
 text = url\_tag.sub(r'', text)  
 return text

# HTML Tags Removal  
  
def remove\_html(text):  
 html\_tag = re.compile(r'<.\*?>')  
 text = html\_tag.sub(r'', text)  
 return text

# Punctuation Removal  
  
def remove\_punctuation(text):   
 punct\_tag = re.compile(r'[^\w\s]')  
 text = punct\_tag.sub(r'', text)   
 return text

# Special Character Removal  
  
def remove\_special\_character(text):  
 special\_tag = re.compile(r'[^a-zA-Z0-9\s]')  
 text = special\_tag.sub(r'', text)  
 return text

# Emojis Removal  
  
def remove\_emoji(text):  
 emoji\_pattern = re.compile("["  
 u"\U0001F600-\U0001F64F" # emoticons  
 u"\U0001F300-\U0001F5FF" # symbols & pictographs  
 u"\U0001F680-\U0001F6FF" # transport & map symbols  
 u"\U0001F1E0-\U0001F1FF" # flags (iOS)  
 u"\U00002702-\U000027B0"  
 u"\U000024C2-\U0001F251"  
 "]+", flags=re.UNICODE)  
 return emoji\_pattern.sub(r'', text)

# Preprocessing Function  
  
from nltk.stem import WordNetLemmatizer  
def clean\_text(text):  
   
 wordnet\_lemmatizer = WordNetLemmatizer()  
 sequencePattern = r"(.)\1\1+"  
 seqReplacePattern = r"\1\1"  
   
 text = custom.preprocess\_text(text)  
 text = remove\_url(text)  
 text = remove\_html(text)  
 text = chat\_words\_conversion(text)  
 text = convert\_abb(text)  
 text = remove\_special\_character(text)  
 text = remove\_emoji(text)  
 text = remove\_emoticons(text)  
 text = remove\_punctuation(text)  
 text = re.sub('[^a-zA-Z]', ' ', text)  
 text = re.sub(sequencePattern, seqReplacePattern, text)  
 text = re.sub(r"\s+[a-zA-Z]\s+", ' ', text) # When we remove apostrophe from the word "Mark's", the apostrophe is replaced by an empty space. Hence, we are left with single character "s" that we are removing here.  
 text = re.sub(r'\s+', ' ', text) # Next, we remove all the single characters and replace it by a space which creates multiple spaces in our text. Finally, we remove the multiple spaces from our text as well.  
 text = text.lower()  
   
 # Remove Stopwords  
 pattern = re.compile(r'\b(' + r'|'.join(stopwords.words('english')) + r')\b\s\*')  
 text = pattern.sub('', text)  
   
 text = stem\_words(text)  
   
 #Lemmetization  
 sentence\_words = nltk.word\_tokenize(text)  
 textwords = ''  
 for word in sentence\_words:  
 name = wordnet\_lemmatizer.lemmatize(word,pos='v')  
 textwords += (name+' ')  
 text = textwords  
   
 text = spell\_check(text)   
   
 return text

movie\_reviews['processed'] = movie\_reviews['review'].apply(lambda x: custom.preprocess\_text(x))

movie\_reviews.head(5)

#Choose randomly a positive review and a negative review  
  
ind\_positive = random.choice(list(movie\_reviews[movie\_reviews['sentiment'] == 'positive'].index))  
ind\_negative = random.choice(list(movie\_reviews[movie\_reviews['sentiment'] == 'negative'].index))  
  
review\_positive = movie\_reviews['processed'][ind\_positive]  
review\_negative = movie\_reviews['processed'][ind\_negative]  
  
print('Positive review: ', review\_positive)  
print('\n')  
print('Negative review: ', review\_negative)  
print('\n')  
  
from wordcloud import WordCloud  
cloud\_positive = WordCloud().generate(review\_positive)  
cloud\_negative = WordCloud().generate(review\_negative)  
  
plt.figure(figsize = (20,15))  
plt.subplot(1,2,1)  
plt.imshow(cloud\_positive)  
plt.title('Positive review')  
  
plt.subplot(1,2,2)  
plt.imshow(cloud\_negative)  
plt.title('Negative review')  
plt.show()

# Converting sentiment labels to 0 & 1  
  
y = movie\_reviews['sentiment']  
  
y = np.array(list(map(lambda x: 1 if x=="positive" else 0, y)))

# Calling preprocessing\_text function on movie\_reviews  
  
X = movie\_reviews['processed']

# Sample cleaned up movie review   
  
X[2]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)  
  
# The train set will be used to train our deep learning models   
# while test set will be used to evaluate how well our model performs

## Data preprocessing for BOW and TF-IDF

from nltk.tokenize.toktok import ToktokTokenizer  
#Tokenization of text  
tokenizer=ToktokTokenizer()

#tokenizing  
def data\_preprocess(text):  
 tokens = tokenizer.tokenize(text)  
 tokens = [token.strip() for token in tokens]  
 filtered\_text = ' '.join(tokens)   
 return filtered\_text  
#Apply function on review column  
movie\_reviews['processed']=movie\_reviews['processed'].apply(data\_preprocess)

#normalized train reviews  
norm\_train\_reviews = movie\_reviews.processed[:40000]  
norm\_train\_reviews[0]

#Normalized test reviews  
norm\_test\_reviews=movie\_reviews.processed[40000:]  
norm\_test\_reviews[45005]

# Bags of words Model

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

from sklearn.feature\_extraction.text import CountVectorizer  
#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

# Term Frequency-Inverse Document Frequency Model (TFIDF)

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

from sklearn.feature\_extraction.text import TfidfVectorizer  
#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)

## Labeling the sentiment text

from sklearn.preprocessing import LabelBinarizer  
#labeling the sentient data  
lb=LabelBinarizer()  
#transformed sentiment data  
sentiment\_data=lb.fit\_transform(movie\_reviews['sentiment'])  
print(sentiment\_data.shape)

#Spliting the sentiment data  
train\_sentiments=sentiment\_data[:40000]  
test\_sentiments=sentiment\_data[40000:]  
print(train\_sentiments)  
print(test\_sentiments)

ml\_data=[]

# Modelling the dataset using Machine Learning

## Linear Logistic Model

Let us build **Logistic Regression Model** for both **Bag of Words** and **TF-IDF** features

from sklearn.linear\_model import LogisticRegression,SGDClassifier  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.svm import SVC  
from sklearn.metrics import classification\_report,confusion\_matrix,accuracy\_score  
  
import warnings  
warnings.filterwarnings('ignore')

#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)

#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)

#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)

import pandas as pd  
import matplotlib.pyplot as plt  
model\_names=["LR\_BOW","LR\_TFIDF"]  
model\_acc = [round(lr\_bow\_score\*100,2),round(lr\_tfidf\_score\*100,2)]  
  
# Create a bar chart  
plt.close('all')  
colors = ['green', 'blue']  
plt.rcParams["figure.figsize"] = (6,5)  
plt.bar(model\_names, model\_acc,color=colors,width=0.05)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(model\_acc):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

#Classification report for bag of words   
lr\_bow\_report=classification\_report(test\_sentiments,lr\_bow\_predict,target\_names=['Positive','Negative'])  
print(lr\_bow\_report)  
  
#Classification report for tfidf features  
lr\_tfidf\_report=classification\_report(test\_sentiments,lr\_tfidf\_predict,target\_names=['Positive','Negative'])  
print(lr\_tfidf\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(lr\_bow.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = lr\_bow.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(lr\_bow\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["LR\_BOW", f1, precision, recall, accuracy])

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(lr\_tfidf.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = lr\_tfidf.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(lr\_tfidf\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["LR\_TFIDF", f1, precision, recall, accuracy])

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_bow = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,lr\_bow\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_bow.plot()  
plt.title('Confusion matrix BOW')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_tfidf = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,lr\_tfidf\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_tfidf.plot()  
plt.title('Confusion matrix TFIDF')  
plt.show()

## Stochastic Gradient Descent or Linear Support Vector Machines

**SVM** for **Bag of Words** and **TF-IDF** 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)

#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)

#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)

import pandas as pd  
import matplotlib.pyplot as plt  
model\_names=["SVM\_BOW","SVM\_TFIDF"]  
model\_acc = [round(svm\_bow\_score\*100,2),round(svm\_tfidf\_score\*100,2)]  
  
# Create a bar chart  
plt.close('all')  
colors = ['green', 'blue']  
plt.rcParams["figure.figsize"] = (6,5)  
plt.bar(model\_names, model\_acc,color=colors,width=0.05)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(model\_acc):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

#Classification report for bag of words   
svm\_bow\_report=classification\_report(test\_sentiments,svm\_bow\_predict,target\_names=['Positive','Negative'])  
print(svm\_bow\_report)  
#Classification report for tfidf features  
svm\_tfidf\_report=classification\_report(test\_sentiments,svm\_tfidf\_predict,target\_names=['Positive','Negative'])  
print(svm\_tfidf\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(svm\_bow.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = svm\_bow.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(svm\_bow\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["SVM\_BOW", f1, precision, recall, accuracy])

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(svm\_tfidf.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = svm\_tfidf.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(svm\_tfidf\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["SVM\_TFIDF", f1, precision, recall, accuracy])

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_bow = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,svm\_bow\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_bow.plot()  
plt.title('Confusion matrix BOW')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_tfidf = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,svm\_tfidf\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_tfidf.plot()  
plt.title('Confusion matrix TFIDF')  
plt.show()

## Multinomial Naive Bayes Model

**MultinominalNB** for **Bag of Words** and **TF-IDF** features

#training the model  
mnb=MultinomialNB()  
#fitting the svm for bag of words  
mnb\_bow=mnb.fit(cv\_train\_reviews,train\_sentiments)  
print(mnb\_bow)  
#fitting the svm for tfidf features  
mnb\_tfidf=mnb.fit(tv\_train\_reviews,train\_sentiments)  
print(mnb\_tfidf)

#Predicting the model for bag of words  
mnb\_bow\_predict=mnb.predict(cv\_test\_reviews)  
print(mnb\_bow\_predict)  
#Predicting the model for tfidf features  
mnb\_tfidf\_predict=mnb.predict(tv\_test\_reviews)  
print(mnb\_tfidf\_predict)

#Accuracy score for bag of words  
mnb\_bow\_score=accuracy\_score(test\_sentiments,mnb\_bow\_predict)  
print("mnb\_bow\_score :",mnb\_bow\_score)  
#Accuracy score for tfidf features  
mnb\_tfidf\_score=accuracy\_score(test\_sentiments,mnb\_tfidf\_predict)  
print("mnb\_tfidf\_score :",mnb\_tfidf\_score)

import pandas as pd  
import matplotlib.pyplot as plt  
model\_names=["MNB\_BOW","MNB\_TFIDF"]  
model\_acc = [round(mnb\_bow\_score\*100,2),round(mnb\_tfidf\_score\*100,2)]  
  
# Create a bar chart  
plt.close('all')  
colors = ['green', 'blue']  
plt.rcParams["figure.figsize"] = (6,5)  
plt.bar(model\_names, model\_acc,color=colors,width=0.05)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(model\_acc):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

#Classification report for bag of words   
mnb\_bow\_report=classification\_report(test\_sentiments,mnb\_bow\_predict,target\_names=['Positive','Negative'])  
print(mnb\_bow\_report)  
#Classification report for tfidf features  
mnb\_tfidf\_report=classification\_report(test\_sentiments,mnb\_tfidf\_predict,target\_names=['Positive','Negative'])  
print(mnb\_tfidf\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(mnb\_bow.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = mnb\_bow.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(mnb\_bow\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["MNB\_BOW", f1, precision, recall, accuracy])

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(mnb\_tfidf.predict(tv\_test\_reviews), axis=-1)  
y\_true = tf.argmax(test\_sentiments, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = mnb\_tfidf.predict(tv\_test\_reviews)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(mnb\_tfidf\_score\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

ml\_data.append(["MNB\_TFIDF", f1, precision, recall, accuracy])

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_bow = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,mnb\_bow\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_bow.plot()  
plt.title('Confusion matrix BOW')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
cm\_tfidf = ConfusionMatrixDisplay(confusion\_matrix(test\_sentiments,mnb\_tfidf\_predict), display\_labels = ['Positive','Negative'])  
plt.figure(figsize = (5,5))  
cm\_tfidf.plot()  
plt.title('Confusion matrix TFIDF')  
plt.show()

# COMPARISION

import numpy as np  
import matplotlib.pyplot as plt  
  
# extract the accuracy, precision, recall, and f1 values for each model  
accuracy = [x[1] for x in ml\_data]  
precision = [x[2] for x in ml\_data]  
recall = [x[3] for x in ml\_data]  
f1 = [(x[4]/100) for x in ml\_data]  
  
# set the width of the bars  
barWidth = 0.20  
  
# set the positions of the bars on the x-axis  
r1 = np.arange(len(ml\_data))  
r2 = [x + barWidth for x in r1]  
r3 = [x + barWidth for x in r2]  
r4 = [x + barWidth for x in r3]  
  
# create the bar chart  
plt.subplots(figsize=(14,10))  
bar1=plt.bar(r1, accuracy, color='b', width=barWidth, edgecolor='white', label='Accuracy')  
bar2=plt.bar(r2, precision, color='g', width=barWidth, edgecolor='white', label='Precision')  
bar3=plt.bar(r3, recall, color='r', width=barWidth, edgecolor='white', label='Recall')  
bar4=plt.bar(r4, f1, color='c', width=barWidth, edgecolor='white', label='F1 Score')  
  
# add labels and titles  
plt.xlabel('Model')  
plt.xticks([r + barWidth for r in range(len(ml\_data))],[x[0] for x in ml\_data])  
plt.ylabel('Score')  
plt.title('Performance Metrics')  
  
# add legend  
plt.legend(bbox\_to\_anchor=(1.05, 1.0), loc='upper left')  
  
# add text labels above each bar  
for i in range(len(ml\_data)):  
 plt.text(r1[i],accuracy[i], '{:.2f}'.format(accuracy[i]),ha='center')  
 plt.text(r2[i],precision[i], '{:.2f}'.format(precision[i]),ha='center')  
 plt.text(r3[i],recall[i], '{:.2f}'.format(recall[i]),ha='center')  
 plt.text(r4[i],f1[i], '{:.2f}'.format(f1[i]),ha='center')  
  
# show the plot  
plt.show()

## BAG OF WORDS

import pandas as pd  
models=["LR","SVM","MNB"]  
accuracy = []  
accuracy.append(lr\_bow\_score)  
accuracy.append(svm\_bow\_score)  
accuracy.append(mnb\_bow\_score)  
  
# Sample data  
accuracy = np.array(accuracy)  
  
# Convert to percent with one decimal place  
accuracy = np.round(accuracy \* 100, 1)  
  
metrics = pd.DataFrame({  
 'Model': models,  
 'Accuracy': accuracy,  
})  
  
# Set Model column as index  
metrics = metrics.set\_index('Model')  
  
# Display DataFrame  
print(metrics)  
  
# Print the results

import matplotlib.pyplot as plt  
  
  
# Create a bar chart  
plt.close('all')  
colors = ['red', 'green', 'blue']  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.bar(models, accuracy,color=colors,width=0.5)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(accuracy):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

## TF-IDF

import pandas as pd  
models=["LR","SVM","MNB"]  
accuracy = []  
accuracy.append(lr\_tfidf\_score)  
accuracy.append(svm\_tfidf\_score)  
accuracy.append(mnb\_tfidf\_score)  
  
# Sample data  
accuracy = np.array(accuracy)  
  
# Convert to percent with one decimal place  
accuracy = np.round(accuracy \* 100, 1)  
  
metrics = pd.DataFrame({  
 'Model': models,  
 'Accuracy': accuracy,  
})  
  
# Set Model column as index  
metrics = metrics.set\_index('Model')  
  
# Display DataFrame  
print(metrics)  
  
# Print the results

import matplotlib.pyplot as plt  
  
  
# Create a bar chart  
plt.close('all')  
colors = ['red', 'green', 'blue']  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.bar(models, accuracy,color=colors,width=0.5)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(accuracy):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

# Preparing Embedding layer

Let's now write the script for our embedding layer. Embedding layer converts our textual data into numeric form. It is then **used as the first layer for the deep learning models in Keras**.

# Embedding layer expects the words to be in numeric form   
# Using Tokenizer function from keras.preprocessing.text library  
# Method fit\_on\_text trains the tokenizer   
# Method texts\_to\_sequences converts sentences to their numeric form  
  
word\_tokenizer = Tokenizer()  
word\_tokenizer.fit\_on\_texts(X\_train)  
  
X\_train = word\_tokenizer.texts\_to\_sequences(X\_train)  
X\_test = word\_tokenizer.texts\_to\_sequences(X\_test)

import io  
import json

# Saving  
  
tokenizer\_json = word\_tokenizer.to\_json()  
with io.open('b3\_tokenizer.json', 'w', encoding='utf-8') as f:  
 f.write(json.dumps(tokenizer\_json, ensure\_ascii=False))

# Adding 1 to store dimensions for words for which no pretrained word embeddings exist  
  
vocab\_length = len(word\_tokenizer.word\_index) + 1  
  
vocab\_length

# Padding all reviews to fixed length 100  
  
maxlen = 100  
  
X\_train = pad\_sequences(X\_train, padding='post', maxlen=maxlen)  
X\_test = pad\_sequences(X\_test, padding='post', maxlen=maxlen)

# Load GloVe word embeddings and create an Embeddings Dictionary  
  
from numpy import asarray  
from numpy import zeros  
  
embeddings\_dictionary = dict()  
glove\_file = open('a2\_glove.6B.100d.txt', encoding="utf8")  
  
for line in glove\_file:  
 records = line.split()  
 word = records[0]  
 vector\_dimensions = asarray(records[1:], dtype='float32')  
 embeddings\_dictionary [word] = vector\_dimensions  
glove\_file.close()

# Create Embedding Matrix having 100 columns   
# Containing 100-dimensional GloVe word embeddings for all words in our corpus.  
  
embedding\_matrix = zeros((vocab\_length, 100))  
for word, index in word\_tokenizer.word\_index.items():  
 embedding\_vector = embeddings\_dictionary.get(word)  
 if embedding\_vector is not None:  
 embedding\_matrix[index] = embedding\_vector

embedding\_matrix.shape

data=[]

# Model Training with:

## Simple Neural Network (SNN) MODEL

# Neural Network architecture  
  
snn\_model = Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
snn\_model.add(embedding\_layer)  
  
snn\_model.add(Flatten())  
snn\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
snn\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
  
print(snn\_model.summary())

# Model training  
  
snn\_model\_history = snn\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
snn\_score = snn\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", snn\_score[0])  
print("Test Accuracy:", snn\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(snn\_model\_history.history['acc'])  
plt.plot(snn\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(snn\_model\_history.history['loss'])  
plt.plot(snn\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = snn\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
snn\_report=classification\_report(y\_test, y\_pred)  
print(snn\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(snn\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = snn\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(snn\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["Simpe ANN", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## Convolutional Neural Network (CNN) MODEL

from keras.layers import Conv1D,MaxPooling1D,BatchNormalization,Dropout,GlobalMaxPooling1D,Dense  
from tensorflow.keras.initializers import RandomNormal, Constant

# Neural Network architecture  
  
cnn\_model = Sequential()  
  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
cnn\_model.add(embedding\_layer)  
  
cnn\_model.add(Conv1D(108, 5, activation='relu'))  
cnn\_model.add(MaxPooling1D(pool\_size=3,strides=1, padding='same'))  
cnn\_model.add(BatchNormalization(momentum=0.95, epsilon=0.005,beta\_initializer=RandomNormal(mean=0.0, stddev=0.05), gamma\_initializer=Constant(value=0.9)))  
cnn\_model.add(GlobalMaxPooling1D())  
cnn\_model.add(Dropout(0.15))  
cnn\_model.add(Dense(1, activation='sigmoid'))

# load saved model

# Model compiling  
  
cnn\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(cnn\_model.summary())

# Model training  
  
cnn\_model\_history = cnn\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
cnn\_score = cnn\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", cnn\_score[0])  
print("Test Accuracy:", cnn\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(cnn\_model\_history.history['acc'])  
plt.plot(cnn\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc = 'upper left')  
plt.show()  
  
plt.plot(cnn\_model\_history.history['loss'])  
plt.plot(cnn\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc = 'upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = cnn\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
cnn\_report=classification\_report(y\_test, y\_pred)  
print(cnn\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(cnn\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = cnn\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(cnn\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["CNN", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## OPTIMIZED CNN MODEL

# Neural Network architecture  
  
cnnop\_model = Sequential()  
  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
cnnop\_model.add(embedding\_layer)  
  
cnnop\_model.add(Conv1D(88, 5, activation='relu'))  
cnnop\_model.add(GlobalMaxPooling1D())  
cnnop\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
cnnop\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(cnnop\_model.summary())

# Model training  
  
cnnop\_model\_history = cnnop\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
cnnop\_score = cnnop\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", cnnop\_score[0])  
print("Test Accuracy:", cnnop\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(cnnop\_model\_history.history['acc'])  
plt.plot(cnnop\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc = 'upper left')  
plt.show()  
  
plt.plot(cnnop\_model\_history.history['loss'])  
plt.plot(cnnop\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc = 'upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = cnn\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
cnnop\_report=classification\_report(y\_test, y\_pred)  
print(cnnop\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(cnnop\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = cnnop\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(cnnop\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["CNNOP", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## Recurrent Neural Network (LSTM) MODEL

from keras.layers import LSTM

# Neural Network architecture  
  
lstm\_model = Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
lstm\_model.add(embedding\_layer)  
lstm\_model.add(LSTM(116))  
  
lstm\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
lstm\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(lstm\_model.summary())

# Model Training  
  
lstm\_model\_history = lstm\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
lstm\_score = lstm\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", lstm\_score[0])  
print("Test Accuracy:", lstm\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(lstm\_model\_history.history['acc'])  
plt.plot(lstm\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(lstm\_model\_history.history['loss'])  
plt.plot(lstm\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = lstm\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
lstm\_report=classification\_report(y\_test, y\_pred)  
print(lstm\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(lstm\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = lstm\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(lstm\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["LSTM", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## BI-LSTM MODEL

from keras.layers import LSTM,Bidirectional,Conv1D

# Neural Network architecture  
  
bilstm\_model = Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
bilstm\_model.add(embedding\_layer)  
bilstm\_model.add(Bidirectional(LSTM(76)))  
  
bilstm\_model.add(Dense(116, activation='sigmoid'))  
bilstm\_model.add(Dropout(0.15))  
bilstm\_model.add(Dense(48, activation='sigmoid'))  
bilstm\_model.add(Dropout(0.20))  
  
bilstm\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
bilstm\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(bilstm\_model.summary())

# Model Training  
  
bilstm\_model\_history = bilstm\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
bilstm\_score = bilstm\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", bilstm\_score[0])  
print("Test Accuracy:", bilstm\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(bilstm\_model\_history.history['acc'])  
plt.plot(bilstm\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(bilstm\_model\_history.history['loss'])  
plt.plot(bilstm\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = bilstm\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
bilstm\_report=classification\_report(y\_test, y\_pred)  
print(bilstm\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(bilstm\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = bilstm\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(bilstm\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["BILSTM", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## CNN-LSTM Model

from keras.layers import LSTM,Bidirectional,Conv1D  
  
cnn\_lstm\_model=Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
cnn\_lstm\_model.add(embedding\_layer)  
  
cnn\_lstm\_model.add(Conv1D(128, 5, activation='relu'))  
  
cnn\_lstm\_model.add(LSTM(52))  
  
cnn\_lstm\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
cnn\_lstm\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(cnn\_lstm\_model.summary())

# Model Training  
  
cnn\_lstm\_model\_history = cnn\_lstm\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
cnn\_lstm\_score = cnn\_lstm\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", cnn\_lstm\_score[0])  
print("Test Accuracy:", cnn\_lstm\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(cnn\_lstm\_model\_history.history['acc'])  
plt.plot(cnn\_lstm\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(cnn\_lstm\_model\_history.history['loss'])  
plt.plot(cnn\_lstm\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = cnn\_lstm\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
cnn\_lstm\_report=classification\_report(y\_test, y\_pred)  
print(cnn\_lstm\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(cnn\_lstm\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = cnn\_lstm\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(cnn\_lstm\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["CNN-LSTM", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## CNN-BI LSTM MODEL

from keras.layers import LSTM,Bidirectional,Conv1D  
  
cnn\_bilstm\_model=Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
cnn\_bilstm\_model.add(embedding\_layer)  
  
cnn\_bilstm\_model.add(Conv1D(164, 5, activation='relu'))  
  
cnn\_bilstm\_model.add(Bidirectional(LSTM(48)))  
cnn\_bilstm\_model.add(Dropout(0.1))  
  
cnn\_bilstm\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
cnn\_bilstm\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(cnn\_bilstm\_model.summary())

# Model Training  
  
cnn\_bilstm\_model\_history = cnn\_bilstm\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
cnn\_bilstm\_score = cnn\_bilstm\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
  
print("Test Score:", cnn\_bilstm\_score[0])  
print("Test Accuracy:", cnn\_bilstm\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(cnn\_bilstm\_model\_history.history['acc'])  
plt.plot(cnn\_bilstm\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(cnn\_bilstm\_model\_history.history['loss'])  
plt.plot(cnn\_bilstm\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = cnn\_bilstm\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
cnn\_bilstm\_report=classification\_report(y\_test, y\_pred)  
print(cnn\_bilstm\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(cnn\_bilstm\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = cnn\_bilstm\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(cnn\_bilstm\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["CNN-BILSTM", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

## HYBRID NEURAL NETWORK (HNN) MODEL

from keras.layers import LSTM,Bidirectional,Conv1D,Flatten,MaxPooling1D,Dropout  
from tensorflow.keras.initializers import RandomNormal, Constant  
  
hybrid\_model=Sequential()  
embedding\_layer = Embedding(vocab\_length, 100, weights=[embedding\_matrix], input\_length=maxlen , trainable=False)  
  
hybrid\_model.add(embedding\_layer)  
  
hybrid\_model.add(Conv1D(500,5,activation='relu'))  
hybrid\_model.add(Conv1D(200,5,activation='relu'))  
hybrid\_model.add(MaxPooling1D(pool\_size=3,strides=1, padding='same'))  
  
hybrid\_model.add(Dropout(0.1))  
  
hybrid\_model.add(Bidirectional(LSTM(60,return\_sequences=True)))  
hybrid\_model.add(LSTM(64))  
  
hybrid\_model.add(Flatten())  
hybrid\_model.add(Dense(24))  
  
hybrid\_model.add(Dense(1, activation='sigmoid'))

# Model compiling  
  
hybrid\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['acc'])  
print(hybrid\_model.summary())

# Model Training  
  
hybrid\_model\_history = hybrid\_model.fit(X\_train, y\_train, batch\_size=156, epochs=5, verbose=1, validation\_split=0.20)

# Predictions on the Test Set  
  
hybrid\_score = hybrid\_model.evaluate(X\_test, y\_test, verbose=1)

# Model Performance  
print("Test Score:", hybrid\_score[0])  
print("Test Accuracy:", hybrid\_score[1])

# Model Performance Charts  
  
import matplotlib.pyplot as plt  
  
plt.plot(hybrid\_model\_history.history['acc'])  
plt.plot(hybrid\_model\_history.history['val\_acc'])  
  
plt.title('model accuracy')  
plt.ylabel('accuracy')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()  
  
plt.plot(hybrid\_model\_history.history['loss'])  
plt.plot(hybrid\_model\_history.history['val\_loss'])  
  
plt.title('model loss')  
plt.ylabel('loss')  
plt.xlabel('epoch')  
plt.legend(['train','test'], loc='upper left')  
plt.show()

from sklearn.metrics import classification\_report, confusion\_matrix, balanced\_accuracy\_score, ConfusionMatrixDisplay  
y\_pred\_proba = hybrid\_model.predict(X\_test)  
y\_pred = np.array([0 if proba < 0.5 else 1 for proba in y\_pred\_proba])  
hnn\_report=classification\_report(y\_test, y\_pred)  
print(hnn\_report)

import tensorflow as tf  
from sklearn.metrics import f1\_score, precision\_score, recall\_score, accuracy\_score  
  
y\_pred = tf.argmax(hybrid\_model.predict(X\_test), axis=-1)  
y\_true = tf.argmax(y\_test, axis=-1)  
  
# Obtain the predicted probabilities for each input instance  
y\_pred\_prob = hybrid\_model.predict(X\_test)  
  
# Apply a threshold to convert the probabilities to binary labels  
y\_pred = (y\_pred\_prob >= 0.5).astype(int)  
  
# Calculate the f1 score  
f1 = f1\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the precision score  
precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the recall score  
recall = recall\_score(y\_test, y\_pred, average='weighted')  
  
# Calculate the accuracy score  
accuracy = round(hybrid\_score[1]\*100,2)  
  
  
# Print the scores  
print('F1 score: {:.4f}'.format(f1))  
print("Precision: {:.2f}%".format(precision \* 100))  
print("Recall Score: {:.2f}".format(recall))  
print("Accuracy:", accuracy)

data.append(["HYBRID", f1, precision, recall, accuracy])

cm = ConfusionMatrixDisplay(confusion\_matrix(y\_test, y\_pred), display\_labels = ['Negative', 'Positive'])  
plt.figure(figsize = (5,5))  
cm.plot()  
plt.title('Confusion matrix')  
plt.show()

# Saving the model

# Saving the model as a h5 file for possible use later  
  
snn\_model.save(f"./c1\_snn\_model\_acc\_{round(snn\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
cnn\_model.save(f"./c1\_cnn\_model\_acc\_{round(cnn\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
cnnop\_model.save(f"./c1\_cnnop\_model\_acc\_{round(cnnop\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
lstm\_model.save(f"./c1\_lstm\_model\_acc\_{round(lstm\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
bilstm\_model.save(f"./c1\_bilstm\_model\_acc\_{round(bilstm\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
cnn\_lstm\_model.save(f"./c1\_cnn\_lstm\_model\_acc\_{round(cnn\_lstm\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
cnn\_bilstm\_model.save(f"./c1\_cnn\_bilstm\_model\_acc\_{round(cnn\_bilstm\_score[1], 3)}.h5", save\_format='h5')

# Saving the model as a h5 file for possible use later  
  
hybrid\_model.save(f"./c1\_hybrid\_model\_acc\_{round(hybrid\_score[1], 3)}.h5", save\_format='h5')

# COMPARISION RESULTS

train\_acc = []  
val\_acc = []  
train\_loss = []  
val\_loss = []  
  
train\_acc.append(snn\_model\_history.history['acc'])  
val\_acc.append(snn\_model\_history.history['val\_acc'])  
train\_loss.append(snn\_model\_history.history['loss'])  
val\_loss.append(snn\_model\_history.history['val\_loss'])  
  
train\_acc.append(cnn\_model\_history.history['acc'])  
val\_acc.append(cnn\_model\_history.history['val\_acc'])  
train\_loss.append(cnn\_model\_history.history['loss'])  
val\_loss.append(cnn\_model\_history.history['val\_loss'])  
  
train\_acc.append(cnnop\_model\_history.history['acc'])  
val\_acc.append(cnnop\_model\_history.history['val\_acc'])  
train\_loss.append(cnnop\_model\_history.history['loss'])  
val\_loss.append(cnnop\_model\_history.history['val\_loss'])  
  
train\_acc.append(lstm\_model\_history.history['acc'])  
val\_acc.append(lstm\_model\_history.history['val\_acc'])  
train\_loss.append(lstm\_model\_history.history['loss'])  
val\_loss.append(lstm\_model\_history.history['val\_loss'])  
  
train\_acc.append(bilstm\_model\_history.history['acc'])  
val\_acc.append(bilstm\_model\_history.history['val\_acc'])  
train\_loss.append(bilstm\_model\_history.history['loss'])  
val\_loss.append(bilstm\_model\_history.history['val\_loss'])  
  
train\_acc.append(cnn\_lstm\_model\_history.history['acc'])  
val\_acc.append(cnn\_lstm\_model\_history.history['val\_acc'])  
train\_loss.append(cnn\_lstm\_model\_history.history['loss'])  
val\_loss.append(cnn\_lstm\_model\_history.history['val\_loss'])  
  
train\_acc.append(cnn\_bilstm\_model\_history.history['acc'])  
val\_acc.append(cnn\_bilstm\_model\_history.history['val\_acc'])  
train\_loss.append(cnn\_bilstm\_model\_history.history['loss'])  
val\_loss.append(cnn\_bilstm\_model\_history.history['val\_loss'])  
  
train\_acc.append(hybrid\_model\_history.history['acc'])  
val\_acc.append(hybrid\_model\_history.history['val\_acc'])  
train\_loss.append(hybrid\_model\_history.history['loss'])  
val\_loss.append(hybrid\_model\_history.history['val\_loss'])

import pandas as pd  
models=["SNN","CNN","CNN-OP","LSTM","BI-LSTM","CNN-LSTM","CNN-BILSTM","HYBRID"]  
accuracy = []  
accuracy.append(round(snn\_score[1]\*100,2))  
accuracy.append(round(cnn\_score[1]\*100,2))  
accuracy.append(round(cnnop\_score[1]\*100,2))  
accuracy.append(round(lstm\_score[1]\*100,2))  
accuracy.append(round(bilstm\_score[1]\*100,2))  
accuracy.append(round(cnn\_lstm\_score[1]\*100,2))  
accuracy.append(round(cnn\_bilstm\_score[1]\*100,2))  
accuracy.append(round(hybrid\_score[1]\*100,2))  
  
# Sample data  
accuracy = np.array(accuracy)  
  
# Convert to percent with one decimal place  
accuracy = np.round(accuracy, 1)  
  
metrics = pd.DataFrame({  
 'Model': models,  
 'Accuracy': accuracy,  
})  
  
# Set Model column as index  
metrics = metrics.set\_index('Model')  
  
# Display DataFrame  
print(metrics)  
  
# Print the results

import matplotlib.pyplot as plt  
  
  
# Create a bar chart  
plt.close('all')  
colors = ['red', 'green', 'blue','yellow','brown','pink','purple','orange']  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.bar(models, accuracy,color=colors,width=0.5)  
  
# Add horizontal grid lines  
plt.grid(True, axis='y')  
  
# Add a title and axis labels  
plt.title('Model Accuracies')  
plt.xlabel('Model')  
plt.ylabel('Accuracy')  
  
# Rotate x-axis labels  
# plt.xticks(rotation=45)  
  
# Add value labels on top of each bar  
for i, v in enumerate(accuracy):  
 plt.text(i, v, str(v), ha='center', va='bottom', fontweight='bold')  
  
# Display the chart  
plt.show()

import numpy as np  
import matplotlib.pyplot as plt  
  
# extract the accuracy, precision, recall, and f1 values for each model  
accuracy = [x[1] for x in data]  
precision = [x[2] for x in data]  
recall = [x[3] for x in data]  
f1 = [(x[4]/100) for x in data]  
  
# set the width of the bars  
barWidth = 0.20  
  
# set the positions of the bars on the x-axis  
r1 = np.arange(len(data))  
r2 = [x + barWidth for x in r1]  
r3 = [x + barWidth for x in r2]  
r4 = [x + barWidth for x in r3]  
  
# create the bar chart  
plt.subplots(figsize=(14,10))  
bar1=plt.bar(r1, accuracy, color='b', width=barWidth, edgecolor='white', label='Accuracy')  
bar2=plt.bar(r2, precision, color='g', width=barWidth, edgecolor='white', label='Precision')  
bar3=plt.bar(r3, recall, color='r', width=barWidth, edgecolor='white', label='Recall')  
bar4=plt.bar(r4, f1, color='c', width=barWidth, edgecolor='white', label='F1 Score')  
  
# add labels and titles  
plt.xlabel('Model')  
plt.xticks([r + barWidth for r in range(len(data))],[x[0] for x in data])  
plt.ylabel('Score')  
plt.title('Performance Metrics')  
  
# add legend  
plt.legend(bbox\_to\_anchor=(1.05, 1.0), loc='upper left')  
  
# add text labels above each bar  
for i in range(len(data)):  
 plt.text(r1[i],accuracy[i], '{:.2f}'.format(accuracy[i]), fontsize=8,ha='center')  
 plt.text(r2[i],precision[i], '{:.2f}'.format(precision[i]), fontsize=8,ha='center')  
 plt.text(r3[i],recall[i], '{:.2f}'.format(recall[i]), fontsize=8,ha='center')  
 plt.text(r4[i],f1[i], '{:.2f}'.format(f1[i]), fontsize=8,ha='center')  
  
# show the plot  
plt.show()

# GRAPH COMPARISION

import sys  
plt.close('all')  
  
# Create the plot for train accuracy  
x = range(len(train\_acc[0]))  
# print(x)  
# sys.exit()  
fig1 = plt.figure()  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.plot(x, train\_acc[0], label="SNN")  
plt.plot(x, train\_acc[1], label="CNN")  
plt.plot(x, train\_acc[2], label="CNN-OP")  
plt.plot(x, train\_acc[3], label="LSTM")  
plt.plot(x, train\_acc[4], label="BILSTM")  
plt.plot(x, train\_acc[5], label="CNN-LSTM")  
plt.plot(x, train\_acc[6], label="CNN-BILSTM")  
plt.plot(x, train\_acc[7], label="HYBRID")  
plt.legend()  
plt.xlabel('Epochs')  
plt.ylabel('Accuracy')  
plt.title('Train Accuracy over time')  
  
  
# Create the plot for validation accuracy  
fig2 = plt.figure()  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.plot(x, val\_acc[0], label="SNN")  
plt.plot(x, val\_acc[1], label="CNN")  
plt.plot(x, val\_acc[2], label="CNN-OP")  
plt.plot(x, val\_acc[3], label="LSTM")  
plt.plot(x, val\_acc[4], label="BILSTM")  
plt.plot(x, val\_acc[5], label="CNN-LSTM")  
plt.plot(x, val\_acc[6], label="CNN-BILSTM")  
plt.plot(x, val\_acc[7], label="HYBRID")  
plt.legend()  
plt.xlabel('Epochs')  
plt.ylabel('Accuracy')  
plt.title('Validation Accuracy over time')  
  
# Create the plot for train loss  
fig3 = plt.figure()  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.plot(x, train\_loss[0], label="SNN")  
plt.plot(x, train\_loss[1], label="CNN")  
plt.plot(x, train\_loss[2], label="CNN-OP")  
plt.plot(x, train\_loss[3], label="LSTM")  
plt.plot(x, train\_loss[4], label="BILSTM")  
plt.plot(x, train\_loss[5], label="CNN-LSTM")  
plt.plot(x, train\_loss[6], label="CNN-BILSTM")  
plt.plot(x, train\_loss[7], label="HYBRID")  
plt.legend()  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.title('Train Loss over time')  
  
# Create the plot for validation loss  
fig4 = plt.figure()  
plt.rcParams["figure.figsize"] = (12,5.5)  
plt.plot(x, val\_acc[0], label="SNN")  
plt.plot(x, val\_acc[1], label="CNN")  
plt.plot(x, val\_acc[2], label="CNN-OP")  
plt.plot(x, val\_acc[3], label="LSTM")  
plt.plot(x, val\_acc[4], label="BILSTM")  
plt.plot(x, val\_acc[5], label="CNN-LSTM")  
plt.plot(x, val\_acc[6], label="CNN-BILSTM")  
plt.plot(x, val\_acc[7], label="HYBRID")  
plt.legend()  
plt.xlabel('Epochs')  
plt.ylabel('Loss')  
plt.title('Validation Loss over time')  
  
# Show all the plots  
plt.show()

# Making Predictions on Live IMDb data

# Load previously trained LSTM Model  
  
from keras.models import load\_model  
  
model\_path ='c1\_hybrid\_model\_acc\_0.864.h5'  
pretrained\_lstm\_model = load\_model(model\_path)  
  
# summarize model.  
pretrained\_lstm\_model.summary()

import chardet  
file = 'a3\_IMDb\_Unseen\_Reviews.csv'  
with open(file, 'rb') as rawdata:  
 result = chardet.detect(rawdata.read(100000))  
file\_type = result['encoding']

# Load sample IMDb reviews csv, having ~6 movie reviews, along with their IMDb rating  
if (file\_type == 'UTF-8-SIG'):  
 sample\_reviews = pd.read\_csv(file)  
 print(sample\_reviews.head(8))  
else:  
 print("CSV file not of encoding: UTF-8-SIG type")  
 exit()

# Preprocess review text with earlier defined preprocess\_text function  
  
unseen\_reviews = sample\_reviews['Review Text']  
  
unseen\_processed = []  
for review in unseen\_reviews:  
 review = convert.conversion(review)  
 review = clean\_text(review)  
 unseen\_processed.append(review)

unseen\_processed

from keras\_preprocessing.text import tokenizer\_from\_json

# Loading  
with open('b3\_tokenizer.json') as f:  
 data = json.load(f)  
 loaded\_tokenizer = tokenizer\_from\_json(data)

# Tokenising instance with earlier trained tokeniser  
unseen\_tokenized = loaded\_tokenizer.texts\_to\_sequences(unseen\_processed)

unseen\_tokenized

# Pooling instance to have maxlength of 100 tokens  
unseen\_padded = pad\_sequences(unseen\_tokenized, padding='post', maxlen=100)

unseen\_padded

# Passing tokenised instance to the LSTM model for predictions  
unseen\_sentiments = pretrained\_lstm\_model.predict(unseen\_padded)  
  
unseen\_sentiments

# Writing model output file back to Google Drive  
  
sample\_reviews['Predicted Sentiments'] = np.round(unseen\_sentiments\*10,1)  
  
df\_prediction\_sentiments = pd.DataFrame(sample\_reviews['Predicted Sentiments'], columns = ['Predicted Sentiments'])  
df\_movie = pd.DataFrame(sample\_reviews['Movie'], columns = ['Movie'])  
df\_review\_text = pd.DataFrame(sample\_reviews['Review Text'], columns = ['Review Text'])  
df\_imdb\_rating = pd.DataFrame(sample\_reviews['IMDb Rating'], columns = ['IMDb Rating'])  
df\_prediction\_sentimenta = pd.DataFrame(sample\_reviews['Predicted Sentiments'], columns = ['Sampleentiments'])  
  
dfx=pd.concat([df\_movie, df\_review\_text, df\_imdb\_rating, df\_prediction\_sentiments], axis=1)  
  
dfx.to\_csv("c2\_IMDb\_Unseen\_Predictions.csv", sep=',', encoding = 'UTF-8-SIG')  
  
unseen\_sentiments[0][0]

prediction\_text= f"Positive Review with probable IMDb rating as: {np.round(unseen\_sentiments[0][0]\*10,1)}"

prediction\_text