In [ ]:
```
# Load the Drive helper and mount
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Importing the dependencies

In [54]:
```
'''
Following packages, libraries and frameworks are used for data understanding, cleaning and modeling purpose.
1. NumPy : Performing image and array manipulations
2. Pillow : Loading and working with jpg images
3. torch : PyTorch framework for working with tensors, GPU and models
4. Sklearn : Useful library that provides very useful functions mainly used to evaluate the performance of our model
Lets start with the number of classes and number of samples in each class in the entire dataset.
5. Seaborn and Matplotlib : Data visualization
'''

import os
import torch
import torch.nn as nn
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import random
from tqdm import tqdm
import pandas as pd
import seaborn as sns
%matplotlib inline

matplotlib.rcParams['figure.facecolor'] = '#ffffff'
```

In [55]:
```
import warnings
warnings.filterwarnings("ignore")
```

Data Understanding

In [56]:
```
df = pd.read_csv('IMDB Dataset.csv') # reading the csv file
```

In [57]:
```
df.head()
```

Out[57]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

In [58]:
```
df.shape
```

Out[58]:
```
(50000, 2)
```

In [59]:
```
df.sentiment.value_counts()
```

Out[59]:
```
positive    25000
negative    25000
Name: sentiment, dtype: int64
```

In [60]:
```
df.sentiment = df.sentiment.replace({'positive': 1, 'negative': 0})
```

In [61]:
```
df.head()
```

Out[61]:

	review	sentiment
0	One of the other reviewers has mentioned that ...	1
1	A wonderful little production. The...	1
2	I thought this was a wonderful way to spend ti...	1
3	Basically there's a family where a little boy ...	0
4	Petter Mattei's "Love in the Time of Money" is...	1

In [62]:
```
from wordcloud import WordCloud
```

In [ ]:

In [ ]:

```

```

Removing URLs from the text

In [63]:

```
import re
def remove_urls(text):
    url_pattern = re.compile(r'https?://\S+|www\.\S+')
    return url_pattern.sub(r'', text)
```

Remove HTML tags

In [64]:

```
def rm_html(text):
    return re.sub(r'<[^>]+>', '', text)
```

Remove Emojis

In [65]:

```
def remove_emoji(string):
    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' ', string)
```

Remove Unwanted Characters

In [66]:

```
def removeunwanted_characters(document):
    # remove user mentions
    document = re.sub("@[A-Za-z0-9_]+"," ", document)
    # remove hashtags
    document = re.sub("#[A-Za-z0-9_]+","", document)
    # remove punctuation
    document = re.sub("[^0-9A-Za-z ]", "" , document)
    #remove emojis
    document = remove_emoji(document)
    # remove double spaces
```

```
    document = document.replace('  ',"")
    return document.strip()
```

Remove unnecessary whitespaces

In [67]:
```
def rm_whitespaces(text):
    return re.sub(r' +', ' ', text)
```

Remove Punctutations

In [68]:
```
from nltk.tokenize import RegexpTokenizer
from nltk.tokenize import RegexpTokenizer

def remove_punct(text):
    tokenizer = RegexpTokenizer(r"\w+")
    lst=tokenizer.tokenize(' '.join(text))
    return lst
```

Remove Stopwords

In [69]:
```
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
custom_stopwords = ['@', 'RT']
stop_words.update(custom_stopwords)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [70]:
```
def remove_stopwords(text_tokens):
    result_tokens = []
    for token in text_tokens:
        if token not in stop_words:
            result_tokens.append(token)
    return result_tokens
```

In [ ]:

Convert to lower order

In [71]:
```
def lower_order(text):
    small_order_text = text.lower()
    return small_order_text
```

Stemming

In [72]:
```
from nltk.stem import WordNetLemmatizer
from nltk import word_tokenize,pos_tag
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

def lemmatization(token_text):
    lemma_tokens = []
    wordnet = WordNetLemmatizer()
    lemmatized_tokens = [wordnet.lemmatize(token, pos = 'v') for token in token_text]
    return lemmatized_tokens
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Tokenization

In [73]:
```
import nltk
nltk.download('punkt')
from nltk import word_tokenize

# preprocessing
def tokenize(text):
    return word_tokenize(text)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Lenovo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Custom Pipeline to clean text

In [74]:
```
def custom_cleaning_pipeline(text):
    text = lower_order(text)
    text = remove_urls(text)
    text = rm_html(text)
    text = remove_emoji(text)
    text = removeunwanted_characters(text)
    text = rm_whitespaces(text)
    return text
```

In [75]:
```
custom_cleaning_pipeline(df['review'].iloc[0])
```

Out[75]:
```
'one of the other reviewers has mentioned that after watching just 1 oz episode youll be hooked they are right as this
is exactly what happened with methe first thing that struck me about oz was its brutality and unflinching scenes of vio
lence which set in right from the word go trust me this is not a show for the faint hearted or timid this show pulls no
punches with regards to drugs sex or violence its is hardcore in the classic use of the wordit is called oz as that is
the nickname given to the oswald maximum security state penitentary it focuses mainly on emerald city an experimental s
ection of the prison where all the cells have glass fronts and face inwards so privacy is not high on the agenda em cit
y is home to manyaryans muslims gangstas latinos christians italians irish and moreso scuffles death stares dodgy deali
ngs and shady agreements are never far awayi would say the main appeal of the show is due to the fact that it goes wher
e other shows wouldnt dare forget pretty pictures painted for mainstream audiences forget charm forget romanceoz doesnt
mess around the first episode i ever saw struck me as so nasty it was surreal i couldnt say i was ready for it but as i
watched more i developed a taste for oz and got accustomed to the high levels of graphic violence not just violence but
injustice crooked guards wholl be sold out for a nickel inmates wholl kill on order and get away with it well mannered
middle class inmates being turned into prison bitches due to their lack of street skills or prison experience watching
oz you may become comfortable with what is uncomfortable viewingthats if you can get in touch with your darker side'
```

After cleaning the messy text data, we need to preprocess it. First, we will tokenize the text and perform Lemmatization. After tokenizaation, we will remove punctuations and stopwords. For Tokenization, we will split the entire text into individual words by using nltk package. Furthermore, we will lemmatize text using nltk by transforming a word into its original form. We can do that by removing their suffix respectively.

In [76]:
```
def custom_preprocessing_pipeline(text):
    text = tokenize(text)
    text = remove_punct(text)
    text = remove_stopwords(text)
    lemmatized_text = lemmatization(text)
    return " ".join(lemmatized_text)
```

In [77]:
```
custom_preprocessing_pipeline("Environment\\him")
```

Out[77]:
```
'Environment'
```

In [78]:
```
%%time

df['cleaned'] = df.review.apply(lambda x : custom_cleaning_pipeline(x))
```

```
CPU times: total: 5.06 s
Wall time: 7.91 s
```

In [79]:
```
%%time

df['cleaned'] = df.cleaned.apply(lambda x : custom_preprocessing_pipeline(x))
```

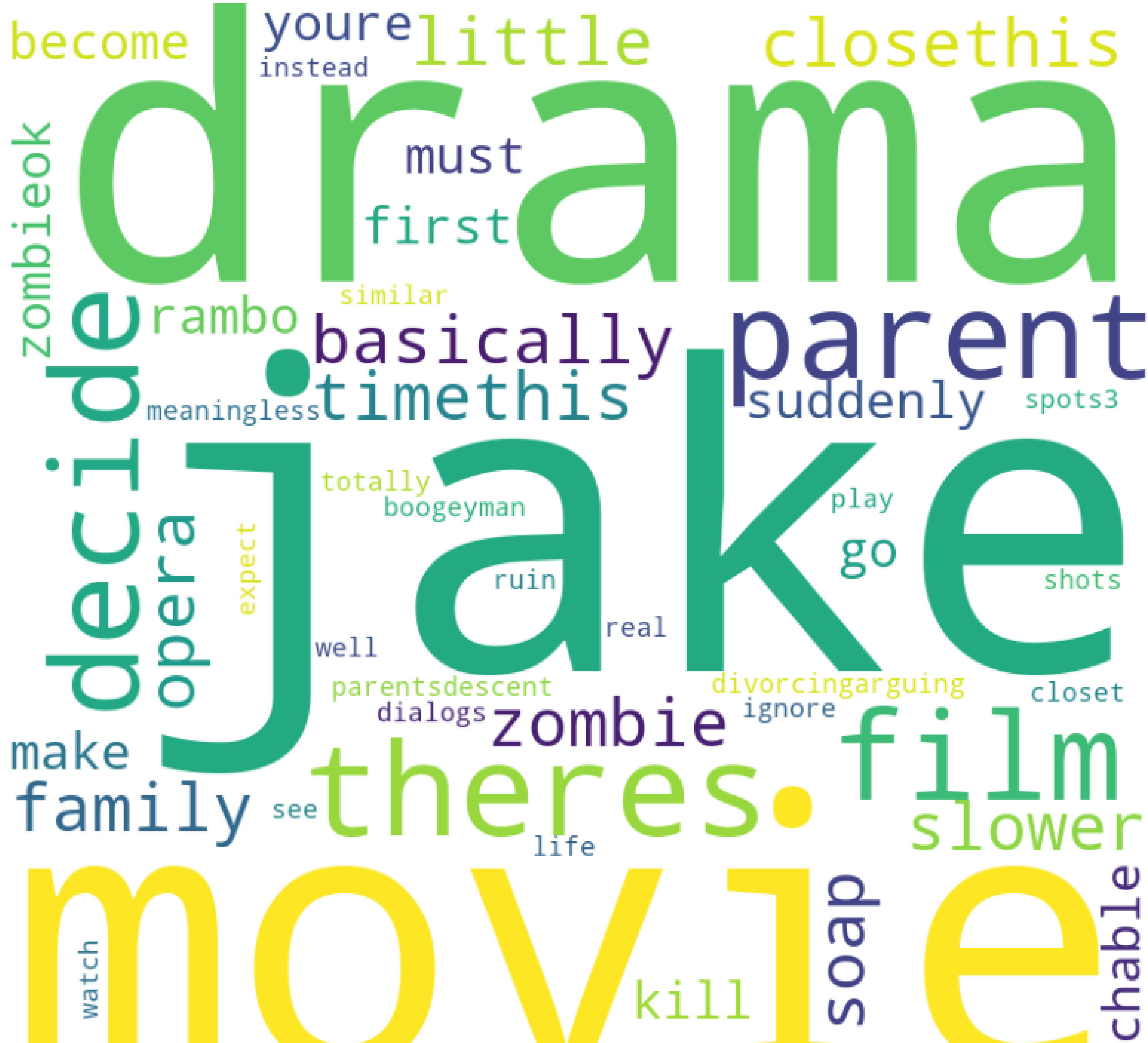
```
CPU times: total: 38.3 s
Wall time: 1min 6s
```

In [80]:
```
df = df[['cleaned', 'sentiment']]
df.rename(columns = {'sentiment': 'label'}, inplace = True)
```

In [81]:
```
# data = df.copy()
```

In [82]:
```
negative_df = df[df.label == 0]
negative_df = "".join(negative_df.cleaned.values[0])

# Create the WordCloud object
wordcloud = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate(negative_df)

# Plot the WordCloud image
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.title("Negative Reviews")
plt.axis('off')
plt.tight_layout(pad=0)

# Show the plot
plt.show()
```

Negative Reviews



In [83]:
```
negative_df = df[df.label == 1]
negative_df = "".join(negative_df.cleaned.values[0])

# Create the WordCloud object
wordcloud = WordCloud(width=800, height=800, background_color='white', min_font_size=10).generate(negative_df)

# Plot the WordCloud image
plt.figure(figsize=(8, 8), facecolor=None)
plt.imshow(wordcloud)
plt.title("Positive Reviews")
plt.axis('off')
plt.tight_layout(pad=0)

# Show the plot
plt.show()
```

Positive Reviews

This plot is called word cloud plot. It is mainly created to understand the most common words in a text. Here, the size of the text in this plot represents the frequency of the occurence of the text.

Now that we have properly cleaned our dataset, it is time to create vocabulary for our words next

In [85]:
```
# lets get all of the cleaned reviews volumn values
complete_reviews = df.cleaned.values
```

In [86]:
```
# combining all words with whitespaces as seperatuion
all_words = ' '.join(complete_reviews)
```

In [87]:
```
# converting to the list
all_words = all_words.split()
```

In [88]:
```
# lets too at first 20 words
all_words[:20]
```

Out[88]:
```
['one',
 'reviewers',
 'mention',
 'watch',
 '1',
 'oz',
 'episode',
 'youll',
 'hook',
 'right',
 'exactly',
 'happen',
 'methe',
 'first',
 'thing',
 'strike',
 'oz',
 'brutality',
 'unflinching',
 'scenes']
```

In [89]:
```
from collections import Counter
```

Next, we will proceed to get the number of counts a word appears in the text corpus. We can calculate that using Counter library, which is a built-in python library. We will proceed to start the index of our unique words from 1. This is so to make sure that we will use the index postiion 0 for the padding `<PAD>` token.

In [90]:
```
# passing all words to Counter function to count the frequency of each word in the list.
counter = Counter(all_words)
```

In [91]:
```
counter
```

Out[91]:
```
Counter({'one': 49988,
         'reviewers': 496,
         'mention': 2972,
         'watch': 26886,
         '1': 2073,
         'oz': 247,
         'episode': 3003,
         'youll': 2558,
         'hook': 572,
         'right': 6480,
         'exactly': 1939,
         'happen': 6869,
         'methe': 93,
         'first': 16775,
         'thing': 8818,
         'strike': 967,
         'brutality': 134,
         'unflinching': 30,
         'scenes': 10057,
         'violence': 1957,
         'set': 7118,
         'word': 3493,
         'go': 26725,
         'trust': 686,
         'show': 21250,
         'faint': 115,
         'hearted': 125,
         'timid': 45,
         'pull': 1843,
         'punch': 521,
         'regard': 925,
         'drug': 1619,
         'sex': 3176,
         'hardcore': 249,
         'classic': 3395,
         'use': 9802,
         'wordit': 2,
         'call': 5383,
         'nickname': 83,
         'give': 17142,
         'oswald': 28,
         'maximum': 96,
         'security': 359,
         'state': 1981,
         'penitentary': 2,
```

```
'focus': 1876,
'mainly': 748,
'emerald': 13,
'city': 2170,
'experimental': 167,
'section': 447,
'prison': 877,
'cells': 95,
'glass': 487,
'front': 1211,
'face': 4091,
'inwards': 2,
'privacy': 25,
'high': 3783,
'agenda': 151,
'em': 224,
'home': 3612,
'manyaryans': 1,
'muslims': 75,
'gangstas': 5,
'latinos': 31,
'christians': 147,
'italians': 81,
'irish': 375,
'moreso': 24,
'scuffle': 17,
'death': 3640,
'star': 7842,
'dodgy': 87,
'deal': 2816,
'shady': 72,
'agreements': 7,
'never': 12793,
'far': 5690,
'awayi': 18,
'would': 23952,
'say': 18851,
'main': 4571,
'appeal': 1333,
'due': 1752,
'fact': 6804,
'wouldnt': 2091,
'dare': 623,
'forget': 2581,
'pretty': 7149,
```

```
'picture': 3623,
'paint': 956,
'mainstream': 373,
'audiences': 974,
'charm': 1702,
'romanceoz': 1,
'doesnt': 8844,
'mess': 1419,
'around': 6943,
'ever': 11570,
'saw': 6299,
'nasty': 640,
'surreal': 424,
'couldnt': 3027,
'ready': 653,
'develop': 1581,
'taste': 1027,
'get': 35361,
'accustom': 58,
'level': 2166,
'graphic': 454,
'injustice': 79,
'crook': 247,
'guard': 504,
'wholl': 35,
'sell': 1130,
'nickel': 11,
'inmates': 97,
'kill': 7167,
'order': 2251,
'away': 5243,
'well': 18608,
'mannered': 60,
'middle': 1557,
'class': 1618,
'turn': 7388,
'bitch': 155,
'lack': 3588,
'street': 1218,
'skills': 481,
'experience': 2964,
'may': 6509,
'become': 7542,
'comfortable': 223,
'uncomfortable': 259,
```

```
'viewingthats': 1,
'touch': 2359,
'darker': 204,
'side': 2730,
'wonderful': 3134,
'little': 12292,
'production': 3343,
'film': 91261,
'technique': 291,
'unassuming': 23,
'oldtimebbc': 1,
'fashion': 836,
'comfort': 230,
'sometimes': 2209,
'discomforting': 11,
'sense': 4590,
'realism': 503,
'entire': 2852,
'piece': 3770,
'actors': 8660,
'extremely': 2109,
'choose': 1529,
'michael': 2309,
'sheen': 167,
'polari': 1,
'voice': 2536,
'pat': 274,
'truly': 3411,
'see': 40434,
'seamless': 36,
'edit': 2068,
'guide': 423,
'reference': 912,
'williams': 591,
'diary': 87,
'entries': 104,
'worth': 4564,
'terrificly': 5,
'write': 7666,
'perform': 1131,
'masterful': 187,
'great': 17619,
'master': 1222,
'comedy': 6006,
'life': 11540,
```

```
'really': 22835,
'come': 16179,
'things': 7124,
'fantasy': 1163,
'rather': 5218,
'traditional': 450,
'dream': 1933,
'techniques': 275,
'remain': 1736,
'solid': 932,
'disappear': 577,
'play': 16786,
'knowledge': 547,
'particularly': 1992,
'concern': 1136,
'orton': 19,
'halliwell': 10,
'flat': 982,
'halliwells': 1,
'murals': 2,
'decorate': 71,
'every': 7815,
'surface': 448,
'terribly': 512,
'do': 5828,
'think': 23859,
'way': 14853,
'spend': 2817,
'time': 29466,
'hot': 1248,
'summer': 717,
'weekend': 384,
'sit': 3106,
'air': 1581,
'condition': 485,
'theater': 1506,
'lighthearted': 207,
'plot': 12884,
'simplistic': 202,
'dialogue': 2932,
'witty': 518,
'character': 27326,
'likable': 692,
'even': 24470,
'bread': 104,
```

```
'suspect': 968,
'serial': 658,
'killer': 2363,
'disappoint': 2693,
'realize': 2385,
'match': 1413,
'point': 7635,
'2': 3963,
'risk': 419,
'addiction': 143,
'proof': 305,
'woody': 298,
'allen': 461,
'still': 10621,
'fully': 754,
'control': 1247,
'style': 3023,
'many': 13213,
'us': 7607,
'grow': 2148,
'lovethis': 13,
'id': 2544,
'laugh': 5768,
'woodys': 20,
'comedies': 796,
'years': 8606,
'decade': 453,
'ive': 6383,
'impress': 958,
'scarlet': 65,
'johanson': 5,
'manage': 2584,
'tone': 1017,
'sexy': 844,
'image': 1729,
'jump': 1190,
'average': 1326,
'spirit': 1268,
'young': 6889,
'womanthis': 2,
'crown': 135,
'jewel': 192,
'career': 2037,
'wittier': 8,
'devil': 560,
```

```
'wear': 1649,
'prada': 13,
'interest': 9262,
'superman': 336,
'friends': 3582,
'basically': 1694,
'theres': 5721,
'family': 5454,
'boy': 2706,
'jake': 230,
'zombie': 1050,
'closethis': 3,
'parent': 1746,
'fight': 3943,
'timethis': 38,
'movie': 82890,
'slower': 79,
'soap': 547,
'opera': 754,
'suddenly': 999,
'decide': 3461,
'rambo': 118,
'zombieok': 1,
'youre': 3793,
'make': 43801,
'must': 6144,
'thriller': 1477,
'drama': 2514,
'watchable': 569,
'divorcingarguing': 1,
'like': 42726,
'real': 8975,
'closet': 219,
'totally': 2704,
'ruin': 1038,
'expect': 4910,
'boogeyman': 47,
'similar': 1601,
'instead': 4091,
'meaningless': 216,
'spots3': 1,
'10': 4085,
'parentsdescent': 1,
'dialogs': 258,
'shots': 1839,
```

```
'ignore': 754,
'petter': 1,
'matteis': 7,
'love': 16684,
'money': 4262,
'visually': 441,
'stun': 920,
'mr': 2645,
'mattei': 41,
'offer': 2018,
'vivid': 194,
'portrait': 263,
'human': 3049,
'relations': 180,
'seem': 14022,
'tell': 8149,
'power': 2356,
'success': 1100,
'people': 17871,
'different': 4557,
'situations': 946,
'encounter': 766,
'variation': 97,
'arthur': 521,
'schnitzlers': 2,
'theme': 2331,
'director': 7201,
'transfer': 319,
'action': 6470,
'present': 2412,
'new': 7940,
'york': 1336,
'meet': 3649,
'connect': 600,
'another': 8126,
'next': 3301,
'person': 2994,
'know': 18558,
'previous': 1276,
'contact': 396,
'stylishly': 26,
'sophisticate': 254,
'luxurious': 33,
'look': 19109,
'take': 17007,
```

```
'live': 8087,
'world': 6731,
'habitatthe': 1,
'souls': 259,
'stag': 438,
'loneliness': 160,
'inhabit': 202,
'big': 6634,
'best': 12224,
'place': 5660,
'find': 15806,
'sincere': 178,
'fulfillment': 42,
'discern': 59,
'case': 3177,
'encounterthe': 2,
'act': 16549,
'good': 28285,
'direction': 2546,
'steve': 774,
'buscemi': 40,
'rosario': 52,
'dawson': 115,
'carol': 242,
'kane': 317,
'imperioli': 20,
'adrian': 94,
'grenier': 6,
'rest': 3512,
'talented': 1080,
'cast': 8317,
'alivewe': 1,
'wish': 2515,
'luck': 450,
'await': 198,
'anxiously': 24,
'work': 13063,
'probably': 5481,
'alltime': 222,
'favorite': 2331,
'story': 21841,
'selflessness': 4,
'sacrifice': 379,
'dedication': 81,
'noble': 230,
```

```
 'cause': 2042,
 'preachy': 134,
 'bore': 4765,
 'old': 7715,
 'despite': 2442,
 '15': 929,
 'last': 5995,
 '25': 372,
 'paul': 1323,
 'lukas': 61,
 'performance': 5370,
 'bring': 4719,
 'tear': 1154,
 'eye': 3646,
 'bette': 198,
 'davis': 493,
 'sympathetic': 435,
 'roles': 1974,
 'delight': 449,
 'kid': 5997,
 'grandma': 83,
 'dressedup': 1,
 'midgets': 22,
 'children': 2559,
 'fun': 5074,
 'mother': 3084,
 'slow': 2008,
 'awaken': 197,
 'whats': 1624,
 'roof': 163,
 'believable': 1364,
 'startle': 160,
 'dozen': 344,
 'thumb': 306,
 'theyd': 261,
 'sure': 5082,
 'resurrection': 67,
 'date': 1573,
 'seahunt': 2,
 'series': 6039,
 'tech': 83,
 'today': 1810,
 'back': 9375,
 'excitement': 378,
 'mei': 102,
```

```
'black': 3971,
'white': 2583,
'tv': 5271,
'gunsmoke': 20,
'heros': 139,
'weekyou': 1,
'vote': 788,
'comeback': 91,
'sea': 595,
'huntwe': 1,
'need': 6480,
'change': 3859,
'pace': 1993,
'water': 1430,
'adventureoh': 1,
'thank': 1599,
'outlet': 34,
'view': 4139,
'viewpoints': 35,
'moviesso': 10,
'ole': 39,
'believe': 5950,
'wan': 338,
'na': 750,
'saywould': 1,
'nice': 3723,
'read': 5179,
'plus': 1198,
'huntif': 1,
'rhyme': 118,
'line': 6301,
'let': 5362,
'submitor': 1,
'leave': 8402,
'doubt': 1674,
'quitif': 1,
'amaze': 2854,
'freshinnovative': 1,
'idea': 3930,
'70s': 1105,
'7': 840,
'8': 836,
'brilliant': 2265,
'drop': 941,
'1990': 143,
```

```
'funny': 8220,
'anymore': 619,
'continue': 1462,
'decline': 176,
'complete': 2207,
'waste': 4265,
'todayits': 2,
'disgraceful': 36,
'fall': 4131,
'painfully': 415,
'bad': 17467,
'performances': 3370,
'almost': 6111,
'badif': 9,
'mildly': 356,
'entertain': 3564,
'respite': 25,
'guesthosts': 1,
'hard': 4700,
'creator': 143,
'handselected': 1,
'original': 6130,
'also': 17420,
'band': 1160,
'hack': 286,
'follow': 4147,
'recognize': 690,
'brilliance': 230,
'fit': 1690,
'replace': 564,
'mediocrity': 98,
'felt': 2849,
'respect': 1193,
'huge': 1903,
'awful': 3130,
'cant': 7471,
'encourage': 345,
'positive': 949,
'comment': 3029,
'forward': 1253,
'mistake': 1279,
'950': 6,
'worst': 5200,
'themits': 4,
'storyline': 1532,
```

```
'soundtrack': 1570,
'songa': 2,
'lame': 1301,
'country': 1696,
'tuneis': 1,
'less': 3501,
'four': 1698,
'cheap': 1690,
'extreme': 686,
'rarely': 609,
'happy': 1715,
'end': 17972,
'credit': 2418,
'prevent': 389,
'1score': 1,
'harvey': 168,
'keitelwhile': 1,
'least': 6006,
'bite': 6309,
'effort': 1473,
'keitel': 57,
'obsessives': 4,
'gut': 346,
'wrench': 91,
'laughter': 428,
'hell': 2126,
'mom': 644,
'itgreat': 5,
'camp': 1062,
'phil': 176,
'alien': 1379,
'quirky': 355,
'humour': 905,
'base': 3050,
'oddness': 18,
'everything': 4552,
'actual': 1481,
'punchlinesat': 1,
'odd': 1062,
'progress': 456,
'didnt': 8765,
'joke': 3087,
'anymoreits': 2,
'low': 2591,
'budget': 3165,
```

```
'thats': 7086,
'problem': 2712,
'eventually': 1352,
'lose': 4335,
'interesti': 4,
'imagine': 1712,
'stoner': 55,
'currently': 232,
'partakingfor': 1,
'something': 9688,
'better': 10975,
'try': 12292,
'brother': 1802,
'planet': 802,
'12': 941,
'recall': 588,
'scariest': 112,
'scene': 10422,
'bird': 417,
'eat': 1388,
'men': 3489,
'dangle': 53,
'helplessly': 23,
'parachute': 23,
'horror': 6582,
'horroras': 2,
'cheesy': 1253,
'b': 1153,
'saturday': 384,
'afternoons': 19,
'tire': 925,
'formula': 442,
'monster': 1456,
'type': 2539,
'movies': 15306,
'usually': 1905,
'include': 3795,
'hero': 1778,
'beautiful': 4075,
'woman': 4857,
'might': 5722,
'daughter': 2004,
'professor': 389,
'resolution': 221,
'die': 3776,
```

```
'care': 3609,
'much': 18815,
'romantic': 1572,
'angle': 727,
'year': 3448,
'predictable': 1587,
'unintentional': 201,
'humorbut': 4,
'later': 4032,
'psycho': 386,
'janet': 109,
'leigh': 156,
'bump': 181,
'early': 2998,
'notice': 1451,
'since': 5516,
'screenwriters': 129,
'scary': 1721,
'possible': 1832,
'wellworn': 20,
'rule': 909,
'im': 8891,
'fan': 6440,
'bolls': 29,
'enjoy': 6362,
'postal': 25,
'maybe': 4345,
'boll': 175,
'apparently': 1716,
'buy': 2849,
'cry': 1429,
'long': 6431,
'ago': 1895,
'game': 3251,
'finsished': 1,
'mercs': 2,
'infiltrate': 64,
'secret': 1096,
'research': 523,
'labs': 14,
'locate': 255,
'tropical': 61,
'island': 1217,
'warn': 1283,
'scheme': 341,
```

```
'together': 4202,
'along': 3604,
'legion': 50,
'schmucks': 4,
'feel': 10066,
'loneley': 1,
'invite': 422,
'three': 4358,
'countrymen': 11,
'players': 542,
'name': 5415,
'til': 70,
'schweiger': 15,
'udo': 34,
'kier': 25,
'ralf': 9,
'moellerthree': 1,
'actually': 8318,
'selfs': 3,
'biz': 29,
'tale': 1459,
'jack': 1755,
'carver': 25,
'yes': 2596,
'german': 1069,
'hail': 122,
'bratwurst': 3,
'dudes': 65,
'however': 6357,
'tils': 1,
'badass': 63,
'complain': 532,
'hes': 5623,
'stay': 2397,
'true': 4302,
'whole': 6059,
'perspective': 450,
'dont': 16519,
'kick': 1070,
'beyond': 1799,
'demented': 139,
'evil': 2571,
'mad': 953,
'scientist': 615,
'dr': 1434,
```

```
'krieger': 2,
'geneticallymutatedsoldiers': 1,
'gms': 5,
'topsecret': 13,
'remind': 1644,
'spoiler': 580,
'vancouver': 39,
'reason': 5564,
'palm': 99,
'tree': 517,
'rich': 1149,
'lumberjackwoods': 1,
'havent': 1630,
'start': 7928,
'mehehe': 1,
'shenanigans': 61,
'deliver': 2010,
'mean': 6721,
'suckthere': 2,
'imply': 283,
'areas': 230,
'boat': 609,
'cromedalbino': 1,
'squad': 186,
'enter': 894,
'reek': 93,
'scheisse': 1,
'poop': 46,
'simpletons': 6,
'wiff': 2,
'ahead': 682,
'btw': 95,
'annoy': 2208,
'sidekick': 218,
'shoot': 5872,
'minutes': 5630,
'screen': 4952,
'shakespeareshakespeare': 1,
'losti': 7,
'appreciate': 1393,
'shakespeare': 429,
'mass': 417,
'goodis': 3,
'scottish': 164,
'certain': 1562,
```

```
'rev': 25,
'bowdler': 1,
'hence': 298,
'bowdlerization': 1,
'victorian': 117,
'erain': 3,
'improve': 492,
'perfectioni': 1,
'ten': 1566,
'text': 274,
'english': 1865,
'composition': 95,
'forte': 29,
'keep': 6497,
'cut': 2707,
'fantastic': 1439,
'prisoners': 228,
'famous': 1444,
'george': 1557,
'clooney': 109,
'roll': 1097,
'man': 10929,
'constant': 544,
'sorrow': 92,
'recommand': 2,
'everybody': 766,
'greet': 129,
'bart': 80,
'kind': 5427,
'draw': 1877,
'erotic': 337,
'amateurish': 402,
'unbelievable': 748,
'bits': 610,
'sort': 3248,
'school': 3397,
'project': 1244,
'rosanna': 34,
'arquette': 87,
'stock': 601,
'bizarre': 949,
'suppose': 3662,
'midwest': 28,
'town': 2270,
'involve': 3633,
```

```
'lessons': 295,
'learn': 2730,
'insights': 101,
'stilted': 178,
'quite': 7176,
'ridiculous': 1778,
'lot': 9547,
'skin': 443,
'intrigue': 1036,
'videotape': 103,
'nonsensewhat': 1,
'bisexual': 48,
'relationship': 1860,
'nowhere': 832,
'heterosexual': 51,
'absurd': 525,
'dance': 2487,
'stereotype': 844,
'pass': 1755,
'million': 704,
'miles': 488,
'could': 15115,
'starve': 95,
'aid': 444,
'africa': 359,
'simply': 3795,
'remake': 1480,
'fail': 2783,
'capture': 1666,
'flavor': 114,
'terror': 424,
'1963': 66,
'title': 3257,
'liam': 67,
'neeson': 43,
'excellent': 3950,
'always': 6231,
'hold': 2680,
'exception': 691,
'owen': 202,
'wilson': 353,
'luke': 318,
'major': 1871,
'fault': 641,
'version': 3970,
```

```
'stray': 113,
'shirley': 179,
'jackson': 535,
'attempt': 3782,
'grandiose': 40,
'thrill': 815,
'earlier': 1203,
'trade': 309,
'snazzier': 1,
'special': 4072,
'effect': 5539,
'friction': 26,
'older': 1205,
'top': 3209,
'horrible': 2433,
'wasnt': 4491,
'continuous': 74,
'minute': 1280,
'chance': 2269,
'development': 1141,
'busy': 269,
'run': 5479,
'sword': 278,
'emotional': 1322,
'attachment': 50,
'except': 2097,
'machine': 807,
'want': 13081,
'destroy': 1106,
'blatantly': 123,
'steal': 1832,
'lotr': 39,
'war': 4614,
'matrix': 290,
'examplesthe': 2,
'ghost': 1079,
'final': 2595,
'yoda': 31,
'obee': 1,
'vader': 77,
'spider': 100,
'begin': 6093,
'frodo': 27,
'attack': 1517,
'return': 2253,
```

```
'kings': 234,
'elijah': 68,
'wood': 562,
'victim': 730,
'waitit': 4,
'hypnotize': 59,
'sting': 86,
'wrap': 406,
'upuh': 1,
'helloand': 1,
'vs': 680,
'humans': 538,
'matrixor': 1,
'terminatorthere': 1,
'examples': 328,
'someone': 4305,
'nazis': 272,
'juvenile': 216,
'rush': 638,
'conclusion': 817,
'childrens': 388,
'adult': 891,
'either': 3368,
'disappointment': 707,
'save': 3339,
'remember': 3934,
'filmit': 127,
'cinema': 2635,
'dark': 2644,
'nervous': 178,
'7475': 1,
'dad': 768,
'brothersister': 12,
'newbury': 2,
'berkshire': 2,
'england': 483,
'tigers': 38,
'snow': 301,
'appearance': 823,
'grizzly': 26,
'adams': 221,
'actor': 4193,
'dan': 421,
'haggery': 1,
'anyone': 5085,
```

```
'dvd': 4513,
'etc': 2010,
'please': 2163,
'knowthe': 21,
'fitness': 13,
'club': 860,
'shame': 1293,
'nearest': 84,
'20': 1231,
'hear': 4156,
'others': 3138,
'stinkers': 31,
'nominate': 422,
'golden': 552,
'globe': 123,
'theyve': 465,
'female': 1811,
'renaissance': 82,
'painter': 109,
'mangle': 41,
'recognition': 209,
'complaint': 227,
'liberties': 89,
'facts': 424,
'perfectly': 1233,
'fine': 2464,
'bizarreby': 1,
'account': 520,
'artist': 534,
'dishwaterdull': 1,
'script': 6197,
'werent': 946,
'enough': 6696,
'naked': 826,
'factual': 101,
'hurriedly': 14,
'cap': 129,
'summary': 336,
'artists': 354,
'lifewe': 3,
'couple': 3751,
'hours': 1862,
'favor': 563,
'brevity': 22,
'sequels': 420,
```

```
         'surprise': 3871,
         '1990s': 161,
         'glut': 11,
         'amovies': 5,
         'cash': 470,
         'wrong': 3386,
         'guy': 8849,
         'concept': 954,
         'cliffhanger': 71,
         'mountain': 370,
         ...})
```

In [92]:
```
'''
We are creating a list that is sorted in the descending order based on the frequency of the words occurrance.
'''

frequency_of_words_sorted = sorted(counter, key=counter.get, reverse=True)
```

In [96]:
```
# creates a dictionary that maps integer IDs to each word in the vocabulary. The enumerate function assigns a unique in
converting_integers_and_words = dict(enumerate(frequency_of_words_sorted, 1))
```

In [98]:
```
# here, we add a padding token with ID 0 to the dictionary, which will be used later for padding sequences of different
converting_integers_and_words[0] = '<PAD>'
```

In [99]:
```
# this code will create a dictionary that maps each word in the vocabulary to its corresponding integer ID.
converting_integers_and_words = {word: id for id, word in converting_integers_and_words.items()}
```

In [101…
```
converting_integers_and_words
```

Out[101]:
```
{'film': 1,
 'movie': 2,
 'one': 3,
 'make': 4,
 'like': 5,
 'see': 6,
 'get': 7,
 'time': 8,
 'good': 9,
 'character': 10,
 'watch': 11,
 'go': 12,
 'even': 13,
 'would': 14,
 'think': 15,
 'really': 16,
 'story': 17,
 'show': 18,
 'look': 19,
 'say': 20,
 'much': 21,
 'well': 22,
 'know': 23,
 'end': 24,
 'people': 25,
 'great': 26,
 'bad': 27,
 'also': 28,
 'give': 29,
 'take': 30,
 'play': 31,
 'first': 32,
 'love': 33,
 'act': 34,
 'dont': 35,
 'come': 36,
 'find': 37,
 'movies': 38,
 'could': 39,
 'way': 40,
 'seem': 41,
 'many': 42,
 'want': 43,
 'work': 44,
 'plot': 45,
```

```
                 'two': 46,
                 'never': 47,
                 'little': 48,
                 'try': 49,
                 'best': 50,
                 'ever': 51,
                 'life': 52,
                 'better': 53,
                 'man': 54,
                 'still': 55,
                 'scene': 56,
                 'feel': 57,
                 'scenes': 58,
                 'part': 59,
                 'use': 60,
                 'something': 61,
                 'lot': 62,
                 'back': 63,
                 'interest': 64,
                 'real': 65,
                 'im': 66,
                 'guy': 67,
                 'doesnt': 68,
                 'thing': 69,
                 'didnt': 70,
                 'actors': 71,
                 'years': 72,
                 'leave': 73,
                 'actually': 74,
                 'cast': 75,
                 'funny': 76,
                 'though': 77,
                 'tell': 78,
                 'another': 79,
                 'live': 80,
                 'nothing': 81,
                 'new': 82,
                 'start': 83,
                 'star': 84,
                 'every': 85,
                 'old': 86,
                 'write': 87,
                 'point': 88,
                 'us': 89,
                 'become': 90,
```

```
 'cant': 91,
 'turn': 92,
 'director': 93,
 'quite': 94,
 'kill': 95,
 'pretty': 96,
 'things': 97,
 'set': 98,
 'thats': 99,
 'around': 100,
 'young': 101,
 'happen': 102,
 'fact': 103,
 'world': 104,
 'mean': 105,
 'enough': 106,
 'big': 107,
 'horror': 108,
 'may': 109,
 'keep': 110,
 'right': 111,
 'need': 112,
 'action': 113,
 'fan': 114,
 'long': 115,
 'ive': 116,
 'enjoy': 117,
 'however': 118,
 'bite': 119,
 'line': 120,
 'saw': 121,
 'without': 122,
 'isnt': 123,
 'always': 124,
 'script': 125,
 'must': 126,
 'original': 127,
 'put': 128,
 'music': 129,
 'almost': 130,
 'begin': 131,
 'whole': 132,
 'series': 133,
 'comedy': 134,
 'least': 135,
```

```
'kid': 136,
'last': 137,
'role': 138,
'believe': 139,
'lead': 140,
'shoot': 141,
'do': 142,
'laugh': 143,
'might': 144,
'theres': 145,
'far': 146,
'place': 147,
'minutes': 148,
'hes': 149,
'anything': 150,
'reason': 151,
'effect': 152,
'since': 153,
'probably': 154,
'run': 155,
'family': 156,
'kind': 157,
'name': 158,
'call': 159,
'performance': 160,
'book': 161,
'let': 162,
'help': 163,
'tv': 164,
'away': 165,
'rather': 166,
'worst': 167,
'read': 168,
'yet': 169,
'anyone': 170,
'sure': 171,
'fun': 172,
'girl': 173,
'screen': 174,
'expect': 175,
'woman': 176,
'course': 177,
'bore': 178,
'bring': 179,
'hard': 180,
```

```
'especially': 181,
'war': 182,
'move': 183,
'day': 184,
'job': 185,
'sense': 186,
'main': 187,
'worth': 188,
'different': 189,
'everything': 190,
'dvd': 191,
'although': 192,
'wasnt': 193,
'sound': 194,
'three': 195,
'recommend': 196,
'maybe': 197,
'lose': 198,
'understand': 199,
'someone': 200,
'true': 201,
'waste': 202,
'money': 203,
'together': 204,
'actor': 205,
'second': 206,
'hear': 207,
'follow': 208,
'view': 209,
'american': 210,
'fall': 211,
'everyone': 212,
'face': 213,
'instead': 214,
'talk': 215,
'10': 216,
'direct': 217,
'beautiful': 218,
'special': 219,
'miss': 220,
'later': 221,
'mind': 222,
'audience': 223,
'open': 224,
'house': 225,
```

```
'black': 226,
'version': 227,
'2': 228,
'short': 229,
'excellent': 230,
'fight': 231,
'remember': 232,
'idea': 233,
'john': 234,
'surprise': 235,
'change': 236,
'night': 237,
'include': 238,
'simply': 239,
'youre': 240,
'appear': 241,
'high': 242,
'attempt': 243,
'die': 244,
'head': 245,
'piece': 246,
'completely': 247,
'father': 248,
'couple': 249,
'nice': 250,
'else': 251,
'poor': 252,
'wife': 253,
'suppose': 254,
'meet': 255,
'eye': 256,
'death': 257,
'release': 258,
'involve': 259,
'picture': 260,
'home': 261,
'care': 262,
'along': 263,
'lack': 264,
'friends': 265,
'feature': 266,
'entertain': 267,
'rest': 268,
'add': 269,
'less': 270,
```

```
 'word': 271,
 'men': 272,
 'half': 273,
 'decide': 274,
 'hand': 275,
 'year': 276,
 'truly': 277,
 'school': 278,
 'classic': 279,
 'wrong': 280,
 'performances': 281,
 'either': 282,
 'production': 283,
 'save': 284,
 'dead': 285,
 'stupid': 286,
 'hollywood': 287,
 'next': 288,
 'title': 289,
 'murder': 290,
 'game': 291,
 'sort': 292,
 'full': 293,
 'women': 294,
 'wonder': 295,
 'review': 296,
 'top': 297,
 'create': 298,
 'camera': 299,
 'case': 300,
 'sex': 301,
 'budget': 302,
 'terrible': 303,
 'others': 304,
 'wonderful': 305,
 'awful': 306,
 'moments': 307,
 'sit': 308,
 'joke': 309,
 'mother': 310,
 'guess': 311,
 'perfect': 312,
 'video': 313,
 'base': 314,
 'human': 315,
```

```
'sequence': 316,
'flick': 317,
'small': 318,
'comment': 319,
'often': 320,
'couldnt': 321,
'perhaps': 322,
'style': 323,
'definitely': 324,
'episode': 325,
'early': 326,
'person': 327,
'mention': 328,
'consider': 329,
'experience': 330,
'rat': 331,
'absolutely': 332,
'dialogue': 333,
'force': 334,
'certainly': 335,
'light': 336,
'amaze': 337,
'entire': 338,
'ask': 339,
'felt': 340,
'buy': 341,
'hope': 342,
'stop': 343,
'spend': 344,
'deal': 345,
'several': 346,
'stand': 347,
'fail': 348,
'age': 349,
'matter': 350,
'worse': 351,
'finally': 352,
'side': 353,
'learn': 354,
'break': 355,
'problem': 356,
'cut': 357,
'boy': 358,
'totally': 359,
'friend': 360,
```

```
'disappoint': 361,
'shes': 362,
'hold': 363,
'close': 364,
'mr': 365,
'dark': 366,
'cinema': 367,
'wont': 368,
'hit': 369,
'wait': 370,
'humor': 371,
'already': 372,
'yes': 373,
'final': 374,
'low': 375,
'manage': 376,
'white': 377,
'forget': 378,
'able': 379,
'evil': 380,
'walk': 381,
'example': 382,
'children': 383,
'youll': 384,
'support': 385,
'direction': 386,
'id': 387,
'type': 388,
'throughout': 389,
'voice': 390,
'speak': 391,
'throw': 392,
'portray': 393,
'wish': 394,
'drama': 395,
'number': 396,
'3': 397,
'dance': 398,
'girls': 399,
'days': 400,
'fine': 401,
'rent': 402,
'group': 403,
'despite': 404,
'quality': 405,
```

```
'horrible': 406,
'pay': 407,
'twist': 408,
'credit': 409,
'oh': 410,
'present': 411,
'theyre': 412,
'stay': 413,
'history': 414,
'realize': 415,
'stuff': 416,
'deserve': 417,
'killer': 418,
'touch': 419,
'power': 420,
'score': 421,
'question': 422,
'unfortunately': 423,
'theme': 424,
'favorite': 425,
'past': 426,
'son': 427,
'michael': 428,
'behind': 429,
'catch': 430,
'car': 431,
'town': 432,
'chance': 433,
'brilliant': 434,
'ones': 435,
'return': 436,
'order': 437,
'obviously': 438,
'body': 439,
'viewer': 440,
'sometimes': 441,
'annoy': 442,
'soon': 443,
'complete': 444,
'decent': 445,
'build': 446,
'overall': 447,
'actress': 448,
'late': 449,
'art': 450,
```

```
 'heart': 451,
 'city': 452,
 'level': 453,
 'please': 454,
 'figure': 455,
 'genre': 456,
 'grow': 457,
 'highly': 458,
 'hell': 459,
 'extremely': 460,
 'child': 461,
 'except': 462,
 'pick': 463,
 'wouldnt': 464,
 'blood': 465,
 'ill': 466,
 'police': 467,
 '1': 468,
 'edit': 469,
 'jam': 470,
 'hour': 471,
 'moment': 472,
 'drive': 473,
 'cause': 474,
 'career': 475,
 'stories': 476,
 'strong': 477,
 'offer': 478,
 'deliver': 479,
 'etc': 480,
 'slow': 481,
 'daughter': 482,
 'provide': 483,
 'pace': 484,
 'serious': 485,
 'particularly': 486,
 'husband': 487,
 'state': 488,
 'allow': 489,
 'obvious': 490,
 'roles': 491,
 'explain': 492,
 'hilarious': 493,
 'cool': 494,
 'result': 495,
```

```
'god': 496,
'across': 497,
'violence': 498,
'simple': 499,
'exactly': 500,
'sing': 501,
'dream': 502,
'compare': 503,
'song': 504,
'cop': 505,
'usually': 506,
'huge': 507,
'robert': 508,
'value': 509,
'ago': 510,
'david': 511,
'room': 512,
'crap': 513,
'whose': 514,
'reality': 515,
'draw': 516,
'focus': 517,
'stick': 518,
'major': 519,
'english': 520,
'convince': 521,
'hours': 522,
'relationship': 523,
'sad': 524,
'shock': 525,
'ok': 526,
'alone': 527,
'dog': 528,
'cinematography': 529,
'check': 530,
'team': 531,
'pull': 532,
'shots': 533,
'subject': 534,
'possible': 535,
'steal': 536,
'none': 537,
'gore': 538,
'talent': 539,
'somewhat': 540,
```

```
'seriously': 541,
'female': 542,
'cover': 543,
'today': 544,
'plan': 545,
'brother': 546,
'note': 547,
'beyond': 548,
'usual': 549,
'important': 550,
'message': 551,
'prove': 552,
'mostly': 553,
'hero': 554,
'ridiculous': 555,
'documentary': 556,
'season': 557,
'form': 558,
'opinion': 559,
'rock': 560,
'single': 561,
'jack': 562,
'pass': 563,
'strange': 564,
'due': 565,
'silly': 566,
'parent': 567,
'detail': 568,
'escape': 569,
'remain': 570,
'novel': 571,
'local': 572,
'upon': 573,
'image': 574,
'attention': 575,
'scary': 576,
'win': 577,
'clearly': 578,
'apparently': 579,
'happy': 580,
'5': 581,
'imagine': 582,
'arent': 583,
'directors': 584,
'produce': 585,
```

'episodes': 586,
'charm': 587,
'musical': 588,
'four': 589,
'train': 590,
'country': 591,
'discover': 592,
'basically': 593,
'problems': 594,
'anyway': 595,
'fit': 596,
'cheap': 597,
'hop': 598,
'doubt': 599,
'avoid': 600,
'television': 601,
'british': 602,
'fill': 603,
'events': 604,
'marry': 605,
'capture': 606,
'modern': 607,
'wear': 608,
'remind': 609,
'4': 610,
'havent': 611,
'space': 612,
'whats': 613,
'french': 614,
'drug': 615,
'easily': 616,
'class': 617,
'songs': 618,
'whether': 619,
'carry': 620,
'clear': 621,
'similar': 622,
'send': 623,
'earth': 624,
'gun': 625,
'thank': 626,
'dialog': 627,
'bunch': 628,
'fire': 629,
'predictable': 630,

```
'develop': 631,
'air': 632,
'viewers': 633,
'date': 634,
'within': 635,
'romantic': 636,
'battle': 637,
'soundtrack': 638,
'copy': 639,
'ten': 640,
'hate': 641,
'certain': 642,
'confuse': 643,
'middle': 644,
'george': 645,
'chase': 646,
'future': 647,
'agree': 648,
'enjoyable': 649,
'five': 650,
'soldier': 651,
'comic': 652,
'mark': 653,
'entertainment': 654,
'near': 655,
'among': 656,
'storyline': 657,
'choose': 658,
'doctor': 659,
'attack': 660,
'easy': 661,
'sequel': 662,
'theater': 663,
'typical': 664,
'dull': 665,
'greatest': 666,
'ways': 667,
'king': 668,
'box': 669,
'actual': 670,
'remake': 671,
'reveal': 672,
'thriller': 673,
'nearly': 674,
'effort': 675,
```

```
'general': 676,
'elements': 677,
'continue': 678,
'tale': 679,
'monster': 680,
'sorry': 681,
'notice': 682,
'trouble': 683,
'contain': 684,
'writer': 685,
'suck': 686,
'finish': 687,
'cartoon': 688,
'describe': 689,
'famous': 690,
'bear': 691,
'fantastic': 692,
'80s': 693,
'issue': 694,
'lady': 695,
'dr': 696,
'youve': 697,
'suffer': 698,
'suspense': 699,
'boys': 700,
'bill': 701,
'water': 702,
'cry': 703,
'realistic': 704,
'hide': 705,
'straight': 706,
'mess': 707,
'sister': 708,
'peter': 709,
'match': 710,
'premise': 711,
'richard': 712,
'mystery': 713,
'somehow': 714,
'beat': 715,
'material': 716,
'list': 717,
'fly': 718,
'appreciate': 719,
'excite': 720,
```

```
 'bother': 721,
 'eat': 722,
 'mix': 723,
 'romance': 724,
 'blow': 725,
 'admit': 726,
 'whos': 727,
 'alien': 728,
 'particular': 729,
 'animation': 730,
 'believable': 731,
 'weak': 732,
 'background': 733,
 'period': 734,
 'eventually': 735,
 'treat': 736,
 'difficult': 737,
 'deep': 738,
 'york': 739,
 'crime': 740,
 'party': 741,
 'appeal': 742,
 'red': 743,
 'crew': 744,
 'atmosphere': 745,
 'average': 746,
 'fast': 747,
 'paul': 748,
 'emotional': 749,
 'lie': 750,
 'baby': 751,
 'possibly': 752,
 'poorly': 753,
 'hat': 754,
 'okay': 755,
 'situation': 756,
 'truth': 757,
 'struggle': 758,
 'lame': 759,
 'fear': 760,
 'suggest': 761,
 'tom': 762,
 'shame': 763,
 'indeed': 764,
 'drink': 765,
```

```
'footage': 766,
'warn': 767,
'sexual': 768,
'free': 769,
'minute': 770,
'memorable': 771,
'mistake': 772,
'oscar': 773,
'previous': 774,
'accent': 775,
'filmmakers': 776,
'whatever': 777,
'scifi': 778,
'spirit': 779,
'writers': 780,
'personal': 781,
'forward': 782,
'cheesy': 783,
'flaw': 784,
'total': 785,
'beauty': 786,
'gay': 787,
'third': 788,
'hot': 789,
'control': 790,
'imdb': 791,
'project': 792,
'towards': 793,
'scream': 794,
'costume': 795,
'inside': 796,
'western': 797,
'nature': 798,
'incredibly': 799,
'unless': 800,
'superb': 801,
'perfectly': 802,
'male': 803,
'otherwise': 804,
'de': 805,
'20': 806,
'amount': 807,
'award': 808,
'society': 809,
'weird': 810,
```

```
 'master': 811,
 'japanese': 812,
 'screenplay': 813,
 'quickly': 814,
 'street': 815,
 'island': 816,
 'adventure': 817,
 'front': 818,
 'various': 819,
 'unique': 820,
 'older': 821,
 'masterpiece': 822,
 'land': 823,
 'earlier': 824,
 'plus': 825,
 'badly': 826,
 'crazy': 827,
 'stage': 828,
 'plenty': 829,
 'respect': 830,
 'company': 831,
 'term': 832,
 'jump': 833,
 'serve': 834,
 'america': 835,
 'answer': 836,
 'track': 837,
 'fairly': 838,
 'rise': 839,
 'promise': 840,
 'powerful': 841,
 'lee': 842,
 'fantasy': 843,
 'color': 844,
 'dramatic': 845,
 'band': 846,
 'creepy': 847,
 'business': 848,
 'tear': 849,
 'b': 850,
 'rich': 851,
 'accept': 852,
 'disturb': 853,
 'amuse': 854,
 'plain': 855,
```

```
'development': 856,
'outside': 857,
'joe': 858,
'political': 859,
'concern': 860,
'dumb': 861,
'dress': 862,
'perform': 863,
'sell': 864,
'store': 865,
'brain': 866,
'channel': 867,
'cat': 868,
'hardly': 869,
'ideas': 870,
'destroy': 871,
'70s': 872,
'recently': 873,
'share': 874,
'la': 875,
'record': 876,
'success': 877,
'listen': 878,
'large': 879,
'roll': 880,
'apart': 881,
'secret': 882,
'girlfriend': 883,
'sleep': 884,
'scar': 885,
'exist': 886,
'design': 887,
'members': 888,
'william': 889,
'clever': 890,
'brothers': 891,
'talented': 892,
'ghost': 893,
'cute': 894,
'introduce': 895,
'pure': 896,
'claim': 897,
'kick': 898,
'german': 899,
'travel': 900,
```

```
 'reach': 901,
 'camp': 902,
 'odd': 903,
 'blue': 904,
 'park': 905,
 'search': 906,
 'potential': 907,
 'disney': 908,
 'visual': 909,
 'slightly': 910,
 'zombie': 911,
 'drag': 912,
 'van': 913,
 'italian': 914,
 'incredible': 915,
 'public': 916,
 'ruin': 917,
 'intrigue': 918,
 'inspire': 919,
 'familiar': 920,
 'hole': 921,
 'fake': 922,
 'unlike': 923,
 'villain': 924,
 'burn': 925,
 'taste': 926,
 'entirely': 927,
 'approach': 928,
 'culture': 929,
 'nudity': 930,
 'popular': 931,
 'judge': 932,
 'spot': 933,
 'tone': 934,
 'step': 935,
 'engage': 936,
 'race': 937,
 'gang': 938,
 'ring': 939,
 'younger': 940,
 'biggest': 941,
 'receive': 942,
 'neither': 943,
 'suddenly': 944,
 'purpose': 945,
```

```
'former': 946,
'hang': 947,
'portrayal': 948,
'rate': 949,
'era': 950,
'sadly': 951,
'common': 952,
'count': 953,
'office': 954,
'trip': 955,
'tension': 956,
'producers': 957,
'intelligent': 958,
'survive': 959,
'flat': 960,
'relate': 961,
'violent': 962,
'trash': 963,
'social': 964,
'audiences': 965,
'hurt': 966,
'pop': 967,
'sweet': 968,
'torture': 969,
'science': 970,
'suspect': 971,
'ship': 972,
'strike': 973,
'college': 974,
'recent': 975,
'choice': 976,
'successful': 977,
'wind': 978,
'language': 979,
'suit': 980,
'intend': 981,
'impress': 982,
'paint': 983,
'cold': 984,
'handle': 985,
'concept': 986,
'mad': 987,
'longer': 988,
'positive': 989,
'bizarre': 990,
```

```
 'bond': 991,
 'basic': 992,
 'century': 993,
 'christmas': 994,
 'raise': 995,
 'situations': 996,
 'werent': 997,
 'haunt': 998,
 'hair': 999,
 'drop': 1000,
 ...}
```

Now, we will proceed to encode the reviews. we will do so by convert each token into different corresponding index in the vocabulary array.

In [104…

```
# Let us encode each word
encoded_reviews = [[converting_integers_and_words[word] for word in individual_review.split()] for individual_review in
```

```
100%|██████████| 50000/50000 [00:02<00:00, 24376.75it/s]
```

In [109…

```
unique_lengths = set([len(i) for i in encoded_reviews])
```

In [110…

```
unique_lengths
```

Out[110]:
```
{3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
 32,
 33,
 34,
 35,
 36,
 37,
 38,
 39,
 40,
 41,
 42,
 43,
 44,
 45,
 46,
 47,
```

```
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
```

```
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
105,
106,
107,
108,
109,
110,
111,
112,
113,
114,
115,
116,
117,
118,
119,
120,
121,
122,
123,
124,
125,
126,
127,
128,
129,
130,
131,
132,
133,
134,
135,
136,
137,
```

```
138,
139,
140,
141,
142,
143,
144,
145,
146,
147,
148,
149,
150,
151,
152,
153,
154,
155,
156,
157,
158,
159,
160,
161,
162,
163,
164,
165,
166,
167,
168,
169,
170,
171,
172,
173,
174,
175,
176,
177,
178,
179,
180,
181,
182,
```

```
183,
184,
185,
186,
187,
188,
189,
190,
191,
192,
193,
194,
195,
196,
197,
198,
199,
200,
201,
202,
203,
204,
205,
206,
207,
208,
209,
210,
211,
212,
213,
214,
215,
216,
217,
218,
219,
220,
221,
222,
223,
224,
225,
226,
227,
```

```
228,
229,
230,
231,
232,
233,
234,
235,
236,
237,
238,
239,
240,
241,
242,
243,
244,
245,
246,
247,
248,
249,
250,
251,
252,
253,
254,
255,
256,
257,
258,
259,
260,
261,
262,
263,
264,
265,
266,
267,
268,
269,
270,
271,
272,
```

```
273,
274,
275,
276,
277,
278,
279,
280,
281,
282,
283,
284,
285,
286,
287,
288,
289,
290,
291,
292,
293,
294,
295,
296,
297,
298,
299,
300,
301,
302,
303,
304,
305,
306,
307,
308,
309,
310,
311,
312,
313,
314,
315,
316,
317,
```

```
318,
319,
320,
321,
322,
323,
324,
325,
326,
327,
328,
329,
330,
331,
332,
333,
334,
335,
336,
337,
338,
339,
340,
341,
342,
343,
344,
345,
346,
347,
348,
349,
350,
351,
352,
353,
354,
355,
356,
357,
358,
359,
360,
361,
362,
```

```
363,
364,
365,
366,
367,
368,
369,
370,
371,
372,
373,
374,
375,
376,
377,
378,
379,
380,
381,
382,
383,
384,
385,
386,
387,
388,
389,
390,
391,
392,
393,
394,
395,
396,
397,
398,
399,
400,
401,
402,
403,
404,
405,
406,
407,
```

```
408,
409,
410,
411,
412,
413,
414,
415,
416,
417,
418,
419,
420,
421,
422,
423,
424,
425,
426,
427,
428,
429,
430,
431,
432,
433,
434,
435,
436,
437,
438,
439,
440,
441,
442,
443,
444,
445,
446,
447,
448,
449,
450,
451,
452,
```

```
453,
454,
455,
456,
457,
458,
459,
460,
461,
462,
463,
464,
465,
466,
467,
468,
469,
470,
471,
472,
473,
474,
475,
476,
477,
478,
479,
480,
481,
482,
483,
484,
485,
486,
487,
488,
489,
490,
491,
492,
493,
494,
495,
496,
497,
```

```
498,
499,
500,
501,
502,
503,
504,
505,
506,
507,
508,
509,
510,
511,
512,
513,
514,
515,
516,
517,
518,
519,
520,
521,
522,
523,
524,
525,
526,
527,
528,
529,
530,
531,
532,
533,
534,
535,
536,
537,
538,
539,
540,
541,
542,
```

```
543,
544,
545,
546,
547,
548,
549,
550,
551,
553,
555,
556,
557,
558,
559,
564,
566,
567,
568,
569,
571,
572,
573,
574,
575,
576,
579,
583,
584,
590,
600,
601,
603,
611,
612,
615,
618,
623,
633,
635,
644,
665,
688,
696,
712,
```

```
 736,
 769,
 787,
 807,
 810,
 891,
 913,
 917,
 927,
 1087,
 1123,
 1163,
 1429}
```

This further proves that we need to ensure all sequences are of equal length, which we will do later

In [112…

```
# lets print the first 10 words of the first 10 reviews we have in our data
for i in range(0, 10):
    print(encoded_reviews[i][:10])
```

```
[3, 1692, 328, 11, 468, 2953, 325, 384, 1504, 111]
[305, 48, 283, 1, 2608, 14832, 76541, 1097, 29, 3107]
[15, 305, 40, 344, 8, 789, 1254, 2079, 308, 632]
[593, 145, 156, 48, 358, 3108, 15, 145, 911, 43648]
[76548, 28362, 33, 8, 203, 1873, 1014, 1, 11, 365]
[154, 3211, 425, 2, 17, 37691, 2099, 6581, 3109, 474]
[171, 14, 5, 6, 7475, 634, 53488, 133, 6477, 544]
[18, 337, 76559, 233, 872, 32, 632, 32, 1091, 1098]
[2271, 989, 319, 1, 19, 782, 11, 1, 27, 772]
[5, 127, 2266, 6116, 1919, 5, 2, 101, 86, 33]
```

Here, we have sucessfully converted the text in each row into individual tokens based on index positions in our vocabulary. However, our tokens are of different lengths. Meaning that not all reviews have the same length. Some might be too large whereas some will be very smaller. Therefore, we need to predefine the maximum possible length of any given sequence. If a particular review is shorter than the predefined length, we will proceed to pad the sequence else we will trim the sequence or review respetively

In [113…

```
# padding sequences
def paddind_sequences(cleaned_text, padding_value, max_length_of_sequence=256):
    '''
    The code below will create a 2D numpy array called new_feature with dimensions (len(reviews), seq_length), filled w
    Therefore, we can use this 2d array to replace the corresponidng sequences at respective indexes whereas unused ele
    '''
```

```
    new_feature = np.full((len(cleaned_text), max_length_of_sequence), padding_value, dtype=int)
    '''
    pads each sequence with the padding ID pad_id up to the desired seq_length. The code first converts the current seq
      numpy array using np.array(row), then takes the first seq_length elements (if the sequence is longer than seq_len
      this to the corresponding row in features. This way, sequences shorter
      than seq_length are padded with the pad_id at the end of the sequence, and longer sequences are truncated to seq_
    '''
    for i, row in enumerate(cleaned_text):
        new_feature[i, :len(row)] = np.array(row)[:max_length_of_sequence]
    return new_feature
```

In [116…

```
max_length_of_sequence = 256

# returns the final 2D numpy array where each row represents a padded sequence.
features = paddind_sequences(encoded_reviews, padding_value=converting_integers_and_words['<PAD>'], max_length_of_seque
```

In this case, the maximum sequence length of 256 was likely chosen based on some analysis of the distribution of sequence lengths in the input data, as well as the available hardware resources. A sequence length of 256 is relatively long and should be sufficient to capture most of the information in the input data, while still being computationally tractable. The reason we choose to make the maximum length of a sequence to be 256 is based on the maximum length of a review in our dataset. We also had to consider the limitations of our hardware becuase longer sequencecs could contain more information, but also might be more computationally expensive. Whereas, shorter sequences might be less computationally expensive but might contain very less information. So, choosing the maximum length of a sequence depends upon a variety of factors respectively.

In [117…

```
# get labels as numpy
targets = df.label.to_numpy()
targets
```

Out[117]:

```
array([1, 1, 1, ..., 0, 0, 0], dtype=int64)
```

Train Test Split

In [118…

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

In [121…

```
# Split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(features, targets, test_size=0.3, random_state=42)
```

In [122...
```
# Split the remaining data into validation and test sets
X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.5, random_state=42)
```

In [123...
```
print("The shapes for Training set: ", X_train.shape, y_train.shape)
```

```
The shapes for Training set:  (35000, 256) (35000,)
```

In [124...
```
print("The shapes for Validation set: ", X_val.shape, y_val.shape)
```

```
The shapes for Validation set:  (7500, 256) (7500,)
```

In [125...
```
print("The shapes for Test set: ", X_test.shape, y_test.shape)
```

```
The shapes for Test set:  (7500, 256) (7500,)
```

We have successfully divided our data into train, test, and valdiation set. Now, we will proceed to convert it into pytorch dataloader with a certain batch size for performing minibatch gradient descent.

In [126...
```
from torch.utils.data import TensorDataset, DataLoader
```

In [127...
```
batch_size = 512
```

In [128...
```
# Lets create tensordatasetss
training_set = TensorDataset(torch.from_numpy(X_train), torch.from_numpy(y_train))
validation_set = TensorDataset(torch.from_numpy(X_val), torch.from_numpy(y_val))
testing_set = TensorDataset(torch.from_numpy(X_test), torch.from_numpy(y_test))
```

In [129...
```
# lets proceed to create pytorch dataloaders next.
train_dl = DataLoader(training_set, shuffle=True, batch_size=batch_size)
val_dl = DataLoader(validation_set, shuffle=True, batch_size=batch_size)
test_dl = DataLoader(testing_set, shuffle=True, batch_size=batch_size)
```

Building LSTM

In [130...
```
import torch.nn.functional as F
```

In [135...
```
class LSTM(nn.Module):
    def __init__(self, vocab_size, output_size, hidden_size=128, embedding_size=400, n_layers=2, dropout=0.2):
        super(LSTM, self).__init__()
```

```
        # Define an embedding layer that maps each token to a dense vector of embedding_size
        self.embedding_layer = nn.Embedding(vocab_size, embedding_size)

        # Define an LSTM layer with hidden_size hidden units, n_layers layers, and a dropout rate of dropout
        self.lstm_layer = nn.LSTM(embedding_size, hidden_size, n_layers, dropout=dropout, batch_first=True)

        # Define a dropout layer with dropout probability of dropout
        self.dropout_layer = nn.Dropout(p=dropout)

        # Define a linear layer that maps the output of the LSTM to the output_size
        self.fully_connected_laeyer = nn.Linear(hidden_size, output_size)

        # Define a sigmoid activation function
        self.sigmoid_layer = nn.Sigmoid()

    def forward(self, input_seq):

        # Convert the input to a LongTensor
        input_seq = input_seq.long()

        # Embed the input sequence to a sequence of dense vectors of embedding_size
        input_seq = self.embedding_layer(input_seq)

        # Feed the embedded sequence through the LSTM layer
        output, _ = self.lstm_layer(input_seq)

        # Select only the last output of the LSTM as the final output
        output = output[:, -1, :]

        # Apply dropout to the output
        output = self.dropout_layer(output)

        # Feed the output through the linear layer to get the logits
        output = self.fully_connected_laeyer(output)

        # Apply sigmoid activation to get the final output probabilities
        output = self.sigmoid_layer(output)

        return output
```

We have created the LSTM neural network, which is a specific type of RNN developed to solve the problem of vanishing and exploding gradients. Here, we have added the embedding layer to reduve the dimensionality of the vocabulary by learning its representation. LSTM will be our main layer as a RNN. The dropout layer will perform regularization for preventing overfitting. Finally, a fully connected layer is used to classify between positive and negative sentiments respectively.

In [131…

```
# define training device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

In [138…

```
len(converting_integers_and_words)
```

Out[138]:

```
234645
```

In [133…

```
# model hyperparamters
vocab_size = len(converting_integers_and_words)
output_size = 1
embedding_size = 256
hidden_size = 512
n_layers = 2
dropout=0.2
weight_decay=None
```

In [136…

```
# model initialization
model = LSTM(vocab_size, output_size, hidden_size, embedding_size, n_layers, dropout)
model = model.to(device)
print(model)
```

```
LSTM(
  (embedding_layer): Embedding(234645, 256)
  (lstm_layer): LSTM(256, 512, num_layers=2, batch_first=True, dropout=0.2)
  (dropout_layer): Dropout(p=0.2, inplace=False)
  (fully_connected_laeyer): Linear(in_features=512, out_features=1, bias=True)
  (sigmoid_layer): Sigmoid()
)
```

Due to some issues with PyTorch, we have instead created model summary for equivalent LSTM model in Keras and Tensorflow.

ModelSummary

This is the model summary of the LSTM model.

Model Training

In [ ]:
```
'''
The learning rate of 0.001 was selected with binary cross entropy as loss function. Additionally,adam optimizer was cho
'''
lr = 0.001
criterion = nn.BCELoss()
optim = torch.optim.Adam(model.parameters(), lr=lr)
grad_clip = 5
epochs = 20
```

In [ ]:
```
device
```

Out[ ]:
```
device(type='cuda')
```

In [ ]:
```
sigmoid_activation = nn.Sigmoid() # activation function for binary classification
```

In [ ]:
```
# train loop
train_losses = []
train_accs = []
val_losses = []
val_accs = []

best_val_loss = float('inf')
patience = 7
early_stopping_counter = 0

for epoch in range(epochs):

    model.train()

    train_loss = 0
    train_acc = 0

    for feature, target in tqdm(train_dl):
        # move to device
        feature, target = feature.to(device), target.to(device)

        # reset optimizer
        optim.zero_grad()

        # forward pass
        out = model(feature)
```

```
            # acc
            predicted = torch.tensor([1 if i == True else 0 for i in out > 0.5], device=device)
            equals = predicted == target
            acc = torch.mean(equals.type(torch.FloatTensor))
            train_acc += acc.item()

            # loss
            loss = criterion(out.squeeze(), target.float())
            train_loss += loss.item()
            loss.backward()

            # clip grad
            nn.utils.clip_grad_norm_(model.parameters(), grad_clip)

            # update optimizer
            optim.step()

            # free some memory
            del feature, target, predicted

        train_loss = train_loss / len(train_dl)
        train_acc = train_acc / len(train_dl)
        train_losses.append(train_loss)
        train_accs.append(train_acc)


        model.eval()

        val_loss = 0
        val_acc = 0

        with torch.no_grad():
            for feature, target in val_dl:
                # move to device
                feature, target = feature.to(device), target.to(device)

                # forward pass
                out = model(feature)

                # acc
                predicted = torch.tensor([1 if i == True else 0 for i in out > 0.5], device=device)
                equals = predicted == target
                acc = torch.mean(equals.type(torch.FloatTensor))
                val_acc += acc.item()
```

```
            # loss
            loss = criterion(out.squeeze(), target.float())
            val_loss += loss.item()

            # free some memory
            del feature, target, predicted

    val_loss = val_loss / len(val_dl)
    val_acc = val_acc / len(val_dl)
    val_losses.append(val_loss)
    val_accs.append(val_acc)

    print(f"Epoch {epoch+1}:")
    print(f"Training Loss: {train_loss:.4f} | Training Accuracy: {train_acc*100:.4f}%")
    print(f"Validation Loss: {val_loss:.4f} | Validation Accuracy: {val_acc*100:.4f}%")

    ''' IMPLEMENTEING CUSTOM EARLY STOPPING '''
    if val_loss < best_val_loss:
        early_stopping_counter = 0
        best_val_loss = val_loss
    else:
        early_stopping_counter += 1
        if early_stopping_counter >= patience:
            print(f"Early stopping triggered after {patience} epochs without improvement.")
            break
```

```
100%|██████████| 69/69 [01:01<00:00,  1.12it/s]
Epoch 1:
Training Loss: 0.6947 | Training Accuracy: 50.2833%
Validation Loss: 0.6941 | Validation Accuracy: 49.1563%
100%|██████████| 69/69 [01:04<00:00,  1.07it/s]
Epoch 2:
Training Loss: 0.6905 | Training Accuracy: 51.3345%
Validation Loss: 0.6941 | Validation Accuracy: 49.3982%
100%|██████████| 69/69 [01:06<00:00,  1.04it/s]
Epoch 3:
Training Loss: 0.6797 | Training Accuracy: 52.5180%
Validation Loss: 0.6994 | Validation Accuracy: 49.7857%
100%|██████████| 69/69 [01:05<00:00,  1.05it/s]
Epoch 4:
Training Loss: 0.6654 | Training Accuracy: 53.2681%
Validation Loss: 0.7121 | Validation Accuracy: 49.7336%
100%|██████████| 69/69 [01:06<00:00,  1.04it/s]
```

```
Epoch 5:
Training Loss: 0.6465 | Training Accuracy: 54.2673%
Validation Loss: 0.7425 | Validation Accuracy: 51.8046%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 6:
Training Loss: 0.6624 | Training Accuracy: 57.1383%
Validation Loss: 0.6894 | Validation Accuracy: 65.7034%
100%|██████████| 69/69 [01:06<00:00,  1.03it/s]
Epoch 7:
Training Loss: 0.6454 | Training Accuracy: 56.1019%
Validation Loss: 0.7343 | Validation Accuracy: 51.4367%
100%|██████████| 69/69 [01:07<00:00,  1.03it/s]
Epoch 8:
Training Loss: 0.6400 | Training Accuracy: 54.4198%
Validation Loss: 0.7651 | Validation Accuracy: 49.6484%
100%|██████████| 69/69 [01:07<00:00,  1.03it/s]
Epoch 9:
Training Loss: 0.6399 | Training Accuracy: 54.4997%
Validation Loss: 0.7723 | Validation Accuracy: 50.1291%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 10:
Training Loss: 0.6291 | Training Accuracy: 58.2234%
Validation Loss: 0.7395 | Validation Accuracy: 52.5152%
100%|██████████| 69/69 [01:07<00:00,  1.03it/s]
Epoch 11:
Training Loss: 0.5709 | Training Accuracy: 71.6408%
Validation Loss: 0.6038 | Validation Accuracy: 73.6182%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 12:
Training Loss: 0.5196 | Training Accuracy: 75.0151%
Validation Loss: 0.5388 | Validation Accuracy: 76.9028%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 13:
Training Loss: 0.4362 | Training Accuracy: 80.5441%
Validation Loss: 0.5230 | Validation Accuracy: 78.4741%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 14:
Training Loss: 0.3738 | Training Accuracy: 84.6499%
Validation Loss: 0.4600 | Validation Accuracy: 82.0021%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
```

```
Epoch 15:
Training Loss: 0.2825 | Training Accuracy: 89.2924%
Validation Loss: 0.4388 | Validation Accuracy: 83.3573%
100%|██████████| 69/69 [01:08<00:00,  1.01it/s]
Epoch 16:
Training Loss: 0.2464 | Training Accuracy: 90.8059%
Validation Loss: 0.4490 | Validation Accuracy: 83.5272%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 17:
Training Loss: 0.2002 | Training Accuracy: 92.9719%
Validation Loss: 0.4467 | Validation Accuracy: 84.1284%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 18:
Training Loss: 0.1562 | Training Accuracy: 94.7661%
Validation Loss: 0.4746 | Validation Accuracy: 84.2988%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 19:
Training Loss: 0.1336 | Training Accuracy: 95.5970%
Validation Loss: 0.4704 | Validation Accuracy: 84.6258%
100%|██████████| 69/69 [01:07<00:00,  1.02it/s]
Epoch 20:
Training Loss: 0.1042 | Training Accuracy: 96.7434%
Validation Loss: 0.5149 | Validation Accuracy: 84.5060%
```

In [83]:
```
# val_accs = [49.1563, 49.3982, 49.7857, 49.7336, 51.8046, 65.7034, 51.4367, 49.6484, 50.1291, 52.5152,73.6182, 76.9028
# val_losses = [0.6941 , 0.6941 , 0.6994 ,0.6654, 0.7425 ,0.6894 , 0.7343 , 0.7651 ,0.7723 ,0.7395 ,0.6038 , 0.5388 ,0.
```

In [ ]:
```
torch.save(model.state_dict(), 'LSTM_20Epochs.pth')
!cp LSTM_20Epochs.pth /content/drive/MyDrive/AI_Coursework_Portfolio_Dataset
```

In [89]:
```
training_stats = pd.DataFrame({"Training_Accuracy": train_accs,
                               "Training_Loss": train_losses,
                               "Val_Loss": val_losses,
                               "Val_Accuracy": val_accs})

training_stats.to_csv("LSTM_Training Statistics.csv", index = True)
```

In [90]:
```
training_stats.head()
```

Out[90]:

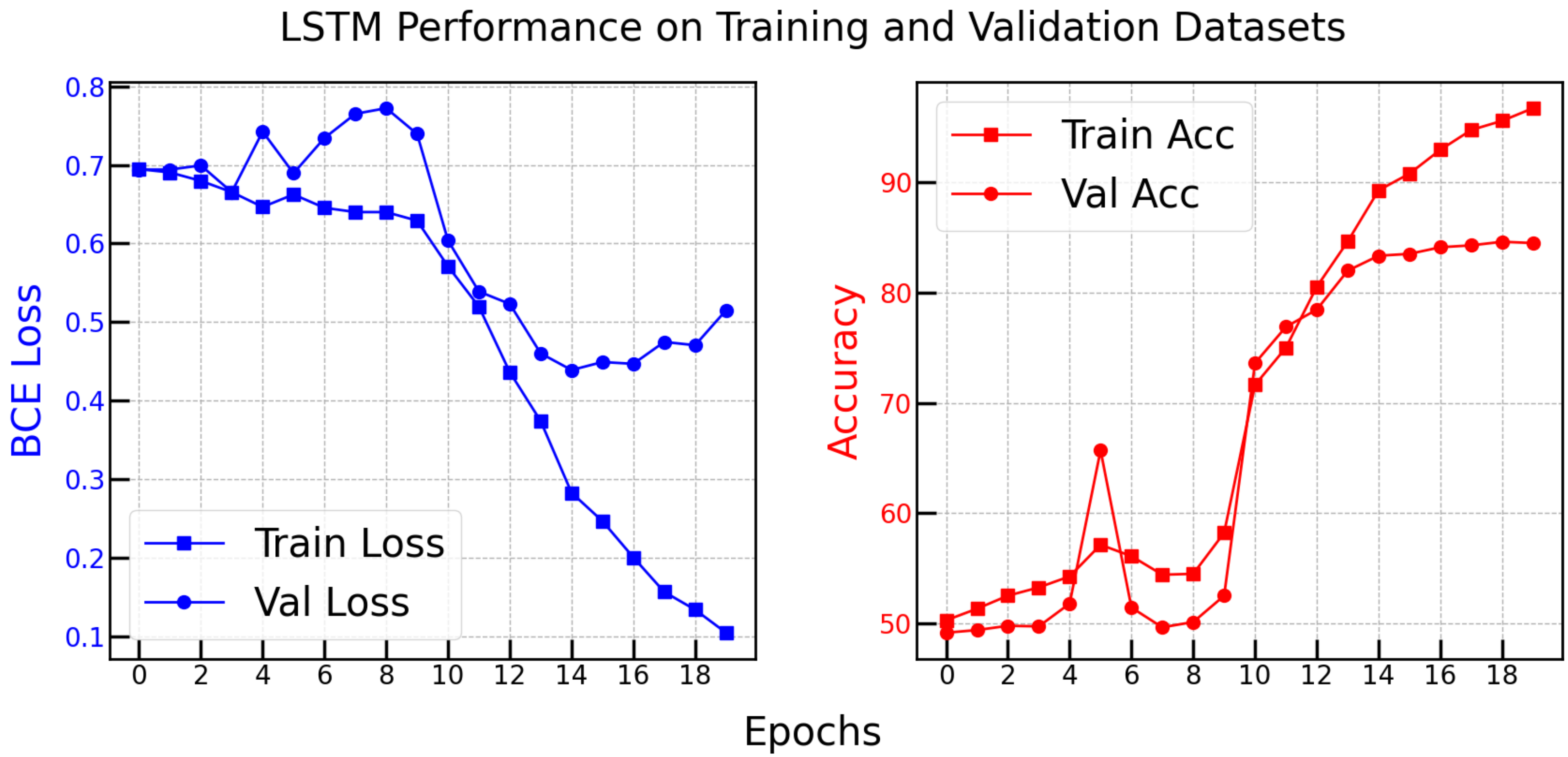
	Training_Accuracy	Training_Loss	Val_Loss	Val_Accuracy
0	50.283308	0.694711	0.6941	49.1563
1	51.334451	0.690464	0.6941	49.3982
2	52.518017	0.679706	0.6994	49.7857
3	53.268131	0.665419	0.6654	49.7336
4	54.267338	0.646485	0.7425	51.8046

In [91]:

```
fig, axes = plt.subplots(1, 2, figsize=(20, 8))

ax1 = plt.subplot(1,2, 1)

''' Left plot contains the validation and training lossess '''
plot_1 = ax1.plot(range(0, 20), training_stats['Training_Loss'], color = 'blue', label = 'Train Loss',\
             marker = 's', linewidth=2.0, markersize = 10)

plot_2 = ax1.plot(range(0, 20), training_stats['Val_Loss'], color = 'blue', label = 'Val Loss',\
             marker = 'o', linewidth=2.0, markersize = 10)

ax1.tick_params(axis ='y', labelcolor = 'blue',labelsize=20, width=3)
ax1.tick_params(axis ='x', labelcolor = 'black',labelsize=20, width=3)
ax1.legend(fontsize = 30)
plt.xticks(range(0,20, 2))
# ax1.set_xlim([-1, 30])
ax1.set_ylabel("BCE Loss", fontsize = 30, labelpad = 10, color = 'blue')


''' Right plot contains the training and validation accuracies '''
ax1a = plt.subplot(1,2, 2)
plot_11 = ax1a.plot(range(0, 20), training_stats['Training_Accuracy'], color = 'red', label = 'Train Acc',\
             marker = 's', linewidth=2.0, markersize = 10)

plot_22 = ax1a.plot(range(0, 20), training_stats['Val_Accuracy'], color = 'red', label = 'Val Acc',\
             marker = 'o', linewidth=2.0, markersize = 10)
ax1a.legend(fontsize = 30)
plt.xticks(range(0,20,2))
# ax1a.set_xlim([-1, 30])
ax1a.tick_params(axis ='y', labelcolor = 'red',labelsize=20, width=3)
ax1a.tick_params(axis ='x', labelcolor = 'black',labelsize=20, width=3)
ax1a.set_ylabel("Accuracy", fontsize = 30, labelpad = 10, color = 'red')
```

```
# for ax1
ax1.tick_params(which='both', width=2.5)
ax1.tick_params(which='major', length=15)
ax1.tick_params(which='minor', length=5)
ax1.tick_params(which = 'both', direction = 'in')

# for ax1a
ax1a.tick_params(which='both', width=2.5)
ax1a.tick_params(which='major', length=15)
ax1a.tick_params(which='minor', length=5)
ax1a.tick_params(which = 'both', direction = 'in')

# set various colors
ax1a.spines['bottom'].set_color('black')
ax1a.spines['top'].set_color('black')
ax1a.spines['right'].set_color('black')
ax1a.spines['right'].set_linewidth(2)
ax1a.spines['top'].set_linewidth(2)
ax1a.spines['bottom'].set_linewidth(2)
ax1a.spines['left'].set_color('black')
ax1a.spines['left'].set_lw(2)

# set various colors
ax1.spines['bottom'].set_color('black')
ax1.spines['top'].set_color('black')
ax1.spines['right'].set_color('black')
ax1.spines['right'].set_linewidth(2)
ax1.spines['top'].set_linewidth(2)
ax1.spines['bottom'].set_linewidth(2)
ax1.spines['left'].set_color('black')
ax1.spines['left'].set_lw(2)


ax1.grid(True, which = 'major', alpha = 1, linestyle='--', linewidth = 1)
ax1a.grid(True, which = 'major', alpha = 1, linestyle='--', linewidth = 1)


plt.subplots_adjust(wspace=0.25,hspace=0.)
fig.text(0.5, 0.01, 'Epochs', ha='center', va='center', fontsize = 30)

fig.text(0.5, 0.95, 'LSTM Performance on Training and Validation Datasets', ha='center', va='center', fontsize = 30)
```

Out[91]:
```
Text(0.5, 0.95, 'LSTM Performance on Training and Validation Datasets')
```



As we can see, the train loss and validation set are decreasing as we train the model for 20 epochs. If we see clearly, the traininig loss is still decreasing whereas the validation loss is starting to diverge. Perhaps the model is starting to overfit. The accuracy plot shows that the model's accuracy on both training and validation dataset is increasing steadily.

Model Evaluation

In [114…

```
'''
This function takes in a trained model and a dataloader and makes predictions

'''

def make_predictions_on_dataloaders(trained_model, dataloader):
    target = []
    probabilities = []
    predictions = []
```

```
    pred_probs_for_all_class = []

    with torch.no_grad():
        trained_model.eval()
        for features_, labels in tqdm(dataloader):
            features_ = features_.to(device)
            labels = labels.to(device)
            yb = trained_model(features_)
            probs = yb.cpu().detach().numpy()
            preds = (probs >= 0.5).astype(int)  # threshold at 0.5 for binary classification
            target.append(labels.cpu().detach().numpy())
            probabilities.append(probs)
            predictions.append(preds)
            pred_probs_for_all_class.append(np.concatenate((1-probs, probs), axis=1))  # add negative class predictions

    return target, probabilities, predictions, pred_probs_for_all_class
```

In [116…

```
target, probabilities, predictions, pred_probs_for_all_class = make_predictions_on_dataloaders(model, testloader)
```

```
100%|██████████| 15/15 [00:04<00:00,  3.51it/s]
```

In [121…

```
def flatten(input_arr):
    output = []
    for i in input_arr:
        for j in i:
            output.append(j)
    return output
```

In [122…

```
predictions = flatten([list(i) for i in predictions])
target = flatten([list(i) for i in target])
probabilities = flatten([list(i) for i in probabilities])
pred_probs_for_all_class = np.array(flatten(pred_probs_for_all_class))
```

In [124…

```
from sklearn.metrics import accuracy_score, classification_report, roc_auc_score, confusion_matrix, roc_curve, precisio
```

In [125…

```
print("The testing accuracy is: {}".format(accuracy_score(target, predictions)*100))
```

```
The testing accuracy is: 84.50666666666666
```

Our LSTM model has achieved a test accuacy of almost 84.5%, which is very good.

In [126…

```
print("Precision (Test): ", precision_score(target, predictions, average = 'weighted'))
print("Recall (Test): ", recall_score(target, predictions, average = 'weighted'))
```

```
print("F1 (Test): ", f1_score(target, predictions, average = 'weighted'))
```

```
Precision (Test):  0.845088204369394
Recall (Test):  0.8450666666666666
F1 (Test):  0.8450687820616207
```

Additionally, the model has also achieved a precision, recall, and f1 scores of about 84.5% each respectively.

In [127…

```
print("Classification Report")
print(classification_report(target, predictions))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.84      0.85      0.84      3722
           1       0.85      0.84      0.85      3778

    accuracy                           0.85      7500
   macro avg       0.85      0.85      0.85      7500
weighted avg       0.85      0.85      0.85      7500
```

The above classification report shows the precision, recall and f1 score of the LSTM for each class in test dataset.

The choice of precision over recall depends upon the problem statement. Lets suppose that I am building a flower classification model to identify toxic or dangerous plants, it may be more important to have high precision. This is to ensure that I can minimize false positives and avoid misclassifying a safe plant as dangerous. However, if I am building a model to identify rare or endangered plant species, it might be in my interest to have high recall, so that I am able to identify as many positive cases as possible, even if it results in some false positives.

In [130…

```
cf_matrix = confusion_matrix(target, predictions)
dataframe = pd.DataFrame(cf_matrix, index = ['Negative', 'Positive'], columns = ['Negative', 'Positive'])
```

In [131…

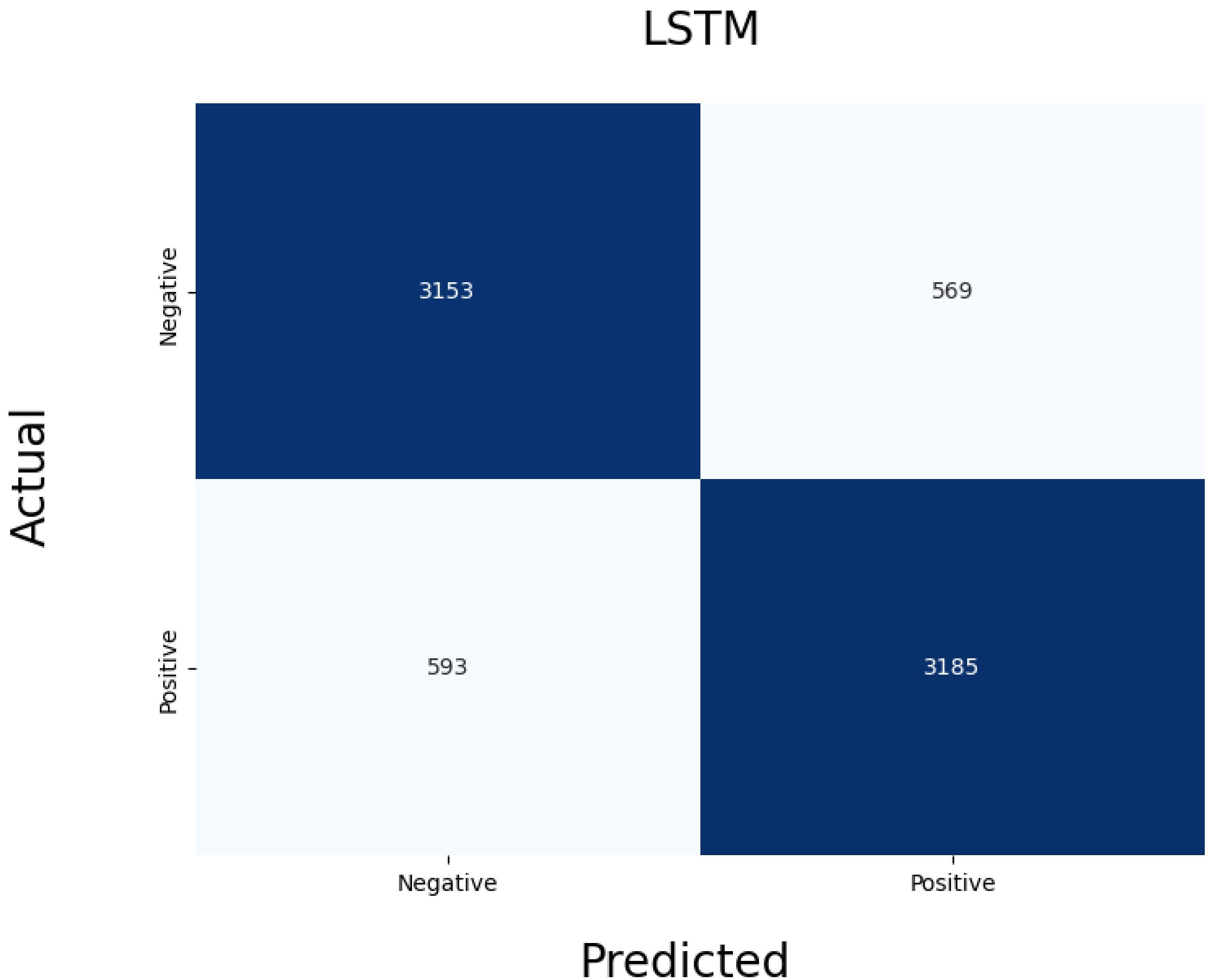
```
dataframe
```

Out[131]:

	Negative	Positive
Negative	3153	569
Positive	593	3185

In [133…

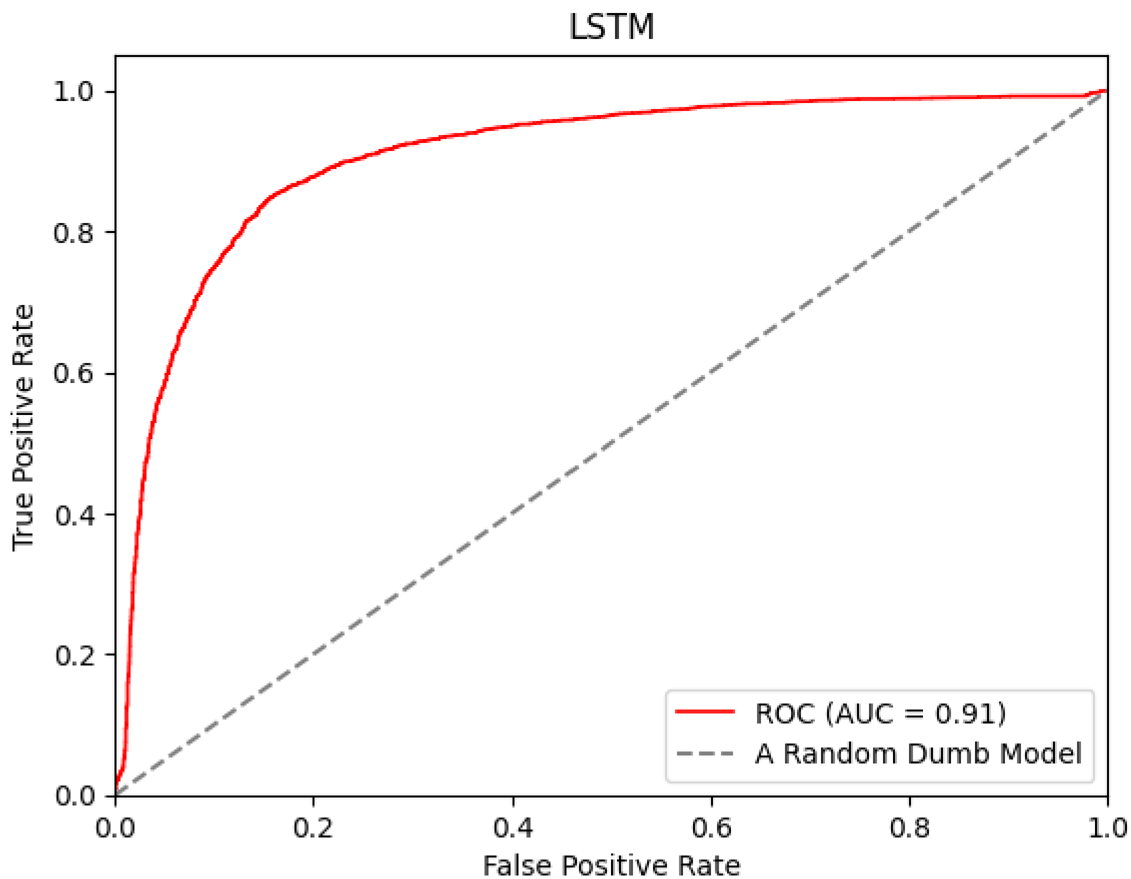
```
fig, axes = plt.subplots(1, 1, figsize=(8, 6))

ax1 = plt.subplot(1, 1, 1)

sns.heatmap(dataframe, cmap="Blues", annot = True, fmt="d", cbar =False)
fig.text(0.5, 0.00, 'Predicted', ha='center', va='center', fontsize = 20)
fig.text(0.0, 0.5, 'Actual', ha='center', va='center', rotation='vertical', fontsize = 20)
ax1.text(0.5, 1.08, 'LSTM',
    horizontalalignment='center',
    fontsize=20,
    transform = ax1.transAxes);
```



The model has a very high levels of true positive and negative predictions. Additionally, The model predicted 593 samples as negative class when infact they were postive reviews. Similarlty, the model predicted 569 samples as positive reviews when infact those samples were negative reviews respectively.

In [137…

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# y_test are the true labels and y_score are the predicted probabilities for the positive class
fpr, tpr, thresholds = roc_curve(target, probabilities, pos_label=1)

# Compute AUC score
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.plot(fpr, tpr, lw=1.25, label='ROC (AUC = %0.2f)' % (roc_auc), color = 'red')
plt.plot([0, 1], [0, 1], '--', color='gray', label='A Random Dumb Model')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('LSTM')
plt.legend(loc="lower right")
plt.show()
```



In [139…

```
auc_score_ovr = roc_auc_score(target, probabilities)
print("AUC SCORE (Test Set): {}".format(auc_score_ovr))
```

```
AUC SCORE (Test Set): 0.9059854074708947
```

An auc score of 0.5 means that our model is making random predictions. Similarly, an auc score of 0 means that the model is predicting positive classes and negative and vice versa. An auc score of about 94.4 % means that the model has good ability to diffrentiate between positive and negative classes. An AUC score of 0.9 indicates that the model has a high probability of correctly ranking a randomly chosen positive instance higher than a randomly chosen negative instance. Specifically, if a positive instance is randomly selected from the dataset and compared to a negative instance that was also randomly selected, then there is a 90% chance that the model will assign a higher predicted probability to the positive instance than to the negative instance.

Therefore, an AUC score of 0.9 suggests that the model is highly capable of distinguishing between the positive and negative classes, and it has a high true positive rate while maintaining a low false positive rate, which is desirable for many classification tasks.

Results and Prediction

In [140…
```
def predict_sentiment(text):
  text = custom_preprocessing_pipeline(custom_cleaning_pipeline(text))
  text = [[converting_integers_and_words[word] for word in text.split() if word in converting_integers_and_words.keys()
  text = paddind_sequences(text, pad_id=converting_integers_and_words['<PAD>'], seq_length=max_length_of_sequence)
  text_tensor = torch.tensor(text).to(device)

  return model(text_tensor).cpu().detach().numpy()
```

In [142…
```
classes = ['Negative', 'Positive']
```

In [145…
```
text = 'The movie is very good. The actors were good. I loved the movie'
prob = predict_sentiment(text)[0][0]
pred = 1 if prob >= 0.5 else 0

if pred == 0:
  prob = 1 - prob
print("The predicted class is {}, with a predicted probability of {}.".format(classes[pred], round(prob, 5)))
```

```
The predicted class is Positive, with a predicted probability of 0.9912099838256836.
```

In [146…
```
text = 'The movie is very bad. The actors were pathetic. I think I wasted my money on this movie.'
prob = predict_sentiment(text)[0][0]
pred = 1 if prob >= 0.5 else 0

if pred == 0:
  prob = 1 - prob
print("The predicted class is {}, with a predicted probability of {}.".format(classes[pred], round(prob, 5)))
```

```
The predicted class is Negative, with a predicted probability of 0.9955.
```

Lets make a single prediction from a real data

In [161…
```
df.sample(1, random_state = 420)
```

Out[161]:

	processed	label
10427	**register imdb post comment awful movie ismy ca...**	**0**

In [162…

```
random_text = df['processed'].sample(1, random_state = 420).values[0];random_text
```

Out[162]:

```
'register imdb post comment awful movie ismy cat ball string better storyline worst act ive ever see wipe almost entire cast movie within 5 minutes leave bite desire wasnt single scare moment movie exception watch movie halloween tv around seem like couldve good story roll credit say chasey lie bite loss didnt recognize right away scene already couldve say oh yeah im glad saw hotel didnt pay id real tick pay cent see normally like least find redeem factor movie one exception bad even amuse sogooditsbadits plain bad'
```

In [165…

```
prob = predict_sentiment(random_text)[0][0]
pred = 1 if prob >= 0.5 else 0

if pred == 0:
  prob = 1 - prob
print("The predicted class is {}, with a predicted probability of {}.".format(classes[pred], round(prob, 5)))
```

```
The predicted class is Negative, with a predicted probability of 0.99564.
```

Conclusion

We trained a LSTM classifier to perform sentiment analysis on a IMDB dataset. It was seen that a LSTM model is very much able to perform classification on sequential data. Traditional ML models like Random Forest and XGBoost might not perform well on such type of data whereas a recurrent neural network like LSTM is very much capable for capturing relationships in a sequential data.