MPST: Movie Plot Synopses with Tags

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

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee64

Enter your authorization code:
··········
Mounted at /content/drive
```

1. BUSINESS PROBLEM

1.1 DESCRIPTION

Abstract Social tagging of movies reveals a wide range of heterogeneous information about movies, like the genre, plot structure, soundtracks, metada emotional experiences. Such information can be valuable in building automatic systems to create tags for movies. Automatic tagging systems can hel recommendation engines to improve the retrieval of similar movies as well as help viewers to know what to expect from a movie in advance. In this pa out to the task of collecting a corpus of movie plot synopses and tags. We describe a methodology that enabled us to build a fine-grained set of aroun exposing heterogeneous characteristics of movie plots and the multi-label associations of these tags with some 14K movie plot synopses. We investig these tags correlate with movies and the flow of emotions throughout different types of movies. Finally, we use this corpus to explore the feasibility of from plot synopses. We expect the corpus will be useful in other tasks where analysis of narratives is relevant.

1.2 SOURCES

DATA SOURCE: https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags

RESEARCH PAPER: https://www.aclweb.org/anthology/L18-1274

RESEARCH PAPER: http://ritual.uh.edu/mpst-2018/

1.3 Real World / Business Objectives and Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience.
3. No strict latency constraints.
4. Avoid redundancy in tags generation.

---

2. Machine Learning Overview

2.1 DATA OVERVIEW

Contains all the IMDB id, title, plot synopsis, tags for the movies. There are 14,828 movies' data in total. The split column indicates where the data insta in the Train/Dev/Test split.

imdb_id: IMDB id of the movie

title: Title of the move

plot_synopsis: Plot Synopsis of the move

tagsTags: assigned to the move

split: Position of the movie in the standard data split

synopsis_source:From where the plot synopsis was collected

Example:

imdb_id: tt1733125

title:Dungeons & Dragons: The Book of Vile Darkness

plot_synopsis: Two thousand years ago, Nhagruul the Foul, a sorcerer who reveled in corrupting the innocent and the spread of despair, neared the end days and was dismayed. Consumed by hatred for the l...

tagsTags: Violence

split: Train synopsis source: IMDB

2.2 TYPE OF MACHINE LEARNING PROBLEM

This problem is a multi-label classification problem. Each movie can have multiple labels/tags that they can be classified as and are not mutually exclu

http://scikit-learn.org/stable/modules/multiclass.html

2.3 PERFOMANCE METRICS

Micro-Averaged F1-Score (Mean F Score) : The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reache value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

'Micro f1 score': Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better metric when we have imbalance. Since the distribution of our tags is imbalanced, we will be using this.

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.hamming_loss.html

3.EXPLORATORY DATA ANALYSIS

3.1 DATA LOADING AND DE-DUPLICATION

```
!pip install scikit-multilearn
```

```
Collecting scikit-multilearn
  Downloading https://files.pythonhosted.org/packages/bb/1f/e6ff649c72a1cdf2c7a1d31eb21705110ce1c5d3e7e26b2cc300e163
     |████████████████████████████████| 92kB 7.3MB/s
Installing collected packages: scikit-multilearn
Successfully installed scikit-multilearn-0.2.0
```

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense, Dropout, Flatten, Reshape, Softmax, InputLayer
from keras.layers import Conv2D, MaxPooling2D, Conv1D, MaxPooling1D, TimeDistributed
```

```
all_data = pd.read_csv('/content/drive/My Drive/Assignment 27-Case study ml/mpst-movie-plot-synopses-with-tags/mpst_full_data.csv'
all_data.columns
```

```
Index(['imdb_id', 'title', 'plot_synopsis', 'tags', 'split',
       'synopsis_source'],
      dtype='object')
```

```
all_data.head()
```

	imdb_id	title	plot_synopsis	
0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, g
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	
2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	
3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring,
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana	cruelty, murd

```
#Creating db file from csv
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('start_data.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///start_data.db')
    start = dt.datetime.now()
    chunksize = 180000
    j = 0
    index_start = 1
    for df in pd.read_csv('mpst_full_data.csv',  chunksize=chunksize, iterator=True, encoding='utf-8', ):
        df.index += index_start
        j+=1
        print('{} rows'.format(j*chunksize))
        df.to_sql('all_data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
    print("Time taken to run this cell :", datetime.now() - start)
```

```
if os.path.isfile('start_data.db'):
    start = datetime.now()
    con = sqlite3.connect('start_data.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM all_data""", con)
    #Always remember to close the database
    print("Number of rows in the database :","\n",num_rows['count(*)'].values[0])
    con.close()
    print("Time taken to count the number of rows :", datetime.now() - start)
```

```
Number of rows in the database :
 14828
Time taken to count the number of rows : 0:00:00.419101
```

3.1.1 CHECKING FOR DUPLICATES

```
#Learn SQL: https://www.w3schools.com/sql/default.asp
if os.path.isfile('start_data.db'):
    start = datetime.now()
    con = sqlite3.connect('start_data.db')
    df_no_dup = pd.read_sql_query('SELECT imdb_id, Title, plot_synopsis, Tags, split, COUNT(*) as cnt_dup FROM all_data GROUP BY t
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:10.158982
```

```
print("number of duplicate movies :", num_rows['count(*)'].values[0]- df_no_dup.shape[0], "(",(1-((df_no_dup.shape[0])/(num_rows['
```

```
number of duplicate movies : 651 ( 4.3903425950903685 % )
```

There seems to be data duplication occuring in the dataset.

```
df_no_dup.cnt_dup.value_counts()
```

```
1    13674
2      415
3       61
4       13
6        6
5        4
8        3
9        1
Name: cnt_dup, dtype: int64
```

We can see that some of our movie names appear more than once and upto even 5 times.

```
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["tags"].apply(lambda text: len(text.split(" ")))
# adding a new feature number of tags per question
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

```
Time taken to run this cell : 0:00:00.023475
```

	imdb_id	title	plot_synopsis	tags
0	tt0284741	Julius Caesar	The play opens with the commoners of Rome cele...	violence, murder
1	tt1202150	Beauty and the Beast	A widower merchant lives in a mansion with his...	romantic, fantasy
2	tt0040558	Macbeth	=== Act I ===\nThe play opens amidst thunder a...	tragedy, murder
3	tt0032028	The Three Musketeers	In Venice, the musketeers Athos, Porthos, and ...	good versus evil, action, historical fiction
4	tt0171359	Hamlet	=== Act I ===\nThe protagonist of Hamlet is Pr...	murder, violence, good versus evil, insanity, ...

```
df_no_dup.tag_count.value_counts()
```

```
1     4922
2     2832
3     1856
4     1271
5      922
6      633
7      479
8      360
9      246
10     184
11     131
12      95
13      67
14      44
15      35
16      31
17      21
18      18
20      10
19       8
21       3
26       2
27       2
22       2
23       2
24       1
Name: tag_count, dtype: int64
```

We can see that the tags repeat across different movies.

3.1.2 De-Duplication

```
d1= all_data.sort_values('imdb_id', axis=0, ascending=True, inplace=False, kind='quicksort', na_position='last')
d1.shape
```

```
(14828, 6)
```

```
dedup_data=d1.drop_duplicates(subset={"title", "plot_synopsis", "tags", "split"}, keep='first', inplace=False)
dedup_data.shape
```

```
(14781, 6)
```

```
(dedup_data['imdb_id'].size*1.0)/(d1['imdb_id'].size*1.0)*100
```

```
99.68303210142973
```

```
dedup_data.head()
```

	imdb_id	title	plot_synopsis	tags	split	synop
8814	tt0000091	Le manoir du diable	The film opens with a large bat flying into a ...	paranormal, gothic	train	
7085	tt0000225	La belle et la bête	A widower merchant lives in a mansion with his...	fantasy	train	
11909	tt0000230	Cendrillon	A prologue in front of the curtain, suppressed...	fantasy	train	
548	tt0000417	Le voyage dans la lune	At a meeting of the Astronomic Club, its presi...	psychedelic, satire	train	
6042	tt0000488	The Land Beyond the Sunset	Joe is an impoverished New York newsboy who li...	fantasy, storytelling	train	

3.2 ANALYSIS OF TAGS

3.2.1 TOTAL NUMBER OF UNIQUE TAGS

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'split()' will token
def tokenize(x):
    x=x.split(',')
    tags=[i.strip() for i in x] #Some tags contains whitespaces before them, so we need to strip them
    return tags

vectorizer = CountVectorizer(tokenizer = tokenize)
tag_dtm = vectorizer.fit_transform(dedup_data['tags'])


print("Number of movies present in the entire dataset :", tag_dtm.shape[0])
print("Number of unique tags present in the entire dataset:", tag_dtm.shape[1])
```

```
Number of movies present in the entire dataset : 14781
Number of unique tags present in the entire dataset: 71
```

```
tags = vectorizer.get_feature_names()
#Lets look at the tags we have.
print("Some of the tags we have :", tags[:10])
```

```
Some of the tags we have : ['absurd', 'action', 'adult comedy', 'allegory', 'alternate history', 'alternate reality'
```

3.2.2 FREQUENCY OF TAGS

```
# https://stackoverflow.com/questions/15115765/how-to-access-sparse-matrix-elements
#Lets now store the document term matrix in a dictionary.
freqs = tag_dtm.sum(axis=0).A1
result = dict(zip(tags, freqs))


#Saving this dictionary to csv files.
if not os.path.isfile('tag_counts_dict_dtm.csv'):
    with open('tag_counts_dict_dtm.csv', 'w') as csv_file:
        writer = csv.writer(csv_file)
        for key, value in result.items():
            writer.writerow([key, value])
tag_df = pd.read_csv("tag_counts_dict_dtm.csv", names=['Tags', 'Counts'])
tag_df.head()
```

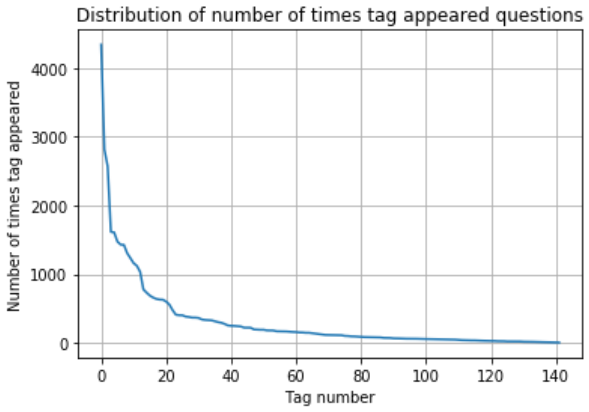
	Tags	Counts
0	absurd	178
1	action	600
2	adult comedy	110
3	allegory	82
4	alternate history	79

```
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)
tag_counts = tag_df_sorted['Counts'].values
tag_df_sorted.head()
```

```
tag_df.describe()
```

	Counts
count	142.000000
mean	310.915493
std	567.182216
min	3.000000
25%	49.000000
50%	112.000000
75%	309.000000
max	4343.000000

```
plt.plot(tag_counts)
plt.title("Distribution of number of times tag appeared questions")
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```




```
plt.plot(tag_counts, c='b')
plt.scatter(x=list(range(0,100,5)), y=tag_counts[0:100:5], c='orange', label="quantiles with 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,100,25)), y=tag_counts[0:100:25], c='m', label = "quantiles with 0.25 intervals")

for x,y in zip(list(range(0,100,25)), tag_counts[0:100:25]):
    plt.annotate(text="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))

plt.title('first 100 tags: Distribution of number of times tag appeared questions')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.legend()
plt.show()
print(len(tag_counts[0:100:5]), tag_counts[0:100:5])
```

OBSERVATIONS

1) The 2 most commonly occuring tags are murder and violence.

2) About 15 tags are used more than 1000 times.

3) Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelem

4)Each tag appears a minimum of 3 times and an average of 310 times in the dataset.

3.2.3 TAGS PER MOVIE TITLE

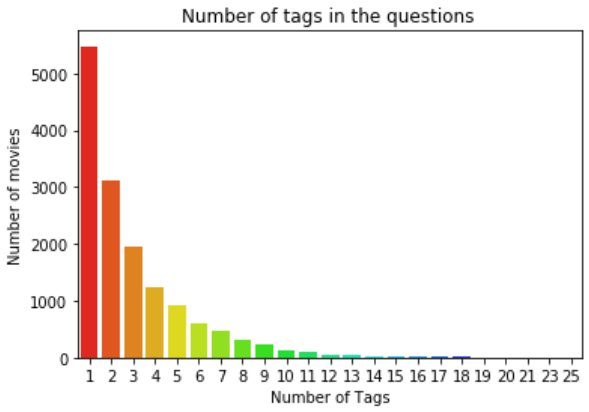
```
#Storing the count of tag in each question in list 'tag_count'
tag_movie_count = tag_dtm.sum(axis=1).tolist()
#Converting list of lists into single list, we will get [[3], [4], [2], [2], [3]] and we are converting this to [3, 4, 2, 2, 3]
tag_movie_count=[int(j) for i in tag_movie_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_movie_count)))

print(tag_movie_count[:5])
```

```
We have total 14781 datapoints.
[2, 1, 1, 2, 2]
```

```
print( "Maximum number of tags per Movie: %d"%max(tag_movie_count))
print( "Minimum number of tags per Movie: %d"%min(tag_movie_count))
print( "Avg. number of tags per Movie: %f"% ((sum(tag_movie_count)*1.0)/len(tag_movie_count)))
```

```
Maximum number of tags per Movie: 25
Minimum number of tags per Movie: 1
Avg. number of tags per Movie: 2.986943
```

```
sns.countplot(tag_movie_count, palette='gist_rainbow')
plt.title("Number of tags in the questions ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of movies")
plt.show()
```



OBSERVATIONS

1) The maximum number of tags for a movie are 25.

2) The minimum number of tags for a movie is 1.

3) The average number of tags for a movie is 2.99.

3.2.4 WORDCLOUD FOR TAGS

```
# Ploting word cloud
start = datetime.now()

# Lets first convert the 'result' dictionary to 'list of tuples'
tup = dict(result.items())
#Initializing WordCloud using frequencies of tags.
wordcloud = WordCloud(    background_color='black',
                          width=1600,
                          height=800,
                    ).generate_from_frequencies(tup)
```

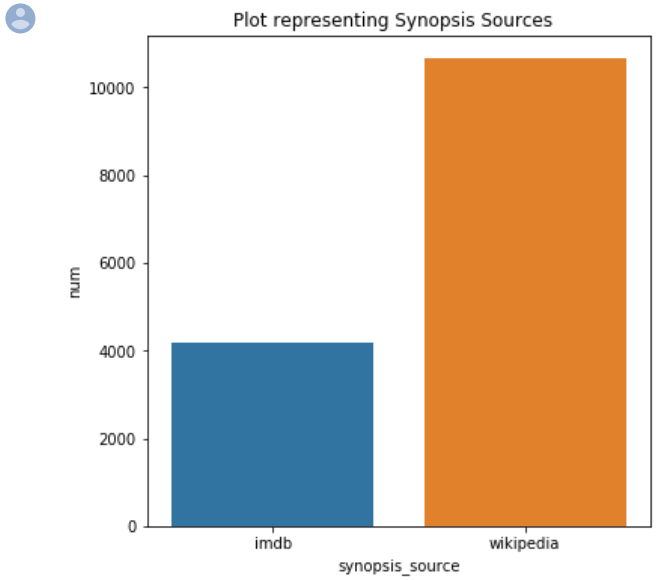
```
fig = plt.figure(figsize=(30,20))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
print("Time taken to run this cell :", datetime.now() - start)
```


```
Time taken to run this cell : 0:00:02.806203
```

```
con = sqlite3.connect('start_data.db')
no_dup = pd.read_sql_query('SELECT synopsis_source, COUNT(*) AS num FROM all_data GROUP BY synopsis_source', con)
con.close()
```

```
plt.figure(figsize=(6, 6))
```

```
plt.title ("Plot representing Synopsis Sources  ")
sns.barplot(no_dup['synopsis_source'],no_dup['num'])
plt.show()
```



3.3 PRE PROCESSING OF SYNOPSIS

STEPS TO BE FOLLOWED

Begin by removing the html tags.

Remove any punctuations or limited set of special characters like , or . or # etc.

Check if the word is made up of english letters and is not alpha-numeric.

Check to see if the length of the word is greater than 2 (as it was researched that there is no adjective in 2-letters).

Convert the word to lowercase.

Remove Stopwords.

Finally Snowball Stemming the word (it was obsereved to be better than Porter Stemming).

```
# https://stackoverflow.com/a/47091490/4084039
import re

def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase
```

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
# <br /><br /> ==> after the above steps, we are getting "br br"
# we are including them into stop words list
# instead of <br /> if we have <br/> these tags would have revmoved in the 1st step

stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

```
from tqdm import tqdm
from bs4 import BeautifulSoup

preprocessed_data = []
# tqdm is for printing the status bar
for sentance in tqdm(dedup_data['plot_synopsis'].values):
    sentance = re.sub(r"http\S+", "", sentance)
    sentance = BeautifulSoup(sentance, 'lxml').get_text()
    sentance = decontracted(sentance)
    sentance = re.sub("\S*\d\S*", "", sentance).strip()
    sentance = re.sub('[^A-Za-z]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e.lower() for e in sentance.split() if e.lower() not in stopwords)
    preprocessed_data.append(sentance.strip())
```

```
dedup_data['preprocessed_synopsis']= preprocessed_data
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-47-460909825fc5> in <module>
----> 1 dedup_data['preprocessed_synopsis']= preprocessed_data

NameError: name 'preprocessed_data' is not defined
```

SEARCH STACK OVERFLOW

```
import pickle
dedup_data.to_pickle("final")
```

```
final = pd.read_pickle("final")
final.head()
```

	imdb_id	title	plot_synopsis	tags	split	synopsis_
8814	tt0000091	Le manoir du diable	The film opens with a large bat flying into a ...	paranormal, gothic	train	wi
7085	tt0000225	La belle et la bête	A widower merchant lives in a mansion with his...	fantasy	train	wi
11909	tt0000230	Cendrillon	A prologue in front of the curtain, suppressed...	fantasy	train	wi
548	tt0000417	Le voyage dans la lune	At a meeting of the Astronomic Club, its presi...	psychedelic,	train	

```
final_with_len=final
final_with_len['synopsis_length']  = final_with_len['plot_synopsis'].str.len()
```

```
final_with_len=final_with_len.drop(columns=['plot_synopsis', 'synopsis_source','imdb_id'])
```

```
final_with_len.head()
```

	title	tags	split	preprocessed_synopsis	
8814	Le manoir du diable	paranormal, gothic	train	film opens large bat flying medieval castle ba...	[film, opens, large
7085	La belle et la bête	fantasy	train	widower merchant lives mansion six children th...	[widower, merchant
11909	Cendrillon	fantasy	train	prologue front curtain suppressed premiere int...	[prologue, front, cur
548	Le voyage dans la lune	psychedelic, satire	train	meeting astronomic club president professor ba...	[meeting, astronomi
6042	The Land Beyond the Sunset	fantasy, storytelling	train	joe impoverished new york newsboy lives abusiv...	[joe, impoverished,

```
final_with_len.to_pickle("final_with_len")
```

```
final = pd.read_pickle("/content/drive/My Drive/Assignment 27-Case study ml/mpst-movie-plot-synopses-with-tags/final_with_len")
```

3.3.1 EDA of SYNOPSIS

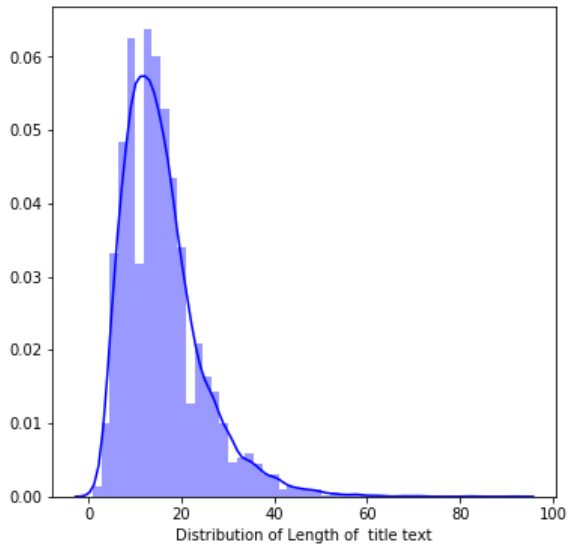
```
synopsis_eda=pd.DataFrame()
synopsis_eda["Length_Title"] = final['title'].apply(lambda x: len(str(x)))
synopsis_eda["Length_Plot_Synopsis"] = final['plot_synopsis'].apply(lambda x: len(str(x)))
```

```
synopsis_eda.head()
```

	Length_Title	Length_Plot_Synopsis
8814	19	1330
7085	19	5540
11909	10	4097
548	22	2424
6042	26	929

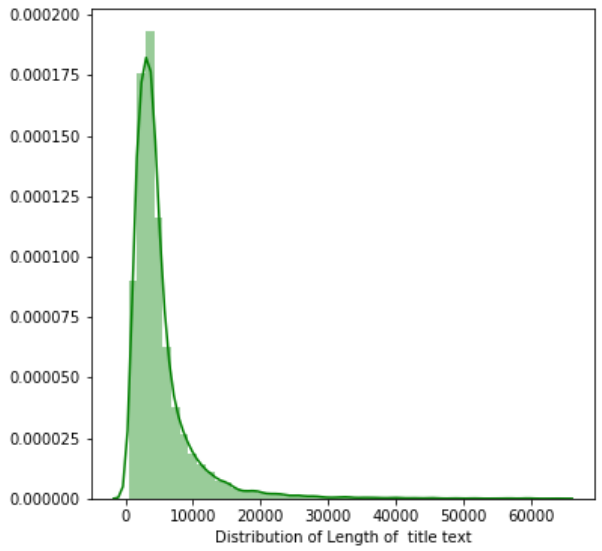
```
plt.figure(figsize=(6, 6))
sns.distplot([synopsis_eda['Length_Title']], color = 'blue', axlabel="Distribution of Length of  title text")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2046384e5f8>
```




```
plt.figure(figsize=(6, 6))
sns.distplot([synopsis_eda['Length_Plot_Synopsis']], color = 'g', axlabel="Distribution of Length of  title text")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2046355d1d0>
```



OBSERVATIONS

1) We can see that the longest title consists of 92 words and the shortest is only 1 word.

2) Titles have an average length of 15.9 words.

3) We can see that the longest synopsis consists of 63959 words and the shortest is 442 word.

4) Synopsis have an average length of 5140 words.

4. MACHINE LEARNING MODELS

4.1 SPLITTING THE DATA

```
#https://stackoverflow.com/questions/24775648/element-wise-logical-or-in-pandas

train_data=final.loc[(final['split'] == 'train') | (final['split'] == 'val')]
test_data=final.loc[(final['split'] == 'test')]

x_train = train_data['preprocessed_synopsis']
y_train = train_data['tags']

x_test = test_data['preprocessed_synopsis']
y_test = test_data['tags']


print("The shape of training data is :" ,x_train.shape)
print("The shape of test data is :" ,x_test.shape)
```

```
The shape of training data is : (11816,)
The shape of test data is : (2965,)
```

4.2 Converting tags for multilabel problems

```
#https://stackoverflow.com/questions/15547409/how-to-get-rid-of-punctuation-using-nltk-tokenizer

def tokenize(sent):
    sent=sent.split(',')
    tags=[i.strip() for i in sent]
    return tags

# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true')
multilabel_y_train = vectorizer.fit_transform(y_train)
multilabel_y_test = vectorizer.transform(y_test)


print("The shape of training label data is :" ,multilabel_y_train.shape)
print("The shape of test label data is :" ,multilabel_y_test.shape)
```

```
The shape of training label data is : (11816, 71)
The shape of test label data is : (2965, 71)
```

4.3 DATA FEATURIZATION

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2",
                             tokenizer = lambda x: x.split())
x_train_multilabel = vectorizer.fit_transform(x_train)
x_test_multilabel = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:04.631263
```

```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 58539) Y : (11816, 71)
Dimensions of test data X: (2965, 58539) Y: (2965, 71)
```

4.3.1 Applying Logistic Regression with SGD with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:07:34.121465
```

```
print('The best hyper parameters are ', gsv.best_params_)
```

```
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l1'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.07655986509274873
macro f1 score : 0.06766277572929784
micro f1 score : 0.27901234567901234
hamming loss : 0.041612236657720354
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.55      0.09      0.15       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.00      0.00      0.00        18
           5       1.00      0.03      0.06        35
           6       0.33      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00        45
          10       0.00      0.00      0.00        11
          11       0.00      0.00      0.00        40
          12       0.75      0.03      0.05       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.30      0.07      0.12       368
          18       0.00      0.00      0.00        27
          19       0.00      0.00      0.00        97
          20       0.43      0.17      0.24       551
          21       0.00      0.00      0.00        41
          22       0.50      0.01      0.02        85
          23       1.00      0.02      0.05        42
          24       0.00      0.00      0.00        83
          25       0.43      0.02      0.04       159
          26       0.60      0.16      0.25       112
          27       0.00      0.00      0.00        11
          28       0.32      0.11      0.17       596
          29       0.64      0.14      0.23       190
          30       0.57      0.14      0.23        83
          31       0.00      0.00      0.00        12
          32       1.00      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       0.75      0.12      0.21        25
          35       0.00      0.00      0.00        29
          36       0.47      0.09      0.15        92
          37       0.27      0.02      0.03       172
          38       0.24      0.03      0.05       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.25      0.01      0.02        93
          43       0.63      0.58      0.61      1155
          44       0.33      0.02      0.03       117
          45       0.26      0.03      0.06       145
          46       0.00      0.00      0.00         5
          47       0.31      0.03      0.06       129
          48       1.00      0.03      0.05        36
          49       0.00      0.00      0.00        44
          50       1.00      0.04      0.07        28
          51       0.00      0.00      0.00        51
          52       0.61      0.16      0.25       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.35      0.11      0.17       507
          57       0.54      0.26      0.35       587
          58       0.23      0.02      0.04       148
          59       0.31      0.02      0.04       173
          60       0.86      0.09      0.16        66
          61       0.00      0.00      0.00        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.28      0.06      0.09       228
          66       0.00      0.00      0.00        27
          67       0.50      0.05      0.09       119
          68       0.62      0.47      0.53       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.54      0.19      0.28      9021
   macro avg       0.26      0.05      0.07      9021
weighted avg       0.42      0.19      0.23      9021
```

```
weighted avg       0.42      0.19      0.25      9021
 samples avg       0.36      0.23      0.25      9021
```

4.3.2 SGD with Hinge loss with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
Time taken to run this cell : 0:08:13.580796
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l2'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.06880269814502529
macro f1 score : 0.10589837577318391
micro f1 score : 0.30321423644640644
hamming loss : 0.04798707930551267
Precision recall report :
              precision    recall  f1-score   support

           0       0.20      0.05      0.08        56
           1       0.25      0.12      0.16       129
           2       0.12      0.04      0.06        28
           3       0.00      0.00      0.00        22
           4       0.29      0.22      0.25        18
           5       0.14      0.03      0.05        35
           6       0.17      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.20      0.02      0.04        45
          10       0.50      0.09      0.15        11
          11       0.00      0.00      0.00        40
          12       0.19      0.04      0.07       115
          13       0.50      0.06      0.10        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.27      0.19      0.22       368
          18       0.00      0.00      0.00        27
          19       0.00      0.00      0.00        97
          20       0.37      0.29      0.32       551
          21       0.08      0.02      0.04        41
          22       0.12      0.04      0.06        85
          23       0.17      0.02      0.04        42
          24       0.03      0.01      0.02        83
          25       0.19      0.08      0.11       159
          26       0.49      0.25      0.33       112
          27       0.00      0.00      0.00        11
          28       0.28      0.21      0.24       596
          29       0.47      0.25      0.33       190
          30       0.39      0.23      0.29        83
          31       0.00      0.00      0.00        12
          32       0.38      0.12      0.18        26
          33       0.00      0.00      0.00        64
          34       0.36      0.20      0.26        25
          35       0.00      0.00      0.00        29
          36       0.26      0.13      0.17        92
          37       0.20      0.09      0.12       172
          38       0.21      0.11      0.15       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.09      0.03      0.05        93
          43       0.58      0.56      0.57      1155
          44       0.16      0.04      0.07       117
          45       0.20      0.09      0.12       145
          46       0.00      0.00      0.00         5
          47       0.32      0.09      0.13       129
          48       0.08      0.03      0.04        36
          49       0.20      0.05      0.07        44
          50       1.00      0.04      0.07        28
          51       0.09      0.02      0.03        51
          52       0.38      0.24      0.29       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.32      0.21      0.26       507
          57       0.44      0.36      0.40       587
          58       0.24      0.06      0.10       148
          59       0.23      0.08      0.12       173
          60       0.53      0.15      0.24        66
          61       0.25      0.04      0.07        51
          62       0.06      0.02      0.02        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.26      0.16      0.20       228
          66       0.00      0.00      0.00        27
          67       0.33      0.08      0.12       119
          68       0.56      0.50      0.53       911
          69       0.50      0.07      0.12        14
          70       0.00      0.00      0.00        13

   micro avg       0.40      0.24      0.30      9021
   macro avg       0.19      0.08      0.11      9021
weighted avg       0.34      0.24      0.27      9021
```

```
weighted avg       0.54      0.24      0.27      9021
 samples avg       0.36      0.28      0.28      9021
```

4.3.3 Logistic Regression with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__C=[0.001,0.1,1,100,500,1000])

gsv2 = GridSearchCV(OneVsRestClassifier(estimator= LogisticRegression(class_weight='balanced')), param_grid=param_grid, n_jobs=-1)
gsv2.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv2.best_params_)
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l1'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.05733558178752108
macro f1 score : 0.11574411774105492
micro f1 score : 0.29614044315865606
hamming loss : 0.05085148326722561
Precision recall report :
              precision    recall  f1-score   support

           0       0.14      0.05      0.08        56
           1       0.23      0.12      0.16       129
           2       0.19      0.11      0.14        28
           3       0.00      0.00      0.00        22
           4       0.36      0.28      0.31        18
           5       0.13      0.06      0.08        35
           6       0.23      0.10      0.14        30
           7       0.04      0.01      0.02        79
           8       0.00      0.00      0.00         8
           9       0.09      0.04      0.06        45
          10       0.50      0.18      0.27        11
          11       0.00      0.00      0.00        40
          12       0.14      0.05      0.08       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.22      0.18      0.19       368
          18       0.22      0.07      0.11        27
          19       0.02      0.01      0.01        97
          20       0.36      0.29      0.32       551
          21       0.13      0.05      0.07        41
          22       0.12      0.05      0.07        85
          23       0.12      0.02      0.04        42
          24       0.05      0.02      0.03        83
          25       0.19      0.09      0.13       159
          26       0.34      0.19      0.24       112
          27       0.17      0.09      0.12        11
          28       0.30      0.23      0.26       596
          29       0.39      0.24      0.29       190
          30       0.42      0.27      0.32        83
          31       0.00      0.00      0.00        12
          32       0.17      0.08      0.11        26
          33       0.04      0.02      0.02        64
          34       0.22      0.16      0.19        25
          35       0.09      0.03      0.05        29
          36       0.25      0.18      0.21        92
          37       0.16      0.08      0.11       172
          38       0.22      0.12      0.16       136
          39       0.20      0.03      0.05        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.09      0.04      0.06        93
          43       0.59      0.56      0.57      1155
          44       0.14      0.08      0.10       117
          45       0.18      0.10      0.13       145
          46       0.00      0.00      0.00         5
          47       0.28      0.12      0.17       129
          48       0.05      0.03      0.04        36
          49       0.14      0.05      0.07        44
          50       0.22      0.07      0.11        28
          51       0.19      0.06      0.09        51
          52       0.37      0.24      0.29       395
          53       0.12      0.05      0.07        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.33      0.25      0.28       507
          57       0.40      0.35      0.38       587
          58       0.10      0.04      0.06       148
          59       0.24      0.12      0.16       173
          60       0.28      0.08      0.12        66
          61       0.11      0.04      0.06        51
          62       0.03      0.02      0.02        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.23      0.17      0.19       228
          66       0.12      0.04      0.06        27
          67       0.15      0.08      0.10       119
          68       0.57      0.50      0.54       911
          69       0.50      0.07      0.12        14
          70       0.00      0.00      0.00        13

   micro avg       0.36      0.25      0.30      9021
   macro avg       0.17      0.09      0.12      9021
weighted avg       0.32      0.25      0.28      9021
```

```
weighted avg       0.32      0.29      0.28      9021
 samples avg       0.34      0.29      0.27      9021
```

OBSERVATIONS

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with TFIDF Vectorizer")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Log Loss",0.076, 0.27, 0.041])
x.add_row(["SGD With Hinge Loss",0.068, 0.30, 0.047 ])
x.add_row(["Logistic Regression",0.057, 0.29, 0.050 ])

print(x)
```

```
Comparison of models with TFIDF Vectorizer
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
|  SGD With Log Loss  |  0.076   |      0.27      |    0.041     |
| SGD With Hinge Loss |  0.068   |      0.3       |    0.047     |
| Logistic Regression |  0.057   |      0.29      |     0.05     |
+---------------------+----------+----------------+--------------+
```

4.4 DATA FEATURIZATION USING NGRAMS (1-3)

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2",
                             tokenizer = lambda x: x.split(),ngram_range=(1,3))
x_train_multilabel = vectorizer.fit_transform(x_train)
x_test_multilabel = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 954239) Y : (11816, 71)
Dimensions of test data X: (2965, 954239) Y: (2965, 71)
```

4.4.1 Applying Logistic Regression with SGD with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:23:41.336727
```

```
print('The best hyper parameters are ', gsv.best_params_)
```

```
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.0806070826306914
macro f1 score : 0.0611344863689567
micro f1 score : 0.2732553245222662
hamming loss : 0.04100895423128993
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.56      0.08      0.14       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.00      0.00      0.00        18
           5       1.00      0.03      0.06        35
           6       0.50      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00        45
          10       0.00      0.00      0.00        11
          11       0.00      0.00      0.00        40
          12       0.33      0.01      0.02       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.29      0.05      0.09       368
          18       0.00      0.00      0.00        27
          19       0.00      0.00      0.00        97
          20       0.50      0.15      0.22       551
          21       0.00      0.00      0.00        41
          22       0.67      0.02      0.05        85
          23       1.00      0.02      0.05        42
          24       0.14      0.01      0.02        83
          25       0.50      0.02      0.04       159
          26       0.64      0.12      0.21       112
          27       0.00      0.00      0.00        11
          28       0.34      0.10      0.15       596
          29       0.64      0.13      0.22       190
          30       0.58      0.13      0.22        83
          31       0.00      0.00      0.00        12
          32       0.33      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       0.60      0.12      0.20        25
          35       0.00      0.00      0.00        29
          36       0.42      0.05      0.10        92
          37       0.50      0.01      0.02       172
          38       0.27      0.02      0.04       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.50      0.02      0.04        93
          43       0.65      0.57      0.61      1155
          44       0.22      0.02      0.03       117
          45       0.21      0.02      0.04       145
          46       0.00      0.00      0.00         5
          47       0.18      0.02      0.03       129
          48       1.00      0.03      0.05        36
          49       0.00      0.00      0.00        44
          50       0.00      0.00      0.00        28
          51       0.00      0.00      0.00        51
          52       0.60      0.15      0.24       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.40      0.10      0.17       507
          57       0.58      0.26      0.36       587
          58       0.29      0.01      0.03       148
          59       0.22      0.01      0.02       173
          60       1.00      0.05      0.09        66
          61       0.00      0.00      0.00        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.26      0.04      0.07       228
          66       0.00      0.00      0.00        27
          67       0.36      0.04      0.08       119
          68       0.63      0.48      0.54       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.57      0.18      0.27      9021
   macro avg       0.24      0.04      0.06      9021
weighted avg       0.43      0.18      0.23      9021
```

```
weighted avg       0.45      0.18      0.25      9021
 samples avg       0.35      0.22      0.24      9021
```

4.4.2 SGD with Hinge loss with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
Time taken to run this cell : 0:24:30.466987
The best hyper parameters are  {'estimator__alpha': 0.0001, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.07723440134907252
macro f1 score : 0.03949363395836659
micro f1 score : 0.23131119041083356
hamming loss : 0.03990689499560601
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.50      0.04      0.07       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.00      0.00      0.00        18
           5       1.00      0.03      0.06        35
           6       0.00      0.00      0.00        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.00      0.00      0.00        45
          10       0.00      0.00      0.00        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.75      0.01      0.02       368
          18       0.00      0.00      0.00        27
          19       0.00      0.00      0.00        97
          20       0.61      0.04      0.07       551
          21       0.00      0.00      0.00        41
          22       0.00      0.00      0.00        85
          23       0.00      0.00      0.00        42
          24       0.00      0.00      0.00        83
          25       1.00      0.01      0.01       159
          26       0.71      0.13      0.23       112
          27       0.00      0.00      0.00        11
          28       0.33      0.01      0.02       596
          29       0.69      0.05      0.09       190
          30       0.50      0.08      0.14        83
          31       0.00      0.00      0.00        12
          32       1.00      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       1.00      0.08      0.15        25
          35       0.00      0.00      0.00        29
          36       0.50      0.05      0.10        92
          37       1.00      0.01      0.01       172
          38       0.00      0.00      0.00       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       1.00      0.01      0.02        93
          43       0.68      0.57      0.62      1155
          44       0.00      0.00      0.00       117
          45       0.00      0.00      0.00       145
          46       0.00      0.00      0.00         5
          47       0.25      0.01      0.02       129
          48       0.00      0.00      0.00        36
          49       0.00      0.00      0.00        44
          50       0.00      0.00      0.00        28
          51       0.00      0.00      0.00        51
          52       0.86      0.11      0.19       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.25      0.01      0.03       507
          57       0.68      0.14      0.23       587
          58       0.00      0.00      0.00       148
          59       0.00      0.00      0.00       173
          60       1.00      0.03      0.06        66
          61       0.00      0.00      0.00        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.67      0.01      0.02       228
          66       0.00      0.00      0.00        27
          67       0.57      0.03      0.06       119
          68       0.66      0.42      0.52       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.66      0.14      0.23      9021
   macro avg       0.23      0.03      0.04      9021
weighted avg       0.47      0.14      0.17      9021
```

```
weighted avg       0.47      0.14      0.17      9021
 samples avg       0.32      0.18      0.20      9021
```

4.4.3 Logistic Regression with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__C=[0.001,0.01,1,100], estimator__penalty=["l1","l2"])

gsv2 = GridSearchCV(OneVsRestClassifier(estimator= LogisticRegression(class_weight='balanced')), param_grid=param_grid, n_jobs=-1)
gsv2.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv2.best_params_)
```

```
Time taken to run this cell : 0:53:19.874091
The best hyper parameters are  {'estimator__penalty': 'l2', 'estimator__C': 100}
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l2'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.0806070826306914
macro f1 score : 0.08518436365933406
micro f1 score : 0.2932715358798146
hamming loss : 0.04201125810512315
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.44      0.11      0.17       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.67      0.11      0.19        18
           5       1.00      0.03      0.06        35
           6       0.33      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.33      0.02      0.04        45
          10       1.00      0.09      0.17        11
          11       0.00      0.00      0.00        40
          12       0.44      0.03      0.06       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.29      0.08      0.13       368
          18       1.00      0.04      0.07        27
          19       0.00      0.00      0.00        97
          20       0.48      0.20      0.28       551
          21       0.00      0.00      0.00        41
          22       0.60      0.04      0.07        85
          23       0.50      0.02      0.05        42
          24       0.09      0.01      0.02        83
          25       0.30      0.04      0.07       159
          26       0.58      0.19      0.28       112
          27       0.00      0.00      0.00        11
          28       0.32      0.12      0.18       596
          29       0.60      0.18      0.28       190
          30       0.56      0.18      0.27        83
          31       0.00      0.00      0.00        12
          32       0.33      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       0.38      0.12      0.18        25
          35       0.00      0.00      0.00        29
          36       0.55      0.12      0.20        92
          37       0.28      0.03      0.05       172
          38       0.36      0.06      0.10       136
          39       0.33      0.03      0.06        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.22      0.02      0.04        93
          43       0.64      0.58      0.61      1155
          44       0.21      0.03      0.05       117
          45       0.28      0.05      0.08       145
          46       0.00      0.00      0.00         5
          47       0.24      0.03      0.05       129
          48       0.25      0.03      0.05        36
          49       0.25      0.02      0.04        44
          50       1.00      0.04      0.07        28
          51       0.00      0.00      0.00        51
          52       0.53      0.16      0.25       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.38      0.14      0.21       507
          57       0.51      0.29      0.37       587
          58       0.25      0.02      0.04       148
          59       0.33      0.02      0.04       173
          60       0.86      0.09      0.16        66
          61       0.25      0.02      0.04        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.32      0.08      0.13       228
          66       1.00      0.04      0.07        27
          67       0.38      0.05      0.09       119
          68       0.63      0.49      0.55       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.53      0.20      0.29      9021
   macro avg       0.29      0.06      0.09      9021
weighted avg       0.42      0.20      0.25      9021
```

```
weighted avg       0.42      0.26      0.29      9021
 samples avg       0.37      0.24      0.26      9021
```

OBSERVATIONS

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with TFIDF Vectorizer and ngrams (1-3)")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Log Loss",0.080, 0.27, 0.041 ])
x.add_row(["SGD With Hinge Loss",0.077, 0.23, 0.039 ])
x.add_row(["Logistic Regression",0.080, 0.29, 0.042 ])

print(x)
```

```
Comparison of models with TFIDF Vectorizer and ngrams (1-3)
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
|  SGD With Log Loss  |   0.08   |      0.27      |    0.041     |
| SGD With Hinge Loss |  0.077   |      0.23      |    0.039     |
| Logistic Regression |   0.08   |      0.29      |    0.042     |
+---------------------+----------+----------------+--------------+
```

4.5 DATA FEATURIZATION USING BIGRAMS

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2",
                             tokenizer = lambda x: x.split(),ngram_range=(2,2))
x_train_multilabel = vectorizer.fit_transform(x_train)
x_test_multilabel = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:38.274666
```

```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 638790) Y : (11816, 71)
Dimensions of test data X: (2965, 638790) Y: (2965, 71)
```

4.5.1 Applying Logistic Regression with SGD with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
```

```
print('The best hyper parameters are ', gsv.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.07183811129848229
macro f1 score : 0.09556536636144591
micro f1 score : 0.33307557117750436
hamming loss : 0.04506567227988505
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.34      0.09      0.15       129
           2       0.00      0.00      0.00        28
           3       0.17      0.05      0.07        22
           4       0.67      0.11      0.19        18
           5       0.33      0.03      0.05        35
           6       0.50      0.03      0.06        30
           7       0.05      0.01      0.02        79
           8       0.00      0.00      0.00         8
           9       0.09      0.02      0.04        45
          10       0.50      0.09      0.15        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.27      0.17      0.21       368
          18       0.50      0.07      0.13        27
          19       0.10      0.01      0.02        97
          20       0.36      0.28      0.31       551
          21       0.00      0.00      0.00        41
          22       0.17      0.04      0.06        85
          23       0.33      0.02      0.04        42
          24       0.12      0.01      0.02        83
          25       0.29      0.06      0.10       159
          26       0.60      0.16      0.25       112
          27       0.00      0.00      0.00        11
          28       0.32      0.28      0.30       596
          29       0.46      0.16      0.24       190
          30       0.32      0.13      0.19        83
          31       0.00      0.00      0.00        12
          32       0.50      0.12      0.19        26
          33       0.00      0.00      0.00        64
          34       0.27      0.12      0.17        25
          35       0.00      0.00      0.00        29
          36       0.38      0.18      0.25        92
          37       0.26      0.03      0.05       172
          38       0.24      0.06      0.09       136
          39       0.00      0.00      0.00        32
          40       0.20      0.03      0.04        40
          41       0.00      0.00      0.00         5
          42       0.25      0.02      0.04        93
          43       0.63      0.64      0.64      1155
          44       0.32      0.06      0.10       117
          45       0.37      0.21      0.27       145
          46       0.00      0.00      0.00         5
          47       0.15      0.05      0.07       129
          48       0.00      0.00      0.00        36
          49       0.33      0.05      0.08        44
          50       0.00      0.00      0.00        28
          51       0.00      0.00      0.00        51
          52       0.52      0.16      0.24       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.39      0.29      0.33       507
          57       0.48      0.45      0.46       587
          58       0.45      0.06      0.11       148
          59       0.22      0.03      0.05       173
          60       0.29      0.03      0.05        66
          61       0.00      0.00      0.00        51
          62       0.17      0.02      0.03        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.25      0.16      0.19       228
          66       0.00      0.00      0.00        27
          67       0.33      0.08      0.12       119
          68       0.58      0.57      0.58       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.46      0.26      0.33      9021
   macro avg       0.20      0.07      0.10      9021
weighted avg       0.37      0.26      0.29      9021
```

```
weighted avg       0.37      0.26      0.25      9021
 samples avg       0.36      0.30      0.28      9021
```

4.5.2 SGD with Hinge loss with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.06475548060708262
macro f1 score : 0.13965776915128802
micro f1 score : 0.3007441996206041
hamming loss : 0.04552644704652875
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.33      0.22      0.26       129
           2       0.09      0.14      0.11        28
           3       0.29      0.09      0.14        22
           4       0.21      0.50      0.30        18
           5       0.22      0.17      0.19        35
           6       0.17      0.27      0.21        30
           7       0.12      0.03      0.04        79
           8       0.00      0.00      0.00         8
           9       0.17      0.07      0.10        45
          10       0.12      0.27      0.16        11
          11       0.00      0.00      0.00        40
          12       0.11      0.02      0.03       115
          13       0.12      0.06      0.08        18
          14       0.22      0.22      0.22         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.32      0.05      0.08       368
          18       0.30      0.11      0.16        27
          19       0.10      0.02      0.03        97
          20       0.43      0.17      0.24       551
          21       0.50      0.07      0.13        41
          22       0.19      0.07      0.10        85
          23       0.39      0.17      0.23        42
          24       0.05      0.01      0.02        83
          25       0.33      0.11      0.16       159
          26       0.60      0.29      0.39       112
          27       0.00      0.00      0.00        11
          28       0.38      0.16      0.23       596
          29       0.48      0.24      0.32       190
          30       0.39      0.40      0.40        83
          31       0.08      0.33      0.13        12
          32       0.26      0.38      0.31        26
          33       0.14      0.16      0.14        64
          34       0.13      0.24      0.17        25
          35       0.00      0.00      0.00        29
          36       0.42      0.28      0.34        92
          37       0.36      0.05      0.08       172
          38       0.38      0.13      0.20       136
          39       0.21      0.12      0.16        32
          40       0.17      0.03      0.04        40
          41       0.00      0.00      0.00         5
          42       0.12      0.02      0.04        93
          43       0.66      0.57      0.61      1155
          44       0.26      0.09      0.14       117
          45       0.34      0.29      0.31       145
          46       0.00      0.00      0.00         5
          47       0.31      0.12      0.18       129
          48       0.11      0.03      0.04        36
          49       0.33      0.05      0.08        44
          50       0.00      0.00      0.00        28
          51       0.12      0.06      0.08        51
          52       0.53      0.15      0.24       395
          53       0.00      0.00      0.00        65
          54       0.09      0.07      0.08        15
          55       0.00      0.00      0.00        37
          56       0.39      0.15      0.22       507
          57       0.48      0.26      0.34       587
          58       0.27      0.09      0.13       148
          59       0.34      0.09      0.15       173
          60       0.23      0.26      0.24        66
          61       0.00      0.00      0.00        51
          62       0.14      0.02      0.03        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.26      0.17      0.20       228
          66       0.00      0.00      0.00        27
          67       0.26      0.08      0.12       119
          68       0.63      0.45      0.53       911
          69       0.17      0.50      0.26        14
          70       0.00      0.00      0.00        13

   micro avg       0.44      0.23      0.30      9021
   macro avg       0.21      0.13      0.14      9021
weighted avg       0.40      0.23      0.28      9021
```

```
weighted avg       0.40      0.25      0.28      9021
 samples avg       0.33      0.25      0.25      9021
```

4.5.3 Logistic Regression with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__C=[0.001,0.01,1,100], estimator__penalty=["l1","l2"])

gsv2 = GridSearchCV(OneVsRestClassifier(estimator= LogisticRegression(class_weight='balanced')), param_grid=param_grid, n_jobs=-1)
gsv2.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv2.best_params_)
```

```
Time taken to run this cell : 0:20:36.086176
The best hyper parameters are  {'estimator__penalty': 'l2', 'estimator__C': 0.001}
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=0.001,penalty='l2', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.06003372681281619
macro f1 score : 0.11921663635861177
micro f1 score : 0.3654870398863771
hamming loss : 0.050932237607771415
Precision recall report :
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	56
1	0.34	0.16	0.21	129
2	0.00	0.00	0.00	28
3	0.17	0.05	0.07	22
4	0.30	0.17	0.21	18
5	0.43	0.09	0.14	35
6	0.50	0.03	0.06	30
7	0.09	0.04	0.05	79
8	0.00	0.00	0.00	8
9	0.09	0.02	0.04	45
10	0.50	0.09	0.15	11
11	0.00	0.00	0.00	40
12	0.09	0.05	0.07	115
13	0.00	0.00	0.00	18
14	0.00	0.00	0.00	9
15	0.00	0.00	0.00	15
16	0.00	0.00	0.00	13
17	0.22	0.29	0.25	368
18	0.40	0.07	0.12	27
19	0.12	0.02	0.04	97
20	0.33	0.43	0.37	551
21	0.00	0.00	0.00	41
22	0.10	0.05	0.06	85
23	0.33	0.02	0.04	42
24	0.16	0.05	0.07	83
25	0.17	0.13	0.15	159
26	0.65	0.20	0.30	112
27	0.00	0.00	0.00	11
28	0.30	0.48	0.37	596
29	0.48	0.24	0.32	190
30	0.32	0.23	0.27	83
31	0.00	0.00	0.00	12
32	0.57	0.15	0.24	26
33	0.00	0.00	0.00	64
34	0.18	0.08	0.11	25
35	0.00	0.00	0.00	29
36	0.29	0.28	0.29	92
37	0.32	0.11	0.16	172
38	0.24	0.11	0.15	136
39	0.00	0.00	0.00	32
40	0.20	0.03	0.04	40
41	0.00	0.00	0.00	5
42	0.14	0.02	0.04	93
43	0.60	0.70	0.65	1155
44	0.22	0.11	0.15	117
45	0.27	0.40	0.33	145
46	0.00	0.00	0.00	5
47	0.20	0.08	0.11	129
48	0.14	0.03	0.05	36
49	0.33	0.05	0.08	44
50	0.00	0.00	0.00	28
51	0.00	0.00	0.00	51
52	0.50	0.20	0.28	395
53	0.00	0.00	0.00	65
54	0.00	0.00	0.00	15
55	0.00	0.00	0.00	37
56	0.35	0.42	0.38	507
57	0.43	0.56	0.48	587
58	0.32	0.12	0.18	148
59	0.26	0.05	0.09	173
60	0.26	0.09	0.13	66
61	0.00	0.00	0.00	51
62	0.12	0.02	0.03	66
63	0.00	0.00	0.00	39
64	0.00	0.00	0.00	8
65	0.23	0.35	0.27	228
66	0.00	0.00	0.00	27
67	0.30	0.08	0.12	119
68	0.55	0.66	0.60	911
69	0.25	0.07	0.11	14
70	0.00	0.00	0.00	13
micro avg	0.39	0.34	0.37	9021
macro avg	0.19	0.11	0.12	9021
weighted avg	0.35	0.34	0.32	9021

```
weighted avg       0.55      0.54      0.52      9021
 samples avg       0.36      0.37      0.31      9021
```

OBSERVATIONS

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with TFIDF Vectorizer and BiGrams")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Log Loss",0.071, 0.33, 0.045])
x.add_row(["SGD With Hinge Loss",0.064, 0.30, 0.045])
x.add_row(["Logistic Regression",0.060, 0.36, 0.050 ])

print(x)
```

```
Comparison of models with TFIDF Vectorizer and BiGrams
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
|  SGD With Log Loss  |  0.071   |      0.33      |    0.045     |
| SGD With Hinge Loss |  0.064   |      0.3       |    0.045     |
| Logistic Regression |   0.06   |      0.36      |     0.05     |
+---------------------+----------+----------------+--------------+
```

4.6 DATA FEATURIZATION USING TRIGRAMS

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2",
                             tokenizer = lambda x: x.split(),ngram_range=(3,3))
x_train_multilabel = vectorizer.fit_transform(x_train)
x_test_multilabel = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
print("Dimensions of train data X:",x_train_multilabel.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_multilabel.shape,"Y:",multilabel_y_test.shape)
```

4.6.1 Applying Logistic Regression with SGD with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:02:37.271240
The best hyper parameters are  {'estimator__alpha': 0.001, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.037436762225969644
macro f1 score : 0.07069721806950237
micro f1 score : 0.24762726488352024
hamming loss : 0.049706671733605684
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.16      0.04      0.06       129
           2       0.00      0.00      0.00        28
           3       0.25      0.05      0.08        22
           4       0.00      0.00      0.00        18
           5       0.20      0.03      0.05        35
           6       0.10      0.03      0.05        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.12      0.04      0.07        45
          10       0.20      0.09      0.13        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.17      0.12      0.14       368
          18       0.33      0.04      0.07        27
          19       0.08      0.01      0.02        97
          20       0.32      0.20      0.25       551
          21       0.50      0.02      0.05        41
          22       0.14      0.02      0.04        85
          23       0.12      0.02      0.04        42
          24       0.07      0.04      0.05        83
          25       0.11      0.02      0.03       159
          26       0.41      0.12      0.19       112
          27       0.00      0.00      0.00        11
          28       0.28      0.22      0.25       596
          29       0.36      0.11      0.16       190
          30       0.25      0.14      0.18        83
          31       0.00      0.00      0.00        12
          32       0.50      0.12      0.19        26
          33       0.00      0.00      0.00        64
          34       0.20      0.08      0.11        25
          35       0.00      0.00      0.00        29
          36       0.32      0.07      0.11        92
          37       0.17      0.05      0.07       172
          38       0.22      0.04      0.06       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.14      0.04      0.07        93
          43       0.55      0.49      0.52      1155
          44       0.12      0.02      0.03       117
          45       0.20      0.08      0.11       145
          46       0.00      0.00      0.00         5
          47       0.15      0.05      0.07       129
          48       0.00      0.00      0.00        36
          49       0.25      0.05      0.08        44
          50       0.00      0.00      0.00        28
          51       0.12      0.02      0.03        51
          52       0.28      0.12      0.17       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.29      0.19      0.23       507
          57       0.35      0.32      0.33       587
          58       0.20      0.03      0.05       148
          59       0.12      0.06      0.08       173
          60       0.38      0.05      0.08        66
          61       0.06      0.02      0.03        51
          62       0.08      0.02      0.03        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.22      0.08      0.12       228
          66       0.00      0.00      0.00        27
          67       0.24      0.07      0.11       119
          68       0.50      0.41      0.45       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.35      0.19      0.25      9021
   macro avg       0.14      0.05      0.07      9021
weighted avg       0.29      0.19      0.22      9021
```

```
weighted avg       0.29      0.19      0.22      9021
 samples avg       0.28      0.22      0.20      9021
```

4.6.2 SGD with Hinge loss with OneVsRest Classifier

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_multilabel, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.038448566610455315
macro f1 score : 0.0657016742296203
micro f1 score : 0.22986411599483794
hamming loss : 0.04819134028454029
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.18      0.03      0.05       129
           2       0.00      0.00      0.00        28
           3       0.20      0.05      0.07        22
           4       0.00      0.00      0.00        18
           5       0.00      0.00      0.00        35
           6       0.14      0.03      0.05        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.05      0.02      0.03        45
          10       0.25      0.09      0.13        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.20      0.07      0.10       368
          18       1.00      0.04      0.07        27
          19       0.09      0.01      0.02        97
          20       0.33      0.15      0.21       551
          21       0.25      0.02      0.04        41
          22       0.11      0.02      0.04        85
          23       0.33      0.02      0.04        42
          24       0.05      0.01      0.02        83
          25       0.07      0.01      0.01       159
          26       0.41      0.11      0.17       112
          27       0.00      0.00      0.00        11
          28       0.26      0.15      0.19       596
          29       0.29      0.08      0.13       190
          30       0.33      0.17      0.22        83
          31       0.00      0.00      0.00        12
          32       0.60      0.12      0.19        26
          33       0.08      0.03      0.04        64
          34       0.22      0.08      0.12        25
          35       0.25      0.03      0.06        29
          36       0.30      0.07      0.11        92
          37       0.20      0.03      0.05       172
          38       0.24      0.04      0.07       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.11      0.02      0.04        93
          43       0.54      0.49      0.52      1155
          44       0.13      0.02      0.03       117
          45       0.11      0.03      0.04       145
          46       0.00      0.00      0.00         5
          47       0.16      0.05      0.07       129
          48       0.08      0.03      0.04        36
          49       0.22      0.05      0.08        44
          50       0.00      0.00      0.00        28
          51       0.12      0.04      0.06        51
          52       0.29      0.08      0.13       395
          53       0.12      0.03      0.05        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.29      0.16      0.21       507
          57       0.35      0.24      0.28       587
          58       0.14      0.02      0.04       148
          59       0.21      0.08      0.11       173
          60       0.18      0.05      0.07        66
          61       0.00      0.00      0.00        51
          62       0.04      0.02      0.02        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.21      0.05      0.08       228
          66       0.00      0.00      0.00        27
          67       0.22      0.07      0.10       119
          68       0.48      0.38      0.43       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.36      0.17      0.23      9021
   macro avg       0.15      0.05      0.07      9021
weighted avg       0.28      0.17      0.20      9021
```

```
weighted avg       0.28      0.17      0.20      9021
 samples avg       0.28      0.20      0.19      9021
```

4.6.3 Logistic Regression with OneVsRest Classifier

```
classifier = OneVsRestClassifier(LogisticRegression(C=0.001,penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_multilabel, multilabel_y_train)
predictions = classifier.predict(x_test_multilabel)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.027655986509274873
macro f1 score : 0.08442745052681629
micro f1 score : 0.26443788097781684
hamming loss : 0.0556017385934494
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.12      0.05      0.07       129
           2       0.00      0.00      0.00        28
           3       0.14      0.05      0.07        22
           4       0.00      0.00      0.00        18
           5       0.09      0.03      0.04        35
           6       0.06      0.03      0.04        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.07      0.04      0.05        45
          10       0.12      0.09      0.11        11
          11       0.00      0.00      0.00        40
          12       0.03      0.01      0.01       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.18      0.20      0.19       368
          18       0.33      0.07      0.12        27
          19       0.04      0.01      0.02        97
          20       0.29      0.26      0.27       551
          21       0.25      0.02      0.04        41
          22       0.08      0.02      0.04        85
          23       0.07      0.02      0.04        42
          24       0.03      0.04      0.03        83
          25       0.18      0.08      0.11       159
          26       0.40      0.15      0.22       112
          27       0.00      0.00      0.00        11
          28       0.27      0.33      0.29       596
          29       0.30      0.15      0.20       190
          30       0.29      0.20      0.24        83
          31       0.00      0.00      0.00        12
          32       0.57      0.15      0.24        26
          33       0.12      0.09      0.10        64
          34       0.20      0.08      0.11        25
          35       0.08      0.03      0.05        29
          36       0.27      0.10      0.14        92
          37       0.12      0.08      0.09       172
          38       0.14      0.04      0.07       136
          39       0.00      0.00      0.00        32
          40       0.08      0.03      0.04        40
          41       0.00      0.00      0.00         5
          42       0.05      0.04      0.05        93
          43       0.54      0.54      0.54      1155
          44       0.08      0.03      0.04       117
          45       0.18      0.14      0.16       145
          46       0.00      0.00      0.00         5
          47       0.16      0.08      0.11       129
          48       0.00      0.00      0.00        36
          49       0.17      0.05      0.07        44
          50       0.07      0.07      0.07        28
          51       0.09      0.02      0.03        51
          52       0.24      0.15      0.19       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.27      0.25      0.26       507
          57       0.34      0.37      0.35       587
          58       0.16      0.05      0.07       148
          59       0.12      0.12      0.12       173
          60       0.19      0.05      0.07        66
          61       0.04      0.04      0.04        51
          62       0.03      0.02      0.02        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.20      0.14      0.17       228
          66       0.00      0.00      0.00        27
          67       0.21      0.07      0.10       119
          68       0.48      0.45      0.46       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.31      0.23      0.26      9021
   macro avg       0.12      0.07      0.08      9021
weighted avg       0.27      0.23      0.24      9021
```

```
weighted avg       0.27      0.25      0.24      9021
 samples avg       0.27      0.26      0.22      9021
```

OBSERVATIONS

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with TFIDF Vectorizer and Trigrams")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Log Loss",0.037, 0.24, 0.049])
x.add_row(["SGD With Hinge Loss",0.038, 0.22, 0.048 ])
x.add_row(["Logistic Regression",0.027, 0.26, 0.055 ])

print(x)
```

4.7 DATA FEATURIZATION USING W2V

```
!pip install gensim
```

```
Collecting gensim
  Using cached https://files.pythonhosted.org/packages/ec/db/d0c6edd6e7211e7c47404034ed9dd71032a0a77c6ae8835505f1bd1
Collecting six>=1.5.0 (from gensim)
  Using cached https://files.pythonhosted.org/packages/73/fb/00a976f728d0d1fecfe898238ce23f502a721c0ac0ecfedb80e0d88
Collecting smart-open>=1.8.1 (from gensim)
Collecting numpy<=1.16.1,>=1.11.3 (from gensim)
  Using cached https://files.pythonhosted.org/packages/e0/b5/63b79fe426433fa1cd110eb04a94ec0c6967e56e5f57c98caf455a5
Collecting scipy>=0.18.1 (from gensim)
  Using cached https://files.pythonhosted.org/packages/1d/f6/7c16d60aeb3694e5611976cb4f1eaf1c6b7f1e7c55771d691013405
Collecting requests (from smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/51/bd/23c926cd341ea6b7dd0b2a00aba99ae0f828be89d72b2190f27c11d
Collecting boto>=2.32 (from smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/23/10/c0b78c27298029e4454a472a1919bde20cb182dab1662cec7f2ca1d
Collecting bz2file (from smart-open>=1.8.1->gensim)
Collecting boto3 (from smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/c3/16/ba4d09feba1fec565f6fa3d5904540c35cf5add21d8ea1b8c7add88
Collecting idna<2.9,>=2.5 (from requests->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/14/2c/cd551d81dbe15200be1cf41cd03869a46fe7226e7450af7a6545bfc
Collecting chardet<3.1.0,>=3.0.2 (from requests->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca55ec7510b22e4c51f14098
Collecting certifi>=2017.4.17 (from requests->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/18/b0/8146a4f8dd402f60744fa380bc73ca47303cccf8b9190fd16a82728
Collecting urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 (from requests->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/e0/da/55f51ea951e1b7c63a579c09dd7db825bb730ec1fe9c0180fc77bfb
Collecting jmespath<1.0.0,>=0.7.1 (from boto3->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/83/94/7179c3832a6d45b266ddb2aac329e101367fbdb11f425f13771d27f
Collecting s3transfer<0.3.0,>=0.2.0 (from boto3->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/16/8a/1fc3dba0c4923c2a76e1ff0d52b305c44606da63f718d14d3231e21
Collecting botocore<1.13.0,>=1.12.236 (from boto3->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/52/2f/1812b5d106b646a8ec86f4066213ad7281006e59157ddafd0f15683
Collecting futures<4.0.0,>=2.2.0; python_version == "2.6" or python_version == "2.7" (from s3transfer<0.3.0,>=0.2.0-
  Using cached https://files.pythonhosted.org/packages/d8/a6/f46ae3f1da0cd4361c344888f59ec2f5785e69c872e175a748ef607
Collecting python-dateutil<3.0.0,>=2.1; python_version >= "2.7" (from botocore<1.13.0,>=1.12.236->boto3->smart-open>
  Using cached https://files.pythonhosted.org/packages/41/17/c62faccbfbd163c7f57f3844689e3a78bae1f403648a6afb1d0866d
Collecting docutils<0.16,>=0.10 (from botocore<1.13.0,>=1.12.236->boto3->smart-open>=1.8.1->gensim)
  Using cached https://files.pythonhosted.org/packages/3a/dc/bf2b15d1fa15a6f7a9e77a61b74ecbbae7258558fcda8ffc9a6638a
Installing collected packages: six, idna, chardet, certifi, urllib3, requests, boto, bz2file, jmespath, futures, pyt
Successfully installed boto-2.49.0 boto3-1.9.236 botocore-1.12.236 bz2file-0.98 certifi-2019.9.11 chardet-3.0.4 docu
```

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from tqdm import tqdm
```

```
word2vec_model=KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True)
word2vec_words = list(word2vec_model.wv.vocab)
```

```
def vectorize_w2v(dataset, word2vec_model, word2vec_words):
    word2vec_corpus=[]
    for sentence in dataset:
        word2vec_corpus.append(sentence.split())

    # Creating average Word2Vec model by computing the average word2vec for each review.
    sent_vectors = []; #The average word2vec for each sentence/review will be stored in this list
    for sentence in tqdm(word2vec_corpus): #For each review
        sent_vec = np.zeros(300) #300 dimensional array, where all elements are zero. This is used to add word vectors and find th
        count_words =0; #This will store the count of the words with a valid vector in each review text
        for word in sentence: #For each word in a given review.
            if word in word2vec_words:
                word_vectors = word2vec_model.wv[word] #Creating a vector(numpy array of 300 dimensions) for each word.
                sent_vec += word_vectors
                count_words += 1
        if count_words != 0:
            sent_vec /= count_words
        sent_vectors.append(sent_vec)
    #print("\nThe length of the sentence vectors :",len(sent_vectors))
    #print("\nSize of each vector : ",len(sent_vectors[0]))
    sent_vectors = np.array(sent_vectors)
    return sent_vectors


X_train_vectors = vectorize_w2v(x_train, word2vec_model, word2vec_words)
X_test_vectors = vectorize_w2v(x_test, word2vec_model, word2vec_words)

import pickle
with open('x_train_W2V.pkl', 'wb') as file:
    pickle.dump(X_train_vectors, file)

with open('x_test_W2V.pkl', 'wb') as file:
    pickle.dump(X_test_vectors, file)
```

```
100%|██████████████████████████████| 11816/11816 [10:15:46<00:00,  3.13s/it]
100%|██████████████████████████████| 2965/2965 [2:40:28<00:00,  3.25s/it]
```

4.7.1 W2V with SGD with Hinge Loss

```
with open('X_train_W2V.pkl', 'rb') as file:
    x_train = pickle.load(file)

with open('X_test_W2V.pkl', 'rb') as file:
    x_test = pickle.load(file)
```

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge')), param_grid=param_grid, n_jobs=-1)
gsv1.fit(x_train, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
Time taken to run this cell : 0:01:36.657924
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l1'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1'), n_jobs=-1)
classifier.fit(x_train, multilabel_y_train)
predictions = classifier.predict(x_test)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.023946037099494097
macro f1 score : 0.025101723729588982
micro f1 score : 0.1124320652173913
hamming loss : 0.023980452214612657
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.00      0.00      0.00        52
           2       0.50      0.03      0.06        29
           3       0.00      0.00      0.00       100
           4       0.00      0.00      0.00        28
           5       0.00      0.00      0.00         3
           6       0.00      0.00      0.00        19
           7       0.00      0.00      0.00        50
           8       0.00      0.00      0.00        30
           9       0.00      0.00      0.00        20
          10       0.00      0.00      0.00        59
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         5
          13       0.29      0.04      0.08        45
          14       1.00      0.12      0.22         8
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         4
          17       0.00      0.00      0.00        36
          18       0.00      0.00      0.00        22
          19       0.00      0.00      0.00        93
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        15
          22       0.00      0.00      0.00         9
          23       0.00      0.00      0.00         4
          24       0.00      0.00      0.00        11
          25       0.00      0.00      0.00         2
          26       0.00      0.00      0.00        11
          27       0.33      0.02      0.03        63
          28       0.39      0.05      0.09       323
          29       0.00      0.00      0.00        15
          30       0.00      0.00      0.00        12
          31       0.00      0.00      0.00         5
          32       1.00      0.02      0.04        92
          33       0.00      0.00      0.00        48
          34       0.34      0.03      0.05       503
          35       0.00      0.00      0.00         7
          36       0.00      0.00      0.00        34
          37       0.00      0.00      0.00         5
          38       0.00      0.00      0.00        80
          39       0.00      0.00      0.00         6
          40       0.00      0.00      0.00        36
          41       0.00      0.00      0.00         1
          42       0.00      0.00      0.00        82
          43       0.00      0.00      0.00        88
          44       0.00      0.00      0.00        71
          45       0.00      0.00      0.00        21
          46       0.00      0.00      0.00       169
          47       0.26      0.49      0.34        37
          48       0.00      0.00      0.00        75
          49       0.00      0.00      0.00         3
          50       0.00      0.00      0.00         8
          51       0.17      0.04      0.07        24
          52       0.00      0.00      0.00         6
          53       0.00      0.00      0.00        10
          54       0.00      0.00      0.00        11
          55       0.29      0.01      0.01       394
          56       0.18      0.06      0.09       202
          57       0.00      0.00      0.00         3
          58       0.00      0.00      0.00        42
          59       1.00      0.01      0.01       190
          60       0.00      0.00      0.00        12
          61       0.00      0.00      0.00        71
          62       0.00      0.00      0.00        12
          63       1.00      0.07      0.12        15
          64       0.29      0.18      0.22        11
          65       1.00      0.02      0.03        59
          66       0.00      0.00      0.00        30
          67       0.00      0.00      0.00         9
          68       0.00      0.00      0.00         9
          69       0.00      0.00      0.00        29
          70       0.00      0.00      0.00        18
          71       0.29      0.03      0.05        74
          72       0.00      0.00      0.00        40
          73       0.10      0.05      0.06       132
          74       0.00      0.00      0.00         8
```

74	0.00	0.00	0.00	8
75	0.14	0.02	0.04	128
76	0.00	0.00	0.00	5
77	0.00	0.00	0.00	27
78	0.00	0.00	0.00	6
79	0.18	0.06	0.09	34
80	0.00	0.00	0.00	5
81	0.22	0.10	0.14	60
82	0.00	0.00	0.00	33
83	0.00	0.00	0.00	28
84	0.00	0.00	0.00	1
85	0.00	0.00	0.00	487
86	0.42	0.51	0.46	668
87	0.00	0.00	0.00	11
88	0.00	0.00	0.00	106
89	0.00	0.00	0.00	145
90	0.00	0.00	0.00	13
91	0.00	0.00	0.00	132
92	0.00	0.00	0.00	5
93	0.00	0.00	0.00	23
94	0.29	0.04	0.07	106
95	0.00	0.00	0.00	11
96	0.00	0.00	0.00	25
97	0.00	0.00	0.00	44
98	0.00	0.00	0.00	7
99	0.00	0.00	0.00	21
100	0.00	0.00	0.00	47
101	0.00	0.00	0.00	4
102	0.22	0.62	0.32	124
103	0.31	0.02	0.03	271
104	0.00	0.00	0.00	7
105	0.50	0.02	0.03	58
106	0.00	0.00	0.00	13
107	0.00	0.00	0.00	2
108	0.00	0.00	0.00	7
109	0.00	0.00	0.00	35
110	0.00	0.00	0.00	9
111	0.00	0.00	0.00	26
112	0.00	0.00	0.00	141
113	0.38	0.01	0.02	366
114	0.56	0.04	0.07	285
115	0.00	0.00	0.00	302
116	0.00	0.00	0.00	113
117	0.00	0.00	0.00	35
118	0.00	0.00	0.00	69
119	0.50	0.02	0.04	104
120	0.40	0.13	0.20	61
121	0.00	0.00	0.00	5
122	0.00	0.00	0.00	37
123	0.00	0.00	0.00	14
124	0.00	0.00	0.00	37
125	0.00	0.00	0.00	29
126	0.00	0.00	0.00	12
127	0.00	0.00	0.00	27
128	0.00	0.00	0.00	4
129	0.00	0.00	0.00	4
130	0.00	0.00	0.00	53
131	0.25	0.02	0.03	175
132	0.00	0.00	0.00	0
133	0.00	0.00	0.00	27
134	0.00	0.00	0.00	24
135	0.00	0.00	0.00	95
136	0.00	0.00	0.00	11
137	0.00	0.00	0.00	33
138	0.35	0.08	0.13	190
139	0.42	0.05	0.08	283
140	0.54	0.13	0.21	628
141	0.00	0.00	0.00	7
142	0.00	0.00	0.00	23
143	0.00	0.00	0.00	2
144	0.50	0.08	0.14	12
145	0.00	0.00	0.00	0
146	0.00	0.00	0.00	13
micro avg	0.34	0.07	0.11	9818
macro avg	0.10	0.02	0.03	9818
weighted avg	0.24	0.07	0.07	9818
samples avg	0.16	0.07	0.08	9818

4.7.2 W2V with SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log')), param_grid=param_grid, n_jobs=-1)
gsv.fit(x_train, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:01:49.381613
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2'), n_jobs=-1)
classifier.fit(x_train, multilabel_y_train)
predictions = classifier.predict(x_test)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.008768971332209106
macro f1 score : 0.01428218407178018
micro f1 score : 0.08245445829338448
hamming loss : 0.024152527790205457
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.00      0.00      0.00        52
           2       0.50      0.03      0.06        29
           3       0.00      0.00      0.00       100
           4       0.00      0.00      0.00        28
           5       0.00      0.00      0.00         3
           6       0.00      0.00      0.00        19
           7       0.00      0.00      0.00        50
           8       0.00      0.00      0.00        30
           9       0.00      0.00      0.00        20
          10       0.00      0.00      0.00        59
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         5
          13       0.00      0.00      0.00        45
          14       0.00      0.00      0.00         8
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         4
          17       0.00      0.00      0.00        36
          18       0.00      0.00      0.00        22
          19       0.00      0.00      0.00        93
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        15
          22       0.00      0.00      0.00         9
          23       0.00      0.00      0.00         4
          24       0.00      0.00      0.00        11
          25       0.00      0.00      0.00         2
          26       0.00      0.00      0.00        11
          27       0.00      0.00      0.00        63
          28       0.31      0.07      0.12       323
          29       0.00      0.00      0.00        15
          30       0.00      0.00      0.00        12
          31       0.00      0.00      0.00         5
          32       0.00      0.00      0.00        92
          33       0.00      0.00      0.00        48
          34       0.30      0.54      0.39       503
          35       0.00      0.00      0.00         7
          36       0.00      0.00      0.00        34
          37       0.00      0.00      0.00         5
          38       0.00      0.00      0.00        80
          39       0.00      0.00      0.00         6
          40       0.00      0.00      0.00        36
          41       0.00      0.00      0.00         1
          42       0.00      0.00      0.00        82
          43       0.00      0.00      0.00        88
          44       0.00      0.00      0.00        71
          45       0.00      0.00      0.00        21
          46       0.35      0.05      0.09       169
          47       1.00      0.03      0.05        37
          48       0.00      0.00      0.00        75
          49       0.00      0.00      0.00         3
          50       0.00      0.00      0.00         8
          51       0.00      0.00      0.00        24
          52       0.00      0.00      0.00         6
          53       0.00      0.00      0.00        10
          54       0.00      0.00      0.00        11
          55       0.00      0.00      0.00       394
          56       0.17      0.31      0.22       202
          57       0.00      0.00      0.00         3
          58       0.14      0.10      0.11        42
          59       0.33      0.01      0.02       190
          60       0.00      0.00      0.00        12
          61       0.00      0.00      0.00        71
          62       0.00      0.00      0.00        12
          63       0.00      0.00      0.00        15
          64       1.00      0.09      0.17        11
          65       0.00      0.00      0.00        59
          66       0.00      0.00      0.00        30
          67       0.00      0.00      0.00         9
          68       0.00      0.00      0.00         9
          69       0.00      0.00      0.00        29
          70       0.00      0.00      0.00        18
          71       0.00      0.00      0.00        74
          72       0.00      0.00      0.00        40
          73       0.00      0.00      0.00       132
          74       0.00      0.00      0.00         8
```

74	0.00	0.00	0.00	8
75	0.17	0.01	0.01	128
76	0.00	0.00	0.00	5
77	0.00	0.00	0.00	27
78	0.00	0.00	0.00	6
79	0.00	0.00	0.00	34
80	0.00	0.00	0.00	5
81	0.00	0.00	0.00	60
82	0.00	0.00	0.00	33
83	0.00	0.00	0.00	28
84	0.00	0.00	0.00	1
85	0.64	0.02	0.04	487
86	0.37	0.01	0.02	668
87	0.00	0.00	0.00	11
88	0.00	0.00	0.00	106
89	0.39	0.09	0.15	145
90	0.00	0.00	0.00	13
91	0.00	0.00	0.00	132
92	0.00	0.00	0.00	5
93	0.00	0.00	0.00	23
94	0.50	0.01	0.02	106
95	0.00	0.00	0.00	11
96	0.00	0.00	0.00	25
97	0.00	0.00	0.00	44
98	0.00	0.00	0.00	7
99	0.00	0.00	0.00	21
100	0.00	0.00	0.00	47
101	0.00	0.00	0.00	4
102	0.94	0.12	0.21	124
103	0.00	0.00	0.00	271
104	0.00	0.00	0.00	7
105	0.50	0.03	0.06	58
106	0.00	0.00	0.00	13
107	0.00	0.00	0.00	2
108	0.00	0.00	0.00	7
109	0.00	0.00	0.00	35
110	0.00	0.00	0.00	9
111	0.00	0.00	0.00	26
112	0.00	0.00	0.00	141
113	0.30	0.02	0.04	366
114	0.50	0.02	0.03	285
115	0.00	0.00	0.00	302
116	0.25	0.01	0.02	113
117	0.00	0.00	0.00	35
118	0.00	0.00	0.00	69
119	0.00	0.00	0.00	104
120	0.00	0.00	0.00	61
121	0.00	0.00	0.00	5
122	0.00	0.00	0.00	37
123	0.00	0.00	0.00	14
124	0.00	0.00	0.00	37
125	0.00	0.00	0.00	29
126	0.00	0.00	0.00	12
127	0.00	0.00	0.00	27
128	0.00	0.00	0.00	4
129	0.00	0.00	0.00	4
130	0.00	0.00	0.00	53
131	0.00	0.00	0.00	175
132	0.00	0.00	0.00	0
133	0.00	0.00	0.00	27
134	0.00	0.00	0.00	24
135	0.00	0.00	0.00	95
136	0.00	0.00	0.00	11
137	0.00	0.00	0.00	33
138	0.43	0.08	0.14	190
139	0.40	0.07	0.12	283
140	0.00	0.00	0.00	628
141	0.00	0.00	0.00	7
142	0.00	0.00	0.00	23
143	0.00	0.00	0.00	2
144	0.00	0.00	0.00	12
145	0.00	0.00	0.00	0
146	0.00	0.00	0.00	13
micro avg	0.29	0.05	0.08	9818
macro avg	0.06	0.01	0.01	9818
weighted avg	0.18	0.05	0.05	9818
samples avg	0.12	0.04	0.05	9818

4.7.3 W2V with Logistic Regression

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__C=[0.001,0.01,1,100], estimator__penalty=["l1","l2"])

gsv2 = GridSearchCV(OneVsRestClassifier(estimator= LogisticRegression()), param_grid=param_grid, n_jobs=-1)
gsv2.fit(x_train, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv2.best_params_)
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l2'), n_jobs=-1)
classifier.fit(x_train, multilabel_y_train)
predictions = classifier.predict(x_test)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.02833052276559865
macro f1 score : 0.03478803226257032
micro f1 score : 0.10062893081761005
hamming loss : 0.022966353489119087
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         4
           1       0.00      0.00      0.00        52
           2       0.25      0.03      0.06        29
           3       0.00      0.00      0.00       100
           4       0.00      0.00      0.00        28
           5       0.00      0.00      0.00         3
           6       0.00      0.00      0.00        19
           7       0.33      0.02      0.04        50
           8       0.00      0.00      0.00        30
           9       0.00      0.00      0.00        20
          10       0.00      0.00      0.00        59
          11       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         5
          13       0.33      0.04      0.08        45
          14       0.00      0.00      0.00         8
          15       0.00      0.00      0.00         3
          16       0.00      0.00      0.00         4
          17       0.00      0.00      0.00        36
          18       0.00      0.00      0.00        22
          19       0.00      0.00      0.00        93
          20       0.00      0.00      0.00         3
          21       0.00      0.00      0.00        15
          22       0.00      0.00      0.00         9
          23       0.00      0.00      0.00         4
          24       0.00      0.00      0.00        11
          25       0.00      0.00      0.00         2
          26       0.00      0.00      0.00        11
          27       0.50      0.02      0.03        63
          28       0.38      0.03      0.06       323
          29       1.00      0.07      0.12        15
          30       0.00      0.00      0.00        12
          31       0.00      0.00      0.00         5
          32       0.50      0.01      0.02        92
          33       0.00      0.00      0.00        48
          34       0.46      0.05      0.08       503
          35       0.00      0.00      0.00         7
          36       0.00      0.00      0.00        34
          37       0.00      0.00      0.00         5
          38       0.00      0.00      0.00        80
          39       0.00      0.00      0.00         6
          40       0.00      0.00      0.00        36
          41       0.00      0.00      0.00         1
          42       0.00      0.00      0.00        82
          43       0.00      0.00      0.00        88
          44       0.00      0.00      0.00        71
          45       0.00      0.00      0.00        21
          46       0.29      0.04      0.07       169
          47       0.41      0.19      0.26        37
          48       0.29      0.03      0.05        75
          49       0.00      0.00      0.00         3
          50       0.00      0.00      0.00         8
          51       0.33      0.04      0.07        24
          52       0.00      0.00      0.00         6
          53       0.00      0.00      0.00        10
          54       0.00      0.00      0.00        11
          55       0.45      0.03      0.05       394
          56       0.00      0.00      0.00       202
          57       0.00      0.00      0.00         3
          58       0.00      0.00      0.00        42
          59       0.38      0.09      0.14       190
          60       0.25      0.17      0.20        12
          61       0.09      0.01      0.02        71
          62       0.00      0.00      0.00        12
          63       0.40      0.13      0.20        15
          64       0.50      0.09      0.15        11
          65       0.17      0.02      0.03        59
          66       0.00      0.00      0.00        30
          67       1.00      0.11      0.20         9
          68       0.00      0.00      0.00         9
          69       0.50      0.03      0.06        29
          70       0.00      0.00      0.00        18
          71       0.36      0.05      0.09        74
          72       0.00      0.00      0.00        40
          73       0.00      0.00      0.00       132
          74       0.00      0.00      0.00         8
```

74	0.00	0.00	0.00	8
75	0.17	0.01	0.01	128
76	0.00	0.00	0.00	5
77	0.00	0.00	0.00	27
78	0.00	0.00	0.00	6
79	0.00	0.00	0.00	34
80	0.00	0.00	0.00	5
81	0.00	0.00	0.00	60
82	0.00	0.00	0.00	33
83	0.50	0.04	0.07	28
84	0.00	0.00	0.00	1
85	0.45	0.10	0.17	487
86	0.50	0.19	0.27	668
87	0.00	0.00	0.00	11
88	0.00	0.00	0.00	106
89	0.43	0.12	0.19	145
90	0.00	0.00	0.00	13
91	0.33	0.08	0.12	132
92	0.00	0.00	0.00	5
93	0.00	0.00	0.00	23
94	0.44	0.04	0.07	106
95	0.00	0.00	0.00	11
96	0.33	0.04	0.07	25
97	0.00	0.00	0.00	44
98	0.00	0.00	0.00	7
99	0.25	0.05	0.08	21
100	0.33	0.02	0.04	47
101	0.00	0.00	0.00	4
102	0.74	0.23	0.36	124
103	0.33	0.01	0.03	271
104	0.00	0.00	0.00	7
105	0.50	0.03	0.06	58
106	0.00	0.00	0.00	13
107	0.00	0.00	0.00	2
108	0.00	0.00	0.00	7
109	0.00	0.00	0.00	35
110	0.00	0.00	0.00	9
111	0.00	0.00	0.00	26
112	0.00	0.00	0.00	141
113	0.37	0.03	0.05	366
114	0.50	0.12	0.19	285
115	0.13	0.01	0.01	302
116	0.00	0.00	0.00	113
117	1.00	0.06	0.11	35
118	0.17	0.01	0.03	69
119	0.67	0.02	0.04	104
120	0.33	0.10	0.15	61
121	0.00	0.00	0.00	5
122	0.00	0.00	0.00	37
123	0.00	0.00	0.00	14
124	0.00	0.00	0.00	37
125	0.00	0.00	0.00	29
126	0.00	0.00	0.00	12
127	0.00	0.00	0.00	27
128	0.00	0.00	0.00	4
129	0.00	0.00	0.00	4
130	0.00	0.00	0.00	53
131	0.23	0.02	0.03	175
132	0.00	0.00	0.00	0
133	0.50	0.04	0.07	27
134	0.00	0.00	0.00	24
135	0.00	0.00	0.00	95
136	0.00	0.00	0.00	11
137	0.00	0.00	0.00	33
138	0.38	0.09	0.14	190
139	0.40	0.06	0.10	283
140	0.53	0.19	0.28	628
141	0.00	0.00	0.00	7
142	0.00	0.00	0.00	23
143	0.00	0.00	0.00	2
144	0.50	0.17	0.25	12
145	0.00	0.00	0.00	0
146	0.00	0.00	0.00	13
micro avg	0.43	0.06	0.10	9818
macro avg	0.14	0.02	0.03	9818
weighted avg	0.30	0.06	0.09	9818
samples avg	0.12	0.07	0.08	9818

OBSERVATIONS

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with w2v")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.023, 0.11, 0.023])
x.add_row(["SGD With Log Loss",0.008, 0.08, 0.024 ])
x.add_row(["Logistic Regression",0.028, 0.10, 0.022 ])

print(x)
```

```
Comparison of models with w2v
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.023   |      0.11      |    0.023     |
|  SGD With Log Loss  |  0.008   |      0.08      |    0.024     |
| Logistic Regression |  0.028   |      0.1       |    0.022     |
+---------------------+----------+----------------+--------------+
```

4.8 Char-3-Gram

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(3, 3))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

```
print("Dimensions of train data X:",x_train_3char.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_3char.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 10893) Y : (11816, 71)
Dimensions of test data X: (2965, 10893) Y: (2965, 71)
```

4.8.1 Char-3-Gram with SGD with Hinge Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced' )), param_grid=param_grid, n
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:18:38.530330
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l1'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.00001, penalty='l1',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.027655986509274873
macro f1 score : 0.1755884081265495
micro f1 score : 0.3315380011968881
hamming loss : 0.05836638719331164
Precision recall report :
              precision    recall  f1-score   support

           0       0.06      0.04      0.05        56
           1       0.25      0.23      0.24       129
           2       0.20      0.18      0.19        28
           3       0.15      0.14      0.14        22
           4       0.24      0.39      0.30        18
           5       0.12      0.11      0.12        35
           6       0.18      0.17      0.17        30
           7       0.08      0.08      0.08        79
           8       0.00      0.00      0.00         8
           9       0.13      0.11      0.12        45
          10       0.29      0.45      0.36        11
          11       0.00      0.00      0.00        40
          12       0.18      0.13      0.15       115
          13       0.16      0.17      0.16        18
          14       0.19      0.33      0.24         9
          15       0.07      0.07      0.07        15
          16       0.04      0.08      0.05        13
          17       0.21      0.24      0.22       368
          18       0.12      0.07      0.09        27
          19       0.10      0.08      0.09        97
          20       0.31      0.37      0.33       551
          21       0.03      0.02      0.03        41
          22       0.13      0.11      0.12        85
          23       0.12      0.10      0.11        42
          24       0.09      0.08      0.09        83
          25       0.20      0.16      0.18       159
          26       0.44      0.30      0.36       112
          27       0.05      0.18      0.08        11
          28       0.30      0.37      0.33       596
          29       0.33      0.31      0.32       190
          30       0.32      0.40      0.35        83
          31       0.06      0.17      0.09        12
          32       0.24      0.31      0.27        26
          33       0.11      0.11      0.11        64
          34       0.14      0.20      0.16        25
          35       0.05      0.03      0.04        29
          36       0.30      0.32      0.31        92
          37       0.16      0.12      0.13       172
          38       0.22      0.20      0.21       136
          39       0.15      0.12      0.14        32
          40       0.11      0.12      0.12        40
          41       0.00      0.00      0.00         5
          42       0.18      0.13      0.15        93
          43       0.59      0.63      0.61      1155
          44       0.19      0.15      0.17       117
          45       0.29      0.34      0.31       145
          46       0.00      0.00      0.00         5
          47       0.24      0.19      0.21       129
          48       0.17      0.17      0.17        36
          49       0.10      0.09      0.09        44
          50       0.17      0.29      0.21        28
          51       0.11      0.08      0.09        51
          52       0.32      0.36      0.34       395
          53       0.34      0.20      0.25        65
          54       0.11      0.27      0.16        15
          55       0.05      0.05      0.05        37
          56       0.33      0.38      0.35       507
          57       0.38      0.42      0.40       587
          58       0.23      0.16      0.19       148
          59       0.22      0.15      0.18       173
          60       0.27      0.20      0.23        66
          61       0.16      0.10      0.12        51
          62       0.11      0.06      0.08        66
          63       0.05      0.05      0.05        39
          64       0.00      0.00      0.00         8
          65       0.20      0.19      0.20       228
          66       0.08      0.07      0.08        27
          67       0.12      0.10      0.11       119
          68       0.53      0.62      0.57       911
          69       0.33      0.50      0.40        14
          70       0.00      0.00      0.00        13

   micro avg       0.33      0.34      0.33      9021
   macro avg       0.18      0.18      0.18      9021
weighted avg       0.32      0.34      0.32      9021
```

```
weighted avg       0.32      0.34      0.32      9021
 samples avg       0.31      0.37      0.29      9021
```

4.8.2 Char-3-Gram with SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log',class_weight='balanced')), param_grid=param_grid,verbos
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:   59.6s
[Parallel(n_jobs=-1)]: Done   4 out of  24 | elapsed:  1.5min remaining:  7.3min
[Parallel(n_jobs=-1)]: Done   7 out of  24 | elapsed:  2.6min remaining:  6.3min
[Parallel(n_jobs=-1)]: Done  10 out of  24 | elapsed:  3.2min remaining:  4.4min
[Parallel(n_jobs=-1)]: Done  13 out of  24 | elapsed:  3.8min remaining:  3.2min
[Parallel(n_jobs=-1)]: Done  16 out of  24 | elapsed:  3.9min remaining:  2.0min
[Parallel(n_jobs=-1)]: Done  19 out of  24 | elapsed:  6.1min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  22 out of  24 | elapsed:  7.8min remaining:   42.4s
[Parallel(n_jobs=-1)]: Done  24 out of  24 | elapsed:  7.9min finished
Time taken to run this cell : 0:09:01.659202
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.03102866779089376
macro f1 score : 0.211025592728005
micro f1 score : 0.36075621139974545
hamming loss : 0.06440871196826829
Precision recall report :
              precision    recall  f1-score   support

           0       0.07      0.04      0.05        56
           1       0.25      0.41      0.31       129
           2       0.32      0.29      0.30        28
           3       0.15      0.09      0.11        22
           4       0.40      0.33      0.36        18
           5       0.36      0.14      0.20        35
           6       0.27      0.13      0.18        30
           7       0.12      0.16      0.14        79
           8       0.00      0.00      0.00         8
           9       0.17      0.20      0.18        45
          10       0.42      0.45      0.43        11
          11       0.00      0.00      0.00        40
          12       0.15      0.19      0.17       115
          13       0.38      0.17      0.23        18
          14       0.33      0.22      0.27         9
          15       0.17      0.07      0.10        15
          16       0.12      0.08      0.10        13
          17       0.25      0.29      0.27       368
          18       0.33      0.07      0.12        27
          19       0.08      0.12      0.09        97
          20       0.32      0.51      0.40       551
          21       0.09      0.10      0.09        41
          22       0.12      0.25      0.16        85
          23       0.10      0.10      0.10        42
          24       0.09      0.19      0.12        83
          25       0.17      0.33      0.23       159
          26       0.27      0.48      0.34       112
          27       0.00      0.00      0.00        11
          28       0.30      0.46      0.37       596
          29       0.32      0.47      0.38       190
          30       0.29      0.58      0.39        83
          31       0.17      0.17      0.17        12
          32       0.27      0.31      0.29        26
          33       0.17      0.20      0.19        64
          34       0.19      0.32      0.24        25
          35       0.15      0.07      0.10        29
          36       0.26      0.41      0.32        92
          37       0.13      0.27      0.18       172
          38       0.19      0.30      0.23       136
          39       0.23      0.09      0.13        32
          40       0.21      0.10      0.14        40
          41       0.00      0.00      0.00         5
          42       0.15      0.19      0.17        93
          43       0.62      0.66      0.64      1155
          44       0.14      0.26      0.18       117
          45       0.18      0.53      0.26       145
          46       0.14      0.20      0.17         5
          47       0.22      0.32      0.26       129
          48       0.17      0.19      0.18        36
          49       0.13      0.09      0.11        44
          50       0.18      0.29      0.22        28
          51       0.24      0.22      0.23        51
          52       0.32      0.45      0.37       395
          53       0.26      0.18      0.21        65
          54       0.16      0.27      0.20        15
          55       0.07      0.05      0.06        37
          56       0.32      0.50      0.39       507
          57       0.43      0.50      0.46       587
          58       0.28      0.20      0.23       148
          59       0.25      0.23      0.24       173
          60       0.31      0.41      0.35        66
          61       0.17      0.12      0.14        51
          62       0.11      0.08      0.09        66
          63       0.07      0.05      0.06        39
          64       0.00      0.00      0.00         8
          65       0.19      0.38      0.25       228
          66       0.16      0.11      0.13        27
          67       0.11      0.18      0.14       119
          68       0.55      0.69      0.61       911
          69       0.44      0.50      0.47        14
          70       0.00      0.00      0.00        13

   micro avg       0.31      0.42      0.36      9021
   macro avg       0.21      0.24      0.21      9021
weighted avg       0.33      0.42      0.36      9021
```

```
weighted avg       0.55      0.42      0.36      9021
 samples avg       0.31      0.45      0.32      9021
```

4.8.3 Char-3-Gram with Logistic Regression

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__penalty=["l1","l2"], estimator__C=[0.001,0.1,1,50,100])

gsv2 = GridSearchCV(OneVsRestClassifier(estimator= LogisticRegression(class_weight='balanced')), param_grid=param_grid,verbose=12,
gsv2.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv2.best_params_)
```

```
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.03237774030354131
macro f1 score : 0.17012374569472488
micro f1 score : 0.3397477712546206
hamming loss : 0.05769660119231409
Precision recall report :
              precision    recall  f1-score   support

           0       0.08      0.04      0.05        56
           1       0.29      0.33      0.31       129
           2       0.23      0.11      0.15        28
           3       0.25      0.05      0.08        22
           4       0.50      0.17      0.25        18
           5       0.27      0.11      0.16        35
           6       0.12      0.03      0.05        30
           7       0.14      0.13      0.13        79
           8       0.00      0.00      0.00         8
           9       0.13      0.09      0.11        45
          10       0.50      0.09      0.15        11
          11       0.00      0.00      0.00        40
          12       0.16      0.15      0.15       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.33      0.07      0.11        15
          16       0.50      0.08      0.13        13
          17       0.23      0.31      0.26       368
          18       0.50      0.07      0.13        27
          19       0.08      0.07      0.08        97
          20       0.32      0.39      0.35       551
          21       0.15      0.05      0.07        41
          22       0.12      0.12      0.12        85
          23       0.18      0.10      0.12        42
          24       0.07      0.06      0.07        83
          25       0.18      0.19      0.19       159
          26       0.49      0.32      0.39       112
          27       0.00      0.00      0.00        11
          28       0.29      0.37      0.33       596
          29       0.33      0.38      0.36       190
          30       0.37      0.47      0.41        83
          31       0.14      0.08      0.11        12
          32       0.35      0.23      0.28        26
          33       0.12      0.08      0.10        64
          34       0.26      0.24      0.25        25
          35       0.14      0.03      0.06        29
          36       0.25      0.33      0.28        92
          37       0.15      0.15      0.15       172
          38       0.19      0.23      0.21       136
          39       0.25      0.06      0.10        32
          40       0.19      0.07      0.11        40
          41       0.00      0.00      0.00         5
          42       0.13      0.11      0.12        93
          43       0.58      0.62      0.60      1155
          44       0.17      0.15      0.16       117
          45       0.26      0.37      0.30       145
          46       0.00      0.00      0.00         5
          47       0.18      0.19      0.19       129
          48       0.17      0.11      0.14        36
          49       0.11      0.05      0.06        44
          50       0.30      0.21      0.25        28
          51       0.16      0.10      0.12        51
          52       0.30      0.37      0.33       395
          53       0.25      0.14      0.18        65
          54       0.08      0.07      0.07        15
          55       0.08      0.03      0.04        37
          56       0.31      0.39      0.35       507
          57       0.37      0.44      0.40       587
          58       0.22      0.20      0.21       148
          59       0.23      0.23      0.23       173
          60       0.40      0.27      0.32        66
          61       0.08      0.04      0.05        51
          62       0.11      0.09      0.10        66
          63       0.08      0.03      0.04        39
          64       0.00      0.00      0.00         8
          65       0.19      0.24      0.21       228
          66       0.50      0.04      0.07        27
          67       0.12      0.10      0.11       119
          68       0.54      0.60      0.57       911
          69       0.67      0.43      0.52        14
          70       0.00      0.00      0.00        13

   micro avg       0.33      0.35      0.34      9021
   macro avg       0.22      0.16      0.17      9021
weighted avg       0.32      0.35      0.33      9021
```

```
weighted avg       0.32      0.37      0.30      9021
 samples avg       0.32      0.37      0.30      9021
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-3-Gram")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.027, 0.33, 0.058])
x.add_row(["SGD With Log Loss",0.031, 0.36, 0.064 ])
x.add_row(["Logistic Regression",0.032, 0.19, 0.026 ])

print(x)
```

```
Comparison of models with Char-3-Gram
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.023   |      0.11      |    0.023     |
|  SGD With Log Loss  |  0.039   |      0.19      |    0.024     |
| Logistic Regression |  0.032   |      0.19      |    0.026     |
+---------------------+----------+----------------+--------------+
```

4.9 Char-4-Gram

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(4, 4))
x_train_4char = vectorizer.fit_transform(x_train)
x_test_4char = vectorizer.transform(x_test)
```

4.9.1 Char-4-Gram with SGD with Hinge Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_4char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
Time taken to run this cell : 0:17:19.747202
The best hyper parameters are  {'estimator__alpha': 0.0001, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_4char, multilabel_y_train)
predictions = classifier.predict(x_test_4char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.4816188870151771
macro f1 score : 0.464379602724942
micro f1 score : 0.4696755994358251
hamming loss : 0.25362563237774033
Precision recall report :
               precision    recall  f1-score   support

           0       0.34      0.49      0.40       487
           1       0.43      0.59      0.50       668
           2       0.43      0.59      0.49       628

   micro avg       0.40      0.56      0.47      1783
   macro avg       0.40      0.55      0.46      1783
weighted avg       0.40      0.56      0.47      1783
 samples avg       0.19      0.24      0.20      1783
```

4.9.2 Char-4-Gram with SGD with Log Loss

```
#Hyperparameter tuning
```

```
from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_4char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:16:14.865048
The best hyper parameters are  {'estimator__alpha': 1e-05, 'estimator__penalty': 'l2'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_4char, multilabel_y_train)
predictions = classifier.predict(x_test_4char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.48026981450252954
macro f1 score : 0.44761536047641864
micro f1 score : 0.44865403788634095
hamming loss : 0.24867903316469928
Precision recall report :
               precision    recall  f1-score   support

            0       0.32      0.52      0.39       487
            1       0.46      0.48      0.47       668
            2       0.44      0.52      0.48       628

    micro avg       0.40      0.50      0.45      1783
    macro avg       0.41      0.51      0.45      1783
weighted avg       0.42      0.50      0.45      1783
 samples avg       0.18      0.22      0.19      1783
```

4.9.3 Char-4-Gram with Logistic Regression

```
start = datetime.now()
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l2'), n_jobs=-1)
classifier.fit(x_train_4char, multilabel_y_train)
predictions = classifier.predict(x_test_4char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
accuracy : 0.07790893760539629
macro f1 score : 0.08327715307415981
micro f1 score : 0.31385045857298305
hamming loss : 0.04193525401990357
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.56      0.16      0.24       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.00      0.00      0.00        18
           5       1.00      0.03      0.06        35
           6       0.33      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.50      0.02      0.04        45
          10       1.00      0.09      0.17        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.31      0.12      0.17       368
          18       0.00      0.00      0.00        27
          19       0.00      0.00      0.00        97
          20       0.42      0.23      0.30       551
          21       0.00      0.00      0.00        41
          22       0.80      0.05      0.09        85
          23       0.50      0.02      0.05        42
          24       0.17      0.01      0.02        83
          25       0.33      0.03      0.05       159
          26       0.70      0.19      0.30       112
          27       0.00      0.00      0.00        11
          28       0.39      0.21      0.27       596
          29       0.62      0.18      0.28       190
          30       0.46      0.14      0.22        83
          31       0.00      0.00      0.00        12
          32       0.33      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       0.33      0.08      0.13        25
          35       0.00      0.00      0.00        29
          36       0.56      0.11      0.18        92
          37       0.21      0.02      0.03       172
          38       0.41      0.05      0.09       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.40      0.02      0.04        93
          43       0.65      0.58      0.61      1155
          44       0.09      0.01      0.02       117
          45       0.47      0.17      0.24       145
          46       0.00      0.00      0.00         5
          47       0.28      0.05      0.09       129
          48       0.00      0.00      0.00        36
          49       0.25      0.02      0.04        44
          50       0.00      0.00      0.00        28
          51       0.00      0.00      0.00        51
          52       0.48      0.21      0.30       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.45      0.23      0.30       507
          57       0.52      0.31      0.39       587
          58       0.56      0.03      0.06       148
          59       0.43      0.06      0.10       173
          60       0.83      0.08      0.14        66
          61       0.00      0.00      0.00        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.25      0.08      0.12       228
          66       0.00      0.00      0.00        27
          67       0.36      0.04      0.08       119
          68       0.62      0.51      0.56       911
          69       0.00      0.00      0.00        14
          70       0.00      0.00      0.00        13

   micro avg       0.53      0.22      0.31      9021
   macro avg       0.23      0.06      0.08      9021
weighted avg       0.42      0.22      0.27      9021
```

```
 weighted avg       0.42      0.22      0.27      9021
  samples avg       0.38      0.26      0.27      9021

Time taken to run this cell : 0:04:31.772192
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-3-Gram")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.048, 0.46, 0.025])
x.add_row(["SGD With Log Loss",0.048, 0.44, 0.024 ])
x.add_row(["Logistic Regression",0.077, 0.31, 0.041 ])

print(x)
```

```
Comparison of models with Char-3-Gram
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.048   |      0.46      |    0.025     |
|  SGD With Log Loss  |  0.048   |      0.44      |    0.024     |
| Logistic Regression |  0.077   |      0.31      |    0.041     |
+---------------------+----------+----------------+--------------+
```

Observations

1) We can see that Char-3-grams gave us the best performing data so far.

2) Going forward we will try to implement Char-3-Grams with different feauture sizes on labels.

4.10 Using only 3 tags

We saw in the EDA section that each movie title has an average of 2.9 tags. So lets try implementing a ML models using only 3 tags per movie.

```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3)
multilabel_y_train = vectorizer.fit_transform(y_train)
multilabel_y_test = vectorizer.transform(y_test)
```

4.10.1 Char-3-Grams with 3 tags and SGD with Hinge Loss

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(3, 3))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

```
print("Dimensions of train data X:",x_train_3char.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_3char.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 10893) Y : (11816, 3)
Dimensions of test data X: (2965, 10893) Y: (2965, 3)
```

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.000001,0.00001,0.0001,0.001], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.4414839797639123
macro f1 score : 0.5080402989342728
micro f1 score : 0.574238698694912
hamming loss : 0.2530635188308038
Precision recall report :
               precision    recall  f1-score   support

            0       0.57      0.18      0.27       596
            1       0.66      0.64      0.65      1155
            2       0.51      0.73      0.60       911

   micro avg       0.58      0.57      0.57      2662
   macro avg       0.58      0.52      0.51      2662
weighted avg       0.59      0.57      0.55      2662
 samples avg       0.32      0.33      0.31      2662
```

4.10.2 Char-3-Grams with 3 tags and SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:00:17.748189
The best hyper parameters are  {'estimator__alpha': 0.001, 'estimator__penalty': 'l1'}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.41922428330522765
macro f1 score : 0.5084075209697796
micro f1 score : 0.5624538063562453
hamming loss : 0.26621697582911746
Precision recall report :
               precision    recall  f1-score   support

            0       0.50      0.22      0.30       596
            1       0.64      0.64      0.64      1155
            2       0.49      0.72      0.58       911

   micro avg       0.55      0.57      0.56      2662
   macro avg       0.54      0.53      0.51      2662
weighted avg       0.56      0.57      0.54      2662
 samples avg       0.31      0.33      0.31      2662
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-3-Gram and 3 tags")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.044, 0.57, 0.025])
x.add_row(["SGD With Log Loss",0.041, 0.56, 0.026 ])


print(x)
```

```
Comparison of models with Char-3-Gram and 3 tags
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.044   |      0.57      |    0.025     |
|  SGD With Log Loss  |  0.041   |      0.56      |    0.026     |
+---------------------+----------+----------------+--------------+
```

4.10.3 Char-4-Grams with 3 tags and SGD with Hinge Loss

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(4, 4))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

```
print("Dimensions of train data X:",x_train_3char.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_3char.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 84799) Y : (11816, 3)
Dimensions of test data X: (2965, 84799) Y: (2965, 3)
```

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.000001,0.00001,0.0001,0.001], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.45193929173693087
macro f1 score : 0.5213300704092406
micro f1 score : 0.5723774174643484
hamming loss : 0.24609331084879146
Precision recall report :
               precision    recall  f1-score   support

           0       0.54      0.23      0.32       596
           1       0.69      0.59      0.64      1155
           2       0.53      0.71      0.61       911

   micro avg       0.60      0.55      0.57      2662
   macro avg       0.59      0.51      0.52      2662
weighted avg       0.60      0.55      0.56      2662
 samples avg       0.31      0.32      0.30      2662
```

4.10.4 Char-4-Grams with 3 tags and SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log')), param_grid=param_grid, n_jobs=-1)
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.0001, penalty='l2', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.4155143338954469
macro f1 score : 0.5660380696131014
micro f1 score : 0.5903386625408286
hamming loss : 0.26790331646992693
Precision recall report :
               precision    recall  f1-score   support

            0       0.35      0.47      0.40       596
            1       0.65      0.70      0.68      1155
            2       0.56      0.69      0.62       911

   micro avg       0.54      0.65      0.59      2662
   macro avg       0.52      0.62      0.57      2662
weighted avg       0.55      0.65      0.60      2662
 samples avg       0.33      0.37      0.33      2662
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-4-Gram and 3 tags")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.045, 0.57, 0.024])
x.add_row(["SGD With Log Loss",0.045, 0.59, 0.026 ])


print(x)
```

```
Comparison of models with Char-4-Gram and 3 tags
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.045   |      0.57      |    0.024     |
|  SGD With Log Loss  |  0.045   |      0.59      |    0.026     |
+---------------------+----------+----------------+--------------+
```

4.11 Using only 4 tags

We saw in the EDA section that each movie title has an average of 2.9 tags. So lets try implementing a ML models using only 4 tags per movie.

```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=4)
multilabel_y_train = vectorizer.fit_transform(y_train)
multilabel_y_test = vectorizer.transform(y_test)
```

4.11.1 Char-3-Grams with 4 tags and SGD with Hinge Loss

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(3, 3))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

```
print("Dimensions of train data X:",x_train_3char.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_3char.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 10893) Y : (11816, 4)
Dimensions of test data X: (2965, 10893) Y: (2965, 4)
```

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.000001,0.00001,0.0001,0.001], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge')), param_grid=param_grid, n_jobs=-1)
gsv1.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)
```

```
print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.3214165261382799
macro f1 score : 0.43860088426525345
micro f1 score : 0.4421969814843629
hamming loss : 0.3022765598650928
Precision recall report :
               precision    recall  f1-score   support

            0       0.28      0.58      0.38       503
            1       0.31      0.57      0.40       487
            2       0.41      0.65      0.50       668
            3       0.37      0.66      0.47       628

    micro avg       0.34      0.62      0.44      2286
    macro avg       0.34      0.62      0.44      2286
weighted avg       0.35      0.62      0.45      2286
 samples avg       0.21      0.30      0.23      2286
```

4.11.2 Char-3-Grams with 4 tags and SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log')), param_grid=param_grid, n_jobs=-1)
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.3254637436762226
macro f1 score : 0.40676411478678437
micro f1 score : 0.4113072090936285
hamming loss : 0.3318718381112985
Precision recall report :
               precision    recall  f1-score   support

            0       0.23      0.52      0.32       503
            1       0.30      0.52      0.38       487
            2       0.37      0.65      0.47       668
            3       0.34      0.67      0.45       628

    micro avg       0.31      0.60      0.41      2286
    macro avg       0.31      0.59      0.41      2286
weighted avg       0.32      0.60      0.41      2286
 samples avg       0.17      0.29      0.20      2286
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-3-Gram and 4 tags")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.032, 0.44, 0.030])
x.add_row(["SGD With Log Loss",0.032, 0.41, 0.033])


print(x)
```

```
Comparison of models with Char-3-Gram and 4 tags
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.032   |      0.44      |     0.03     |
|  SGD With Log Loss  |  0.032   |      0.41      |    0.033     |
+---------------------+----------+----------------+--------------+
```

4.11.3 Char-4-Grams with 4 tags and SGD with Hinge Loss

```
# Lets try to implement char-gram and see if there is an improvement in performance compared to the previous models.
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(4, 4))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

```
print("Dimensions of train data X:",x_train_3char.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",x_test_3char.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 84799) Y : (11816, 4)
Dimensions of test data X: (2965, 84799) Y: (2965, 4)
```

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.000001,0.00001,0.0001,0.001], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge')), param_grid=param_grid, n_jobs=-1)
gsv1.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.28802698145025296
macro f1 score : 0.5418774738406413
micro f1 score : 0.564021164021164
hamming loss : 0.2779089376053963
Precision recall report :
               precision    recall  f1-score   support

           0       0.31      0.49      0.38       596
           1       0.65      0.72      0.69      1155
           2       0.40      0.63      0.49       587
           3       0.55      0.70      0.61       911

   micro avg       0.49      0.66      0.56      3249
   macro avg       0.48      0.63      0.54      3249
weighted avg       0.51      0.66      0.57      3249
 samples avg       0.40      0.46      0.40      3249
```

4.11.4 Char-4-Grams with 4 tags and SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log')), param_grid=param_grid, n_jobs=-1)
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.28566610455311975
macro f1 score : 0.48925918282624536
micro f1 score : 0.5312725696952189
hamming loss : 0.2736087689713322
Precision recall report :
               precision    recall  f1-score   support

            0       0.51      0.21      0.30       596
            1       0.65      0.61      0.63      1155
            2       0.36      0.61      0.45       587
            3       0.49      0.71      0.58       911

    micro avg       0.50      0.57      0.53      3249
    macro avg       0.50      0.54      0.49      3249
 weighted avg       0.52      0.57      0.52      3249
  samples avg       0.38      0.41      0.37      3249
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-4-Gram and 4 tags")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.028, 0.56, 0.027])
x.add_row(["SGD With Log Loss",0.028, 0.53, 0.027])

print(x)
```

```
Comparison of models with Char-4-Gram and 4 tags
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.028   |      0.56      |    0.027     |
|  SGD With Log Loss  |  0.028   |      0.53      |    0.027     |
+---------------------+----------+----------------+--------------+
```

4.12 Glove Vector Model

```
#https://stackoverflow.com/questions/37793118/load-pretrained-glove-vectors-in-python
#https://nlp.stanford.edu/projects/glove/
from gensim.scripts.glove2word2vec import glove2word2vec
glove2word2vec(glove_input_file="glove.6B.50d.txt", word2vec_output_file="gensim_glove_vectors.txt")
```

```
(400000, 50)
```

```
from gensim.models.keyedvectors import KeyedVectors
glove_model = KeyedVectors.load_word2vec_format("gensim_glove_vectors.txt", binary=False)
```

```
glove_words = list(glove_model.wv.vocab)
```

```
from tqdm import tqdm_notebook as tqdm
def vectorize_glove(dataset, glove_model, glove_words):
    glove_corpus=[]
    for sentence in dataset:
        glove_corpus.append(sentence.split())

    # Creating average Glove model by computing the average word2vec for each review.
    sent_vectors = []; #The average word2vec for each sentence/review will be stored in this list
    for sentence in tqdm(glove_corpus): #For each review
        sent_vec = np.zeros(50) #50 dimensional array, where all elements are zero. This is used to add word vectors and find the
        count_words =0; #This will store the count of the words with a valid vector in each review text
        for word in sentence: #For each word in a given review.
            if word in glove_words:
                word_vectors = glove_model.wv[word] #Creating a vector(numpy array of 300 dimensions) for each word.
                sent_vec += word_vectors
                count_words += 1
        if count_words != 0:
            sent_vec /= count_words
        sent_vectors.append(sent_vec)
```

```
    #print("\nThe length of the sentence vectors :",len(sent_vectors))
    #print("\nSize of each vector : ",len(sent_vectors[0]))
    sent_vectors = np.array(sent_vectors)
    return sent_vectors


X_train_glove = vectorize_glove(x_train, glove_model, glove_words)
X_test_glove = vectorize_glove(x_test, glove_model, glove_words)

import pickle
with open('x_train_glove.pkl', 'wb') as file:
    pickle.dump(X_train_glove, file)

with open('x_test_glove.pkl', 'wb') as file:
    pickle.dump(X_test_glove, file)
```

```
HBox(children=(IntProgress(value=0, max=11816), HTML(value='')))
```

4.13 LDA - TOPIC MODELLING

Lets try to implement Char-4 grams with 3 tags along with LDA to see if we can further improve the performance.

https://www.youtube.com/watch?v=Cpt97Bpl-t4

https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/

https://towardsdatascience.com/unsupervised-nlp-topic-models-as-a-supervised-learning-input-cf8ee9e5cf28

```
import gensim, spacy, logging, warnings
import gensim.corpora as corpora
from gensim.utils import lemmatize, simple_preprocess
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt
from pprint import pprint
from gensim.models import ldamulticore
```

```
final.head()
```

	imdb_id	title	plot_synopsis	tags	split	synopsis_
8814	tt0000091	Le manoir du diable	The film opens with a large bat flying into a ...	paranormal, gothic	train	wi
7085	tt0000225	La belle et la bête	A widower merchant lives in a mansion with his...	fantasy	train	wi
11909	tt0000230	Cendrillon	A prologue in front of the curtain, suppressed...	fantasy	train	wi
548	tt0000417	Le voyage dans la lune	At a meeting of the Astronomic Club, its presi...	psychedelic, satire	train	

```
#Adding a colmn with a list of words
final["word_list"]=final["preprocessed_synopsis"].apply(lambda x: x.split())
```

```
final.head()
```

	imdb_id	title	plot_synopsis	tags	split	synopsis_source	prep
8814	tt0000091	Le manoir du diable	The film opens with a large bat flying into a ...	paranormal, gothic	train	wikipedia	film
7085	tt0000225	La belle et la bête	A widower merchant lives in a mansion with his...	fantasy	train	wikipedia	widower m
11909	tt0000230	Cendrillon	A prologue in front of the curtain, suppressed...	fantasy	train	wikipedia	prologue fro
548	tt0000417	Le voyage dans la lune	At a meeting of the Astronomic Club, its presi...	psychedelic, satire	train	imdb	me pre
6042	tt0000488	The Land Beyond the Sunset	Joe is an impoverished New York newsboy who li...	fantasy, storytelling	train	wikipedia	joe ir n

```
word_list=[]
for word in final["word_list"].values:
    word_list.append(word)
```

```
word_list[:1]
```

```
[['film',
  'opens',
  'large',
  'bat',
  'flying',
  'medieval',
  'castle',
  'bat',
  'circles',
  'room',
  'suddenly',
  'changing',
  'mephistopheles',
  'incarnation',
  'devil',
  'mephistopheles',
  'produces',
  'cauldron',
  'assistant',
  'helps',
  'conjure',
  'woman',
  'cauldron',
  'room',
  'cleared',
  'shortly',
  'two',
  'cavaliers',
  'enter',
  'devil',
  'assistant',
  'pokes',
  'backs',
  'instantaneously',
  'transporting',
  'different',
  'areas',
  'room',
  'confusing',
  'pair',
  'causing',
  'one',
  'flee',
  'second',
  'stays',
  'several',
  'tricks',
  'played',
  'furniture',
  'moved',
  'around',
  'sudden',
  'appearance',
  'skeleton',
  'cavalier',
  'unfazed',
  'using',
  'sword',
  'attack',
  'skeleton',
  'turns',
  'bat',
  'mephistopheles',
  'conjures',
  'four',
  'spectres',
  'subdue',
  'man',
  'recovering',
  'spectres',
  'attack',
  'man',
  'visibly',
  'dazed',
  'brought',
  'woman',
  'cauldron',
  'impresses',
  'beauty',
  'mephistopheles',
  'turns',
  'withered'
```

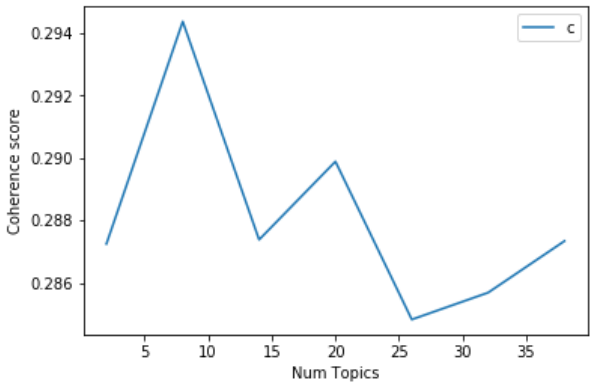
```
        'withered',
        'old',
        'crone',
        'front',
        'man',
        'eyes',
        'four',
        'spectres',
        'second',
        'cavalier',
        'returns',
        'brief',
        'show',
        'bravery',
        'flees',
        'time',
        'leaping',
        'balcony',
        'edge',
        'spectres',
        'disappear',
        'cavalier',
        'confronted',
        'face',
        'face',
        'devil',
        'reaching',
        'brandishing',
        'large',
        'crucifix',
        'causes',
        'devil',
        'vanish']]
```

4.13.1 Hyperparameter tuning to determing number of topics

```
from tqdm import tqdm_notebook as tqdm


id2word= corpora.Dictionary(word_list)
corpus = [id2word.doc2bow(text) for text in word_list]


#https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    coherence_values = []
    model_list = []
    for num_topics in tqdm(range(start, limit, step)):
        model = gensim.models.LdaMulticore(corpus=corpus, num_topics=num_topics, id2word=id2word, workers=7)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence='c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values


model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus, texts=word_list, start=2, limit=40, ste
```

```
HBox(children=(IntProgress(value=0, max=7), HTML(value='')))
```

```
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
```



We are getting a good coherence when we take close to 10 topics. Hence let us take number of topics as 8.

4.13.2 GENERATING LDA MODEL

```
# Build LDA model
start = datetime.now()
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=8,
                                       random_state=100,
                                       chunksize=10,
                                       passes=10,
                                       alpha='symmetric',
                                       iterations=15,
                                       per_word_topics=True,
                                       workers=8)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:32:10.650443
```

```
pprint(lda_model.print_topics())
```

```
[(0,
  '0.011*"not" + 0.008*"mother" + 0.008*"father" + 0.007*"family" + '
  '0.007*"love" + 0.006*"life" + 0.006*"home" + 0.005*"one" + 0.005*"time" + '
  '0.004*"film"'),
 (1,
  '0.017*"not" + 0.015*"tells" + 0.012*"says" + 0.011*"back" + 0.008*"asks" + '
  '0.007*"room" + 0.007*"see" + 0.007*"man" + 0.006*"get" + 0.006*"one"'),
 (2,
  '0.016*"de" + 0.016*"marcus" + 0.015*"nina" + 0.014*"le" + 0.013*"el" + '
  '0.012*"maria" + 0.011*"la" + 0.010*"n" + 0.009*"robin" + 0.009*"que"'),
 (3,
  '0.047*"peter" + 0.035*"alice" + 0.021*"lucy" + 0.020*"jacob" + '
  '0.017*"vampire" + 0.011*"vampires" + 0.011*"jonathan" + 0.011*"wolf" + '
  '0.010*"snow" + 0.010*"blood"'),
 (4,
  '0.014*"police" + 0.008*"kill" + 0.008*"killed" + 0.006*"john" + '
  '0.006*"kills" + 0.006*"man" + 0.005*"gang" + 0.005*"death" + 0.005*"murder" '
  '+ 0.005*"one"'),
 (5,
  '0.008*"king" + 0.005*"fight" + 0.004*"father" + 0.004*"village" + '
  '0.004*"one" + 0.004*"prince" + 0.004*"battle" + 0.003*"queen" + '
  '0.003*"arthur" + 0.003*"castle"'),
 (6,
  '0.005*"one" + 0.004*"team" + 0.004*"group" + 0.003*"two" + 0.003*"also" + '
  '0.003*"new" + 0.003*"escape" + 0.003*"world" + 0.002*"city" + 0.002*"find"'),
 (7,
  '0.008*"not" + 0.006*"tells" + 0.005*"get" + 0.005*"car" + 0.005*"back" + '
  '0.004*"go" + 0.004*"david" + 0.004*"home" + 0.004*"jack" + 0.004*"goes"')]
```

```
def format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=word_list):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list
        # print(row)
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
```

```
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,4), topic_keywords]), ignore_in
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

    # Add original text to the end of the output
    contents = pd.Series(texts)
    sent_topics_df = pd.concat([sent_topics_df, contents], axis=1)
    return(sent_topics_df)


df_topic_sents_keywords = format_topics_sentences(ldamodel=lda_model, corpus=corpus, texts=word_list)

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'Text']
#Show
df_dominant_topic.head(10)
```

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	
0	0	1.0	0.5572	not, tells, says, back, asks, room, see, man, ...	[film, opens, large, bat,
1	1	5.0	0.4883	king, fight, father, village, one, prince, bat...	[widower, merchant, lives
2	2	0.0	0.4257	not, mother, father, family, love, life, home,...	[prologue, front, curtain, s
3	3	6.0	0.6535	one, team, group, two, also, new, escape, worl...	[meeting, astronomic, club
4	4	0.0	0.3505	not, mother, father, family, love, life, home,...	[joe, impoverished, new, y
5	5	2.0	0.4741	de, marcus, nina, le, el, maria, la, n, robin,...	[mark, antony, one, triumv
6	6	5.0	0.7055	king, fight, father, village, one, prince, bat...	[play, opens, commoners, rc
7	7	0.0	0.3451	not, mother, father, family, love, life, home,...	[play, consists, four, int
8	8	0.0	0.6243	not, mother, father, family, love, life, home,...	[film, starts, intro, page,
9	9	4.0	0.8981	police, kill, killed, john, kills, man, gang, ...	[dr, richard, kimble, promi

```
import pickle
df_dominant_topic.to_pickle("lda_model")
```

```
df_dominant_topic.head()
```

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	
0	0	1.0	0.5572	not, tells, says, back, asks, room, see, man, ...	[film, opens, large, bat, fl
1	1	5.0	0.4883	king, fight, father, village, one, prince, bat...	[widower, merchant, lives, r
2	2	0.0	0.4257	not, mother, father, family, love, life, home,...	[prologue, front, curtain, su
3	3	6.0	0.6535	one, team, group, two, also, new, escape, worl...	[meeting, astronomic, club,
4	4	0.0	0.3505	not, mother, father, family, love, life, home,...	[joe, impoverished, new, yor

```
final.head()
```

	title	tags	split	preprocessed_synopsis	
8814	Le manoir du diable	paranormal, gothic	train	film opens large bat flying medieval castle ba...	[film, opens, large
7085	La belle et la bête	fantasy	train	widower merchant lives mansion six children th...	[widower, merchant
11909	Cendrillon	fantasy	train	prologue front curtain suppressed premiere int...	[prologue, front, cu
548	Le voyage dans la lune	psychedelic, satire	train	meeting astronomic club president professor ba...	[meeting, astronomi
6042	The Land Beyond the Sunset	fantasy, storytelling	train	joe impoverished new york newsboy lives abusiv...	[joe, impoverished,

```
combined_df=pd.concat([final,df_dominant_topic], axis=1)
combined_df.head()
```

	title	tags	split	preprocessed_synopsis	word_list	synopsis_length	Document_No	Dominant_Top
0	I tre volti della paura	cult, horror, gothic, murder, atmospheric	train	note synopsis orginal italian release segments...	[note, synopsis, orginal, italian, release, se...	7527.0	0.0	
1	Dungeons & Dragons: The Book of Vile Darkness	violence	train	two thousand years ago nhagruul foul sorcerer ...	[two, thousand, years, ago, nhagruul, foul, so...	2077.0	1.0	
2	The Shop Around the Corner	romantic	test	matuschek gift store budapest workplace alfred...	[matuschek, gift, store, budapest, workplace, ...	4003.0	2.0	
3	Mr. Holland's Opus	inspiring, romantic, stupid, feel-good	train	glenn holland not morning person anyone standa...	[glenn, holland, not, morning, person, anyone,...	13215.0	3.0	
4	Scarface	cruelty, murder, dramatic, cult, violence, atm...	val	may cuban man named tony montana al pacino cla...	[may, cuban, man, named, tony, montana, al, pa...	17575.0	4.0	

4.13.3 SPLITTING DATA AND VECTORIZING

```
#https://stackoverflow.com/questions/24775648/element-wise-logical-or-in-pandas

train_data=combined_df.loc[(combined_df['split'] == 'train') | (combined_df['split'] == 'val')]
test_data=combined_df.loc[(combined_df['split'] == 'test')]

x_train = train_data['preprocessed_synopsis']
y_train = train_data['tags']

x_test = test_data['preprocessed_synopsis']
y_test = test_data['tags']


print("The shape of training data is :" ,x_train.shape)
print("The shape of test data is :" ,x_test.shape)
```

```
#https://stackoverflow.com/questions/15547409/how-to-get-rid-of-punctuation-using-nltk-tokenizer

def tokenize(sent):
    sent=sent.split(',')
    tags=[i.strip() for i in sent]
    return tags

# binary='true' will give a binary vectorizerr
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3)
multilabel_y_train = vectorizer.fit_transform(y_train)
multilabel_y_test = vectorizer.transform(y_test)


print("The shape of training label data is :" ,multilabel_y_train.shape)
print("The shape of test label data is :" ,multilabel_y_test.shape)
```

```
The shape of training label data is : (11816, 3)
The shape of test label data is : (2965, 3)
```

```
## Applying Char-4-grams
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(4, 4))
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
```

4.13.4 SGD with Hinge Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.000001,0.00001,0.0001,0.001], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge')), param_grid=param_grid, n_jobs=-1)
gsv1.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.40741989881956153
macro f1 score : 0.5770508339805563
micro f1 score : 0.6001332889036987
hamming loss : 0.26981450252951095
Precision recall report :
               precision    recall  f1-score   support

            0       0.34      0.51      0.41       596
            1       0.67      0.72      0.69      1155
            2       0.56      0.73      0.63       911

    micro avg       0.54      0.68      0.60      2662
    macro avg       0.52      0.65      0.58      2662
weighted avg       0.56      0.68      0.61      2662
 samples avg       0.34      0.39      0.35      2662
```

4.13.5 Char-4-Grams with 3 tags and SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log')), param_grid=param_grid, n_jobs=-1)
gsv.fit(x_train_3char, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.0001, penalty='l2', class_weight='balanced'), n_jobs=-1)
classifier.fit(x_train_3char, multilabel_y_train)
predictions = classifier.predict(x_test_3char)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.39426644182124787
macro f1 score : 0.5688797835595052
micro f1 score : 0.5928489042675893
hamming loss : 0.277796514896009
Precision recall report :
              precision    recall  f1-score   support

           0       0.33      0.51      0.40       596
           1       0.64      0.75      0.69      1155
           2       0.56      0.69      0.62       911

   micro avg       0.53      0.68      0.59      2662
   macro avg       0.51      0.65      0.57      2662
weighted avg       0.54      0.68      0.60      2662
 samples avg       0.33      0.39      0.34      2662
```

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-4-Gram and 3 tags and LDA")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.040, 0.60, 0.026])
x.add_row(["SGD With Log Loss",0.039, 0.59, 0.027])


print(x)
```

```
Comparison of models with Char-4-Gram and 3 tags and LDA
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |   0.04   |      0.6       |    0.026     |
|  SGD With Log Loss  |  0.039   |      0.59      |    0.027     |
+---------------------+----------+----------------+--------------+
```

4.14 Char 3 grams + Char 4 grams

```
start = datetime.now()
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(3, 3),  max_features=20000)
x_train_3char = vectorizer.fit_transform(x_train)
x_test_3char = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:25.052465
```

```
start = datetime.now()
vectorizer = TfidfVectorizer(sublinear_tf=True, strip_accents='unicode', analyzer='char', ngram_range=(4, 4),  max_features=20000)
x_train_4char = vectorizer.fit_transform(x_train)
x_test_4char = vectorizer.transform(x_test)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:00:33.640678
```

```
from scipy.sparse import coo_matrix, hstack
train_datacombine = hstack((x_train_3char, x_train_4char),format="csr",dtype='float64')
test_datacombine = hstack((x_test_3char, x_test_4char),format="csr",dtype='float64')


print("Dimensions of train data X:",train_datacombine.shape, "Y :",multilabel_y_train.shape)
print("Dimensions of test data X:",test_datacombine.shape,"Y:",multilabel_y_test.shape)
```

```
Dimensions of train data X: (11816, 30893) Y : (11816, 71)
Dimensions of test data X: (2965, 30893) Y: (2965, 71)
```

4.14.1 Char-3 -Gram and Char-4-Gram with SGD with Hinge Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv1 = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='hinge', class_weight='balanced')), param_grid=param_grid, n
gsv1.fit(train_datacombine, multilabel_y_train)
```

```
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv1.best_params_)
```

```
Time taken to run this cell : 0:23:07.368385
The best hyper parameters are  {'estimator__penalty': 'l2', 'estimator__alpha': 0.0001}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='hinge', alpha=0.0001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(train_datacombine, multilabel_y_train)
predictions = classifier.predict(test_datacombine)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.04283305227655986
macro f1 score : 0.2110730006978387
micro f1 score : 0.3751404350934532
hamming loss : 0.05812412417167423
Precision recall report :
              precision    recall  f1-score   support

           0       0.08      0.05      0.07        56
           1       0.31      0.46      0.37       129
           2       0.18      0.32      0.23        28
           3       0.16      0.14      0.15        22
           4       0.46      0.33      0.39        18
           5       0.23      0.09      0.13        35
           6       0.21      0.13      0.16        30
           7       0.15      0.08      0.10        79
           8       0.00      0.00      0.00         8
           9       0.16      0.07      0.09        45
          10       0.50      0.36      0.42        11
          11       0.02      0.35      0.05        40
          12       0.18      0.12      0.14       115
          13       0.67      0.11      0.19        18
          14       0.33      0.33      0.33         9
          15       0.33      0.07      0.11        15
          16       0.08      0.08      0.08        13
          17       0.26      0.43      0.33       368
          18       0.33      0.07      0.12        27
          19       0.11      0.09      0.10        97
          20       0.35      0.47      0.40       551
          21       0.14      0.07      0.10        41
          22       0.17      0.07      0.10        85
          23       0.25      0.12      0.16        42
          24       0.10      0.08      0.09        83
          25       0.22      0.24      0.23       159
          26       0.55      0.30      0.39       112
          27       0.00      0.00      0.00        11
          28       0.34      0.46      0.39       596
          29       0.41      0.44      0.42       190
          30       0.35      0.52      0.42        83
          31       0.11      0.08      0.10        12
          32       0.37      0.27      0.31        26
          33       0.12      0.08      0.09        64
          34       0.22      0.28      0.25        25
          35       0.14      0.07      0.09        29
          36       0.33      0.27      0.30        92
          37       0.20      0.13      0.16       172
          38       0.23      0.26      0.24       136
          39       0.27      0.12      0.17        32
          40       0.26      0.12      0.17        40
          41       0.00      0.00      0.00         5
          42       0.20      0.14      0.16        93
          43       0.63      0.70      0.66      1155
          44       0.18      0.18      0.18       117
          45       0.32      0.37      0.34       145
          46       0.17      0.20      0.18         5
          47       0.27      0.24      0.25       129
          48       0.24      0.14      0.18        36
          49       0.09      0.20      0.12        44
          50       0.21      0.46      0.29        28
          51       0.29      0.20      0.24        51
          52       0.31      0.42      0.36       395
          53       0.21      0.18      0.20        65
          54       0.14      0.20      0.17        15
          55       0.05      0.03      0.04        37
          56       0.35      0.47      0.40       507
          57       0.42      0.52      0.47       587
          58       0.28      0.24      0.26       148
          59       0.29      0.23      0.26       173
          60       0.34      0.35      0.35        66
          61       0.12      0.10      0.11        51
          62       0.05      0.09      0.06        66
          63       0.04      0.05      0.04        39
          64       0.00      0.00      0.00         8
          65       0.23      0.29      0.25       228
          66       0.00      0.00      0.00        27
          67       0.16      0.10      0.12       119
          68       0.57      0.68      0.62       911
          69       0.58      0.50      0.54        14
          70       0.00      0.00      0.00        13

   micro avg       0.35      0.41      0.38      9021
   macro avg       0.23      0.22      0.21      9021
weighted avg       0.36      0.41      0.37      9021
```

```
weighted avg       0.50      0.41      0.39      9021
 samples avg       0.34      0.44      0.34      9021
```

4.14.2 Char-3 -Gram and Char-4-Gram with SGD with Log Loss

```
#Hyperparameter tuning

from sklearn.model_selection import GridSearchCV

start = datetime.now()

param_grid = dict(estimator__alpha=[0.00001,0.0001,0.001,0.1], estimator__penalty=["l1","l2"])

gsv = GridSearchCV(OneVsRestClassifier(estimator= SGDClassifier(loss='log', class_weight='balanced')), param_grid=param_grid, n_jo
gsv.fit(train_datacombine, multilabel_y_train)
print("Time taken to run this cell :", datetime.now() - start)
print('The best hyper parameters are ', gsv.best_params_)
```

```
Time taken to run this cell : 0:26:58.847473
The best hyper parameters are  {'estimator__penalty': 'l2', 'estimator__alpha': 1e-05}
```

```
classifier = OneVsRestClassifier(SGDClassifier(loss='log', alpha=0.00001, penalty='l2',class_weight='balanced'), n_jobs=-1)
classifier.fit(train_datacombine, multilabel_y_train)
predictions = classifier.predict(test_datacombine)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
```

```
accuracy : 0.05092748735244519
macro f1 score : 0.20858883619189467
micro f1 score : 0.38177436529731545
hamming loss : 0.052400066503574566
Precision recall report :
              precision    recall  f1-score   support

           0       0.08      0.02      0.03        56
           1       0.31      0.36      0.33       129
           2       0.37      0.25      0.30        28
           3       0.25      0.05      0.08        22
           4       0.50      0.28      0.36        18
           5       0.31      0.11      0.17        35
           6       0.43      0.20      0.27        30
           7       0.18      0.15      0.16        79
           8       0.00      0.00      0.00         8
           9       0.17      0.09      0.12        45
          10       0.80      0.36      0.50        11
          11       0.00      0.00      0.00        40
          12       0.20      0.11      0.14       115
          13       1.00      0.11      0.20        18
          14       0.33      0.33      0.33         9
          15       0.33      0.07      0.11        15
          16       0.17      0.08      0.11        13
          17       0.30      0.25      0.28       368
          18       0.40      0.07      0.12        27
          19       0.12      0.06      0.08        97
          20       0.34      0.47      0.40       551
          21       0.09      0.02      0.04        41
          22       0.16      0.11      0.13        85
          23       0.20      0.12      0.15        42
          24       0.11      0.10      0.10        83
          25       0.23      0.19      0.21       159
          26       0.53      0.33      0.41       112
          27       0.00      0.00      0.00        11
          28       0.34      0.39      0.36       596
          29       0.42      0.39      0.41       190
          30       0.40      0.48      0.44        83
          31       0.20      0.17      0.18        12
          32       0.38      0.23      0.29        26
          33       0.19      0.12      0.15        64
          34       0.22      0.20      0.21        25
          35       0.33      0.03      0.06        29
          36       0.31      0.36      0.33        92
          37       0.18      0.17      0.18       172
          38       0.24      0.24      0.24       136
          39       0.43      0.09      0.15        32
          40       0.38      0.07      0.12        40
          41       0.00      0.00      0.00         5
          42       0.17      0.09      0.11        93
          43       0.59      0.72      0.65      1155
          44       0.19      0.11      0.14       117
          45       0.29      0.37      0.33       145
          46       0.25      0.20      0.22         5
          47       0.27      0.27      0.27       129
          48       0.24      0.14      0.18        36
          49       0.18      0.07      0.10        44
          50       0.25      0.29      0.27        28
          51       0.36      0.18      0.24        51
          52       0.41      0.30      0.34       395
          53       0.29      0.15      0.20        65
          54       0.21      0.20      0.21        15
          55       0.08      0.03      0.04        37
          56       0.39      0.36      0.37       507
          57       0.37      0.57      0.45       587
          58       0.33      0.18      0.23       148
          59       0.26      0.21      0.23       173
          60       0.34      0.27      0.30        66
          61       0.12      0.04      0.06        51
          62       0.10      0.08      0.09        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.23      0.29      0.25       228
          66       0.50      0.04      0.07        27
          67       0.14      0.13      0.14       119
          68       0.57      0.64      0.60       911
          69       0.55      0.43      0.48        14
          70       0.00      0.00      0.00        13

   micro avg       0.39      0.38      0.38      9021
   macro avg       0.28      0.19      0.21      9021
weighted avg       0.36      0.38      0.36      9021
```

```
weighted avg       0.50      0.58      0.50      9021
 samples avg       0.37      0.41      0.34      9021
```

4.14.3 Char-3 -Gram and Char-4-Gram with Logistic Regression

```
start = datetime.now()
classifier = OneVsRestClassifier(LogisticRegression(C=100,penalty='l2'), n_jobs=-1)
classifier.fit(train_datacombine, multilabel_y_train)
predictions = classifier.predict(test_datacombine)

print("accuracy :",metrics.accuracy_score(multilabel_y_test,predictions))
print("macro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'macro'))
print("micro f1 score :",metrics.f1_score(multilabel_y_test, predictions, average = 'micro'))
print("hamming loss :",metrics.hamming_loss(multilabel_y_test,predictions))
print("Precision recall report :\n",metrics.classification_report(multilabel_y_test, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
accuracy : 0.06846543001686341
macro f1 score : 0.09352324581695191
micro f1 score : 0.3187006145741879
hamming loss : 0.04423437759779588
Precision recall report :
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        56
           1       0.48      0.16      0.24       129
           2       0.00      0.00      0.00        28
           3       0.00      0.00      0.00        22
           4       0.00      0.00      0.00        18
           5       1.00      0.03      0.06        35
           6       0.33      0.03      0.06        30
           7       0.00      0.00      0.00        79
           8       0.00      0.00      0.00         8
           9       0.25      0.02      0.04        45
          10       1.00      0.09      0.17        11
          11       0.00      0.00      0.00        40
          12       0.00      0.00      0.00       115
          13       0.00      0.00      0.00        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.00      0.00      0.00        13
          17       0.29      0.17      0.22       368
          18       1.00      0.04      0.07        27
          19       0.11      0.01      0.02        97
          20       0.36      0.26      0.30       551
          21       0.00      0.00      0.00        41
          22       0.44      0.05      0.09        85
          23       0.50      0.02      0.05        42
          24       0.14      0.01      0.02        83
          25       0.27      0.05      0.08       159
          26       0.68      0.21      0.32       112
          27       0.00      0.00      0.00        11
          28       0.36      0.25      0.30       596
          29       0.52      0.20      0.29       190
          30       0.42      0.17      0.24        83
          31       0.00      0.00      0.00        12
          32       0.33      0.04      0.07        26
          33       0.00      0.00      0.00        64
          34       0.29      0.08      0.12        25
          35       0.00      0.00      0.00        29
          36       0.32      0.11      0.16        92
          37       0.21      0.04      0.07       172
          38       0.32      0.09      0.14       136
          39       0.00      0.00      0.00        32
          40       0.00      0.00      0.00        40
          41       0.00      0.00      0.00         5
          42       0.20      0.02      0.04        93
          43       0.61      0.58      0.60      1155
          44       0.06      0.01      0.02       117
          45       0.44      0.20      0.27       145
          46       0.00      0.00      0.00         5
          47       0.27      0.07      0.11       129
          48       0.25      0.03      0.05        36
          49       0.25      0.02      0.04        44
          50       0.00      0.00      0.00        28
          51       0.00      0.00      0.00        51
          52       0.44      0.24      0.31       395
          53       0.00      0.00      0.00        65
          54       0.00      0.00      0.00        15
          55       0.00      0.00      0.00        37
          56       0.39      0.26      0.31       507
          57       0.45      0.33      0.38       587
          58       0.36      0.05      0.09       148
          59       0.39      0.08      0.13       173
          60       0.45      0.08      0.13        66
          61       0.00      0.00      0.00        51
          62       0.00      0.00      0.00        66
          63       0.00      0.00      0.00        39
          64       0.00      0.00      0.00         8
          65       0.25      0.12      0.16       228
          66       0.00      0.00      0.00        27
          67       0.29      0.04      0.07       119
          68       0.60      0.52      0.56       911
          69       0.67      0.14      0.24        14
          70       0.00      0.00      0.00        13

   micro avg       0.47      0.24      0.32      9021
   macro avg       0.23      0.07      0.09      9021
weighted avg       0.38      0.24      0.28      9021
```

```
 weighted avg       0.38      0.24      0.28      9021
  samples avg       0.37      0.28      0.28      9021

Time taken to run this cell : 0:06:26.112625
```

Observations

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of models with Char-3-Gram Char-4-grams")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Hamming Loss']
x.add_row(["SGD With Hinge Loss",0.042, 0.37, 0.058])
x.add_row(["SGD With Log Loss",0.050, 0.38, 0.052])
x.add_row(["Logistic Regression",0.068, 0.31, 0.044])

print(x)
```

```
Comparison of models with Char-3-Gram Char-4-grams
+---------------------+----------+----------------+--------------+
|        Model        | Accuracy | Micro F1 Score | Hamming Loss |
+---------------------+----------+----------------+--------------+
| SGD With Hinge Loss |  0.042   |      0.37      |    0.058     |
|  SGD With Log Loss  |   0.05   |      0.38      |    0.052     |
| Logistic Regression |  0.068   |      0.31      |    0.044     |
+---------------------+----------+----------------+--------------+
```

6. DEEP LEARNING MODELS

We are going to try to implement a few deep learning models as well to see if we can further improve our performance.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.layers import Dropout
from keras.layers import BatchNormalization, Dense, Dropout, Flatten, LSTM
from keras import backend as K
from keras.optimizers import SGD
```

```
Using TensorFlow backend.
```

```
def recall_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
        recall = true_positives / (possible_positives + K.epsilon())
        return recall

def precision_m(y_true, y_pred):
        true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
        predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
        precision = true_positives / (predicted_positives + K.epsilon())
        return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))
```

```
from keras.preprocessing.text import Tokenizer
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(x_train)

x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
```

```
vocab_size = len(tokenizer.word_index) + 1
```

```
max_review_length = 500
x_train = sequence.pad_sequences(x_train, maxlen=max_review_length)
x_test = sequence.pad_sequences(x_test, maxlen=max_review_length)

print(x_train.shape)
print(x_train[1])
```

```
(11816, 500)
[   0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0 3701  291 1003  756  138   66 1703   66 1986 1986
  689 3937 1638   92 1080   92 1071 4428  395    6 2721 1751 1037 1638
 3135   54 2229  212 3701  114  848 3335  992 4874 3701 3786  138  376
  268  191 4984   89  285   69   17 3701  361    3 2356 3012  316 1491
    4 3853 1373 2195  123   16  138 1510  386 4223    4 1703  590 2430
 1549 1986  590 2525 4813 3461 1234  544 3335  883 1638 3895 1430  526
 1560 1857  233  564 3701 3564   18  209 3157  511 4772  123  399  865
  138 1767   96 3701   95  347  978 1694 1812  280 1458  627  696  256
  974 1544 4242  690 3701   18  105  530  677 1180  102 1458 3528  575
 3701  699 1570 1374   29   30  174 3701   45   43  526 1722 3321 1638
  526  186 3452  526   15 3701 1274 1323    2  213 4203 2561 4566 3701
  164  449 3701 1026  156  340 3077 1827  526 1570 3937   98 1323  183
  277  155  526 1638 3701    3 1986   96 3701  702  699 1357 1323  380
   42 3335 4813 2133  879 1703 1986 1638  164  121  129  518 3701  186
 1144   13   34  796  311 1638  667  471   11  555   93 1323 3701  176
  807   25  497  863 1638  183   11 1323  555 1323  631  428 2981  555
 2229   76 2525  530 1634  465   29 1323   16 1638  452 2800    9 1638
 1242 3153  565 1729 1118  776 2921  742  294  452 1323  605   74 1638
    1   50 2405 3153  565 1323   95 1049 1323  586  565 2594 1943  555
 2116  162 2335 2562  121  565 1242  126  576 1638  291   36 1323 1458
  465 3528 3462   25  119 2669 2412  114   95 1026 1323 1048   11   26
   38  927 1357   75 2239 1407   17 1638  183  524   13  995  613  995
  927   26   82    4 1323  555  613  927   96  555  781   66  567   68
 2336  853 1751 1077   15   92 3308  939 1638   34  895 4813 1323  964
 1751  292 1751 2010 3641  883 1638 1323 2367 1751  515  487   36  555
  843  164   96 1323 1675   31  228   39   31  103 1342  479   50  785
  151 1140 1323  525 1638 1045 1430 1124  349 1638  395 1268 1751 2183
  139   53  183  228 1638   52 1139 1530 1045 1430 1323  253  995   26
    4  555 1683  410 1323  997  517   58  375  526   14  526  301  253
  613   96 1323 1638 1323  176  605 1411 1699 1323 4204 3153  565 1638
 1242  565  428  173  734 2619  781 1323 2800  277 1616  542  462   53
  242  157 2321  617 1638  329  268 1626  457  109]
```

```
scaler = StandardScaler()
x_trainstd = scaler.fit_transform(x_train)
x_teststd = scaler.fit_transform(x_test)
```

6.1 Dense Model 1

```
model = Sequential()
model.add(Embedding(input_dim=vocab_size,
                    output_dim=32,
                    input_length=max_review_length))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(71, activation='sigmoid'))
sgd = SGD(lr=0.001, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=[f1_m, 'accuracy'])
```

```
model.summary()
```

```
Model: "sequential_40"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 500, 32)           3518240
_________________________________________________________________
flatten_11 (Flatten)         (None, 16000)             0
_________________________________________________________________
dense_35 (Dense)             (None, 128)               2048128
_________________________________________________________________
dense_36 (Dense)             (None, 71)                9159
=================================================================
Total params: 5,575,527
Trainable params: 5,575,527
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x_trainstd, multilabel_y_train, batch_size = 32, epochs = 20, verbose=2, validation_data=(x_teststd, multilabel_y_test))
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/20
 - 4s - loss: 11.0228 - f1_m: 0.2509 - acc: 0.1413 - val_loss: 10.7053 - val_f1_m: 0.3131 - val_acc: 0.1393
Epoch 2/20
 - 3s - loss: 10.4506 - f1_m: 0.3137 - acc: 0.1461 - val_loss: 10.6657 - val_f1_m: 0.3183 - val_acc: 0.1393
Epoch 3/20
 - 3s - loss: 10.4266 - f1_m: 0.3156 - acc: 0.1461 - val_loss: 10.6623 - val_f1_m: 0.3181 - val_acc: 0.1393
Epoch 4/20
 - 3s - loss: 10.4153 - f1_m: 0.3161 - acc: 0.1461 - val_loss: 10.6617 - val_f1_m: 0.3170 - val_acc: 0.1393
Epoch 5/20
 - 3s - loss: 10.4004 - f1_m: 0.3159 - acc: 0.1461 - val_loss: 10.6233 - val_f1_m: 0.3171 - val_acc: 0.1393
Epoch 6/20
 - 3s - loss: 10.3666 - f1_m: 0.3160 - acc: 0.1461 - val_loss: 10.6348 - val_f1_m: 0.3139 - val_acc: 0.1393
Epoch 7/20
 - 3s - loss: 10.3517 - f1_m: 0.3160 - acc: 0.1461 - val_loss: 10.6262 - val_f1_m: 0.3164 - val_acc: 0.1393
Epoch 8/20
 - 3s - loss: 10.3271 - f1_m: 0.3169 - acc: 0.1461 - val_loss: 10.6233 - val_f1_m: 0.3068 - val_acc: 0.1393
Epoch 9/20
 - 3s - loss: 10.3040 - f1_m: 0.3196 - acc: 0.1461 - val_loss: 10.6381 - val_f1_m: 0.3143 - val_acc: 0.1393
Epoch 10/20
 - 3s - loss: 10.2701 - f1_m: 0.3160 - acc: 0.1461 - val_loss: 10.6374 - val_f1_m: 0.3067 - val_acc: 0.1393
Epoch 11/20
 - 3s - loss: 10.2240 - f1_m: 0.3188 - acc: 0.1461 - val_loss: 10.6558 - val_f1_m: 0.3084 - val_acc: 0.1393
Epoch 12/20
 - 3s - loss: 10.1744 - f1_m: 0.3193 - acc: 0.1461 - val_loss: 10.6885 - val_f1_m: 0.3099 - val_acc: 0.1393
Epoch 13/20
 - 3s - loss: 10.1214 - f1_m: 0.3238 - acc: 0.1461 - val_loss: 10.7277 - val_f1_m: 0.3055 - val_acc: 0.1393
Epoch 14/20
 - 3s - loss: 10.0584 - f1_m: 0.3240 - acc: 0.1461 - val_loss: 10.7454 - val_f1_m: 0.2968 - val_acc: 0.1393
Epoch 15/20
 - 3s - loss: 9.9935 - f1_m: 0.3281 - acc: 0.1461 - val_loss: 10.8208 - val_f1_m: 0.3027 - val_acc: 0.1393
Epoch 16/20
 - 3s - loss: 9.9310 - f1_m: 0.3324 - acc: 0.1461 - val_loss: 10.8268 - val_f1_m: 0.3022 - val_acc: 0.1393
Epoch 17/20
 - 3s - loss: 9.8608 - f1_m: 0.3361 - acc: 0.1461 - val_loss: 10.9176 - val_f1_m: 0.2973 - val_acc: 0.1393
Epoch 18/20
 - 3s - loss: 9.7947 - f1_m: 0.3372 - acc: 0.1461 - val_loss: 10.9646 - val_f1_m: 0.3028 - val_acc: 0.1393
Epoch 19/20
 - 3s - loss: 9.7223 - f1_m: 0.3401 - acc: 0.1461 - val_loss: 10.9701 - val_f1_m: 0.2998 - val_acc: 0.1393
Epoch 20/20
 - 3s - loss: 9.6441 - f1_m: 0.3414 - acc: 0.1459 - val_loss: 11.0481 - val_f1_m: 0.2955 - val_acc: 0.1383
<keras.callbacks.History at 0x7f3511b97c50>
```

6.2 Dense Model 2

```
model2=Sequential()
model2.add(Dense(256, activation='relu', input_dim=max_review_length))
model2.add(Dropout(0.5))
model2.add(Dense(200, activation='relu'))
model2.add(Dropout(0.3))
model2.add(Dense(128, activation='relu'))
model2.add(Dropout(0.5))
model2.add(Dense(100, activation='relu'))
model2.add(Dense(71, activation='sigmoid'))
model2.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=[f1_m, 'accuracy'])
```

```
model2.summary()
```

```
Model: "sequential_46"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_43 (Dense)             (None, 256)               128256
_________________________________________________________________
dropout_50 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_44 (Dense)             (None, 200)               51400
_________________________________________________________________
dropout_51 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_45 (Dense)             (None, 128)               25728
_________________________________________________________________
dropout_52 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_46 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_47 (Dense)             (None, 71)                7171
=================================================================
Total params: 225,455
Trainable params: 225,455
Non-trainable params: 0
_________________________________________________________________
```

```
model2.fit(x_trainstd, multilabel_y_train, batch_size = 32, epochs = 20, verbose=2, validation_data=(x_teststd, multilabel_y_test)
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/20
 - 4s - loss: 11.3390 - f1_m: 0.1999 - acc: 0.0945 - val_loss: 10.9278 - val_f1_m: 0.3018 - val_acc: 0.1197
Epoch 2/20
 - 2s - loss: 10.7393 - f1_m: 0.2942 - acc: 0.1128 - val_loss: 10.9042 - val_f1_m: 0.3066 - val_acc: 0.1204
Epoch 3/20
 - 2s - loss: 10.6430 - f1_m: 0.3045 - acc: 0.1197 - val_loss: 10.9069 - val_f1_m: 0.3128 - val_acc: 0.1248
Epoch 4/20
 - 2s - loss: 10.5837 - f1_m: 0.3086 - acc: 0.1182 - val_loss: 10.9316 - val_f1_m: 0.3133 - val_acc: 0.1231
Epoch 5/20
 - 2s - loss: 10.5669 - f1_m: 0.3076 - acc: 0.1211 - val_loss: 10.9220 - val_f1_m: 0.3131 - val_acc: 0.1228
Epoch 6/20
 - 2s - loss: 10.5515 - f1_m: 0.3100 - acc: 0.1263 - val_loss: 10.8909 - val_f1_m: 0.3154 - val_acc: 0.1275
Epoch 7/20
 - 2s - loss: 10.5112 - f1_m: 0.3097 - acc: 0.1314 - val_loss: 10.8684 - val_f1_m: 0.3162 - val_acc: 0.1285
Epoch 8/20
 - 2s - loss: 10.5103 - f1_m: 0.3126 - acc: 0.1319 - val_loss: 10.9048 - val_f1_m: 0.3181 - val_acc: 0.1268
Epoch 9/20
 - 2s - loss: 10.4853 - f1_m: 0.3142 - acc: 0.1320 - val_loss: 10.8941 - val_f1_m: 0.3182 - val_acc: 0.1251
Epoch 10/20
 - 2s - loss: 10.4678 - f1_m: 0.3144 - acc: 0.1339 - val_loss: 10.9048 - val_f1_m: 0.3208 - val_acc: 0.1241
Epoch 11/20
 - 2s - loss: 10.4487 - f1_m: 0.3155 - acc: 0.1326 - val_loss: 10.8696 - val_f1_m: 0.3187 - val_acc: 0.1278
Epoch 12/20
 - 2s - loss: 10.4321 - f1_m: 0.3173 - acc: 0.1343 - val_loss: 10.8630 - val_f1_m: 0.3195 - val_acc: 0.1218
Epoch 13/20
 - 2s - loss: 10.4260 - f1_m: 0.3160 - acc: 0.1344 - val_loss: 10.9167 - val_f1_m: 0.3180 - val_acc: 0.1238
Epoch 14/20
 - 2s - loss: 10.4093 - f1_m: 0.3168 - acc: 0.1346 - val_loss: 10.9252 - val_f1_m: 0.3195 - val_acc: 0.1261
Epoch 15/20
 - 2s - loss: 10.3974 - f1_m: 0.3178 - acc: 0.1386 - val_loss: 10.8879 - val_f1_m: 0.3207 - val_acc: 0.1302
Epoch 16/20
 - 2s - loss: 10.3897 - f1_m: 0.3170 - acc: 0.1380 - val_loss: 10.8878 - val_f1_m: 0.3183 - val_acc: 0.1332
Epoch 17/20
 - 2s - loss: 10.3917 - f1_m: 0.3171 - acc: 0.1413 - val_loss: 10.9113 - val_f1_m: 0.3138 - val_acc: 0.1339
Epoch 18/20
 - 2s - loss: 10.3814 - f1_m: 0.3159 - acc: 0.1409 - val_loss: 10.8977 - val_f1_m: 0.3191 - val_acc: 0.1342
Epoch 19/20
 - 2s - loss: 10.3710 - f1_m: 0.3169 - acc: 0.1430 - val_loss: 10.8742 - val_f1_m: 0.3179 - val_acc: 0.1359
Epoch 20/20
 - 2s - loss: 10.3707 - f1_m: 0.3182 - acc: 0.1437 - val_loss: 10.9064 - val_f1_m: 0.3147 - val_acc: 0.1390
<keras.callbacks.History at 0x7f3511b60a58>
```

6.3 Dense Model 3

```
model3=Sequential()
model3.add(Dense(512, activation='relu', input_dim=max_review_length))
model3.add(BatchNormalization())
model3.add(Dense(256, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(200, activation='relu'))
```

```
model3.add(Dropout(0.5))
model3.add(Dense(128, activation='relu'))
model3.add(Dropout(0.5))
model3.add(Dense(100, activation='relu'))
model3.add(Dense(71, activation='sigmoid'))
model3.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=[f1_m, 'accuracy'])
```

```
model3.summary()
```

```
Model: "sequential_47"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_48 (Dense)             (None, 512)               256512
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_49 (Dense)             (None, 256)               131328
_________________________________________________________________
dropout_53 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_50 (Dense)             (None, 200)               51400
_________________________________________________________________
dropout_54 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_51 (Dense)             (None, 128)               25728
_________________________________________________________________
dropout_55 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_52 (Dense)             (None, 100)               12900
_________________________________________________________________
dense_53 (Dense)             (None, 71)                7171
=================================================================
Total params: 487,087
Trainable params: 486,063
Non-trainable params: 1,024
_________________________________________________________________
```

```
model3.fit(x_trainstd, multilabel_y_train, batch_size = 16, epochs = 20, verbose=2, validation_data=(x_teststd, multilabel_y_test)
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/20
 - 8s - loss: 11.0745 - f1_m: 0.2383 - acc: 0.0892 - val_loss: 11.0855 - val_f1_m: 0.3092 - val_acc: 0.1292
Epoch 2/20
 - 6s - loss: 10.6479 - f1_m: 0.3036 - acc: 0.1081 - val_loss: 11.0940 - val_f1_m: 0.3152 - val_acc: 0.1393
Epoch 3/20
 - 6s - loss: 10.5812 - f1_m: 0.3072 - acc: 0.1304 - val_loss: 11.1341 - val_f1_m: 0.3101 - val_acc: 0.1393
Epoch 4/20
 - 6s - loss: 10.5370 - f1_m: 0.3080 - acc: 0.1337 - val_loss: 11.0881 - val_f1_m: 0.3119 - val_acc: 0.1393
Epoch 5/20
 - 6s - loss: 10.4992 - f1_m: 0.3105 - acc: 0.1396 - val_loss: 11.1212 - val_f1_m: 0.3159 - val_acc: 0.1393
Epoch 6/20
 - 6s - loss: 10.4704 - f1_m: 0.3119 - acc: 0.1414 - val_loss: 11.1272 - val_f1_m: 0.3152 - val_acc: 0.1393
Epoch 7/20
 - 6s - loss: 10.4451 - f1_m: 0.3139 - acc: 0.1443 - val_loss: 11.1144 - val_f1_m: 0.3180 - val_acc: 0.1393
Epoch 8/20
 - 6s - loss: 10.4154 - f1_m: 0.3146 - acc: 0.1447 - val_loss: 11.0710 - val_f1_m: 0.3178 - val_acc: 0.1393
Epoch 9/20
 - 6s - loss: 10.3992 - f1_m: 0.3129 - acc: 0.1451 - val_loss: 11.0521 - val_f1_m: 0.3176 - val_acc: 0.1393
Epoch 10/20
 - 6s - loss: 10.3798 - f1_m: 0.3141 - acc: 0.1461 - val_loss: 11.0408 - val_f1_m: 0.3172 - val_acc: 0.1393
Epoch 11/20
 - 6s - loss: 10.3593 - f1_m: 0.3156 - acc: 0.1458 - val_loss: 10.9868 - val_f1_m: 0.3186 - val_acc: 0.1393
Epoch 12/20
 - 6s - loss: 10.3393 - f1_m: 0.3155 - acc: 0.1461 - val_loss: 10.9701 - val_f1_m: 0.3205 - val_acc: 0.1393
Epoch 13/20
 - 6s - loss: 10.3256 - f1_m: 0.3158 - acc: 0.1461 - val_loss: 10.9717 - val_f1_m: 0.3183 - val_acc: 0.1393
Epoch 14/20
 - 6s - loss: 10.3072 - f1_m: 0.3151 - acc: 0.1461 - val_loss: 10.9516 - val_f1_m: 0.3167 - val_acc: 0.1393
Epoch 15/20
 - 6s - loss: 10.2991 - f1_m: 0.3152 - acc: 0.1461 - val_loss: 10.9400 - val_f1_m: 0.3203 - val_acc: 0.1393
Epoch 16/20
 - 6s - loss: 10.2825 - f1_m: 0.3156 - acc: 0.1461 - val_loss: 10.9221 - val_f1_m: 0.3176 - val_acc: 0.1393
Epoch 17/20
 - 6s - loss: 10.2631 - f1_m: 0.3175 - acc: 0.1461 - val_loss: 10.8883 - val_f1_m: 0.3190 - val_acc: 0.1393
Epoch 18/20
 - 6s - loss: 10.2454 - f1_m: 0.3165 - acc: 0.1461 - val_loss: 10.9456 - val_f1_m: 0.3131 - val_acc: 0.1393
Epoch 19/20
 - 6s - loss: 10.2376 - f1_m: 0.3172 - acc: 0.1461 - val_loss: 10.8848 - val_f1_m: 0.3164 - val_acc: 0.1393
Epoch 20/20
 - 6s - loss: 10.2101 - f1_m: 0.3180 - acc: 0.1461 - val_loss: 10.9067 - val_f1_m: 0.3147 - val_acc: 0.1393
<keras.callbacks.History at 0x7f35116b7ba8>
```

6.4 LSTM Model 1

```
import numpy as np
x_trainstd1 = np.reshape(x_trainstd, (x_trainstd.shape[0], 1, x_trainstd.shape[1]))
x_teststd1 = np.reshape(x_teststd, (x_teststd.shape[0], 1, x_teststd.shape[1]))
```

```
model2 = Sequential()

model2.add(LSTM(128, return_sequences=True, input_shape=(1, max_review_length)))
# Adding a dropout layer
model2.add(Dropout(0.2))
model2.add(BatchNormalization())
model2.add(LSTM(100))
# Adding a dropout layer
model2.add(Dropout(0.5))
# Adding a dense output layer with sigmoid activation
model2.add(Dense(71, activation='sigmoid'))
model2.summary()
model2.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=[f1_m, 'accuracy'])
```

```
Model: "sequential_49"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_21 (LSTM)               (None, 1, 128)            322048
_________________________________________________________________
dropout_56 (Dropout)         (None, 1, 128)            0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1, 128)            512
_________________________________________________________________
lstm_22 (LSTM)               (None, 100)               91600
_________________________________________________________________
dropout_57 (Dropout)         (None, 100)               0
_________________________________________________________________
dense_54 (Dense)             (None, 71)                7171
=================================================================
Total params: 421,331
Trainable params: 421,075
Non-trainable params: 256
_________________________________________________________________
```

```
model2.fit(x_trainstd1, multilabel_y_train, batch_size = 16, epochs = 20, verbose=2, validation_data=(x_teststd1, multilabel_y_tes
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/20
 - 11s - loss: 9.3540 - f1_m: 0.3663 - acc: 0.1535 - val_loss: 11.4163 - val_f1_m: 0.2944 - val_acc: 0.1346
Epoch 2/20
 - 11s - loss: 9.3018 - f1_m: 0.3724 - acc: 0.1564 - val_loss: 11.4198 - val_f1_m: 0.2930 - val_acc: 0.1336
Epoch 3/20
 - 11s - loss: 9.2539 - f1_m: 0.3738 - acc: 0.1583 - val_loss: 11.5169 - val_f1_m: 0.2894 - val_acc: 0.1376
Epoch 4/20
 - 11s - loss: 9.1757 - f1_m: 0.3790 - acc: 0.1616 - val_loss: 11.6068 - val_f1_m: 0.2864 - val_acc: 0.1369
Epoch 5/20
 - 11s - loss: 9.1422 - f1_m: 0.3811 - acc: 0.1658 - val_loss: 11.6463 - val_f1_m: 0.2878 - val_acc: 0.1393
Epoch 6/20
 - 11s - loss: 9.0915 - f1_m: 0.3853 - acc: 0.1655 - val_loss: 11.7013 - val_f1_m: 0.2827 - val_acc: 0.1336
Epoch 7/20
 - 11s - loss: 9.0429 - f1_m: 0.3860 - acc: 0.1671 - val_loss: 11.7202 - val_f1_m: 0.2806 - val_acc: 0.1342
Epoch 8/20
 - 11s - loss: 8.9939 - f1_m: 0.3892 - acc: 0.1694 - val_loss: 11.7967 - val_f1_m: 0.2722 - val_acc: 0.1359
Epoch 9/20
 - 11s - loss: 8.9592 - f1_m: 0.3906 - acc: 0.1758 - val_loss: 11.9015 - val_f1_m: 0.2733 - val_acc: 0.1288
Epoch 10/20
 - 11s - loss: 8.9148 - f1_m: 0.3928 - acc: 0.1737 - val_loss: 11.9292 - val_f1_m: 0.2724 - val_acc: 0.1373
Epoch 11/20
 - 11s - loss: 8.8819 - f1_m: 0.3988 - acc: 0.1774 - val_loss: 12.0214 - val_f1_m: 0.2669 - val_acc: 0.1292
Epoch 12/20
 - 11s - loss: 8.8413 - f1_m: 0.3968 - acc: 0.1781 - val_loss: 12.1080 - val_f1_m: 0.2669 - val_acc: 0.1332
Epoch 13/20
 - 11s - loss: 8.8161 - f1_m: 0.3977 - acc: 0.1816 - val_loss: 12.0560 - val_f1_m: 0.2614 - val_acc: 0.1298
Epoch 14/20
 - 11s - loss: 8.7695 - f1_m: 0.3989 - acc: 0.1808 - val_loss: 12.0645 - val_f1_m: 0.2504 - val_acc: 0.1224
Epoch 15/20
 - 11s - loss: 8.7564 - f1_m: 0.3982 - acc: 0.1877 - val_loss: 12.1804 - val_f1_m: 0.2534 - val_acc: 0.1234
Epoch 16/20
 - 11s - loss: 8.7215 - f1_m: 0.4000 - acc: 0.1940 - val_loss: 12.1735 - val_f1_m: 0.2499 - val_acc: 0.1292
Epoch 17/20
 - 11s - loss: 8.6819 - f1_m: 0.4001 - acc: 0.1952 - val_loss: 12.3230 - val_f1_m: 0.2547 - val_acc: 0.1248
Epoch 18/20
 - 11s - loss: 8.6703 - f1_m: 0.4014 - acc: 0.1997 - val_loss: 12.3667 - val_f1_m: 0.2326 - val_acc: 0.1184
Epoch 19/20
 - 11s - loss: 8.6581 - f1_m: 0.3980 - acc: 0.2040 - val_loss: 12.3265 - val_f1_m: 0.2365 - val_acc: 0.1268
Epoch 20/20
 - 11s - loss: 8.6257 - f1_m: 0.3978 - acc: 0.2063 - val_loss: 12.4067 - val_f1_m: 0.2349 - val_acc: 0.1204
<keras.callbacks.History at 0x7f3510c981d0>
```

6.5 LSTM Model 2

```
model3 = Sequential()

#model2.add(Embedding(vocab_size, 32, input_length=max_review_length))
model3.add(LSTM(226, return_sequences=True, input_shape=(1, max_review_length)))
# Adding a dropout layer
model3.add(Dropout(0.5))
model3.add(BatchNormalization())
model3.add(LSTM(185, return_sequences=True))
# Adding a dropout layer
model3.add(Dropout(0.6))
model3.add(LSTM(120))
```

```
model3.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model3.add(Dense(71, activation='sigmoid'))
model3.summary()
model3.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=[f1_m, 'accuracy'])
```

```
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_pr
WARNING:tensorflow:Large dropout rate: 0.6 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_pr
Model: "sequential_50"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_23 (LSTM)               (None, 1, 226)            657208
_________________________________________________________________
dropout_58 (Dropout)         (None, 1, 226)            0
_________________________________________________________________
batch_normalization_3 (Batch (None, 1, 226)            904
_________________________________________________________________
lstm_24 (LSTM)               (None, 1, 185)            304880
_________________________________________________________________
dropout_59 (Dropout)         (None, 1, 185)            0
_________________________________________________________________
lstm_25 (LSTM)               (None, 120)               146880
_________________________________________________________________
dropout_60 (Dropout)         (None, 120)               0
_________________________________________________________________
dense_55 (Dense)             (None, 71)                8591
=================================================================
Total params: 1,118,463
Trainable params: 1,118,011
Non-trainable params: 452
_________________________________________________________________
```

```
model3.fit(x_trainstd1, multilabel_y_train, batch_size = 16, epochs = 20, verbose=2, validation_data=(x_teststd1, multilabel_y_tes
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/20
 - 19s - loss: 10.8764 - f1_m: 0.2767 - acc: 0.1196 - val_loss: 10.6188 - val_f1_m: 0.3134 - val_acc: 0.1393
Epoch 2/20
 - 16s - loss: 10.5051 - f1_m: 0.3137 - acc: 0.1385 - val_loss: 10.5935 - val_f1_m: 0.3175 - val_acc: 0.1393
Epoch 3/20
 - 15s - loss: 10.4564 - f1_m: 0.3157 - acc: 0.1432 - val_loss: 10.5818 - val_f1_m: 0.3207 - val_acc: 0.1393
Epoch 4/20
 - 15s - loss: 10.4263 - f1_m: 0.3149 - acc: 0.1460 - val_loss: 10.5804 - val_f1_m: 0.3186 - val_acc: 0.1393
Epoch 5/20
 - 16s - loss: 10.4034 - f1_m: 0.3177 - acc: 0.1458 - val_loss: 10.5784 - val_f1_m: 0.3188 - val_acc: 0.1393
Epoch 6/20
 - 16s - loss: 10.3831 - f1_m: 0.3169 - acc: 0.1462 - val_loss: 10.5841 - val_f1_m: 0.3172 - val_acc: 0.1393
Epoch 7/20
 - 16s - loss: 10.3628 - f1_m: 0.3202 - acc: 0.1461 - val_loss: 10.5847 - val_f1_m: 0.3197 - val_acc: 0.1393
Epoch 8/20
 - 16s - loss: 10.3435 - f1_m: 0.3209 - acc: 0.1461 - val_loss: 10.5922 - val_f1_m: 0.3169 - val_acc: 0.1393
Epoch 9/20
 - 16s - loss: 10.3204 - f1_m: 0.3200 - acc: 0.1460 - val_loss: 10.6077 - val_f1_m: 0.3186 - val_acc: 0.1393
Epoch 10/20
 - 16s - loss: 10.2940 - f1_m: 0.3220 - acc: 0.1461 - val_loss: 10.6253 - val_f1_m: 0.3179 - val_acc: 0.1393
Epoch 11/20
 - 16s - loss: 10.2789 - f1_m: 0.3223 - acc: 0.1460 - val_loss: 10.6182 - val_f1_m: 0.3181 - val_acc: 0.1393
Epoch 12/20
 - 16s - loss: 10.2456 - f1_m: 0.3225 - acc: 0.1462 - val_loss: 10.6441 - val_f1_m: 0.3150 - val_acc: 0.1393
Epoch 13/20
 - 16s - loss: 10.2288 - f1_m: 0.3238 - acc: 0.1461 - val_loss: 10.6587 - val_f1_m: 0.3127 - val_acc: 0.1390
Epoch 14/20
 - 16s - loss: 10.2036 - f1_m: 0.3249 - acc: 0.1461 - val_loss: 10.6552 - val_f1_m: 0.3101 - val_acc: 0.1390
Epoch 15/20
 - 16s - loss: 10.1854 - f1_m: 0.3234 - acc: 0.1462 - val_loss: 10.6863 - val_f1_m: 0.3120 - val_acc: 0.1383
Epoch 16/20
 - 16s - loss: 10.1674 - f1_m: 0.3251 - acc: 0.1464 - val_loss: 10.7197 - val_f1_m: 0.3125 - val_acc: 0.1390
Epoch 17/20
 - 16s - loss: 10.1410 - f1_m: 0.3266 - acc: 0.1458 - val_loss: 10.7401 - val_f1_m: 0.3105 - val_acc: 0.1386
Epoch 18/20
 - 16s - loss: 10.1178 - f1_m: 0.3260 - acc: 0.1463 - val_loss: 10.7533 - val_f1_m: 0.3102 - val_acc: 0.1376
Epoch 19/20
 - 16s - loss: 10.0995 - f1_m: 0.3279 - acc: 0.1462 - val_loss: 10.7522 - val_f1_m: 0.3113 - val_acc: 0.1379
Epoch 20/20
 - 16s - loss: 10.0829 - f1_m: 0.3277 - acc: 0.1460 - val_loss: 10.7664 - val_f1_m: 0.3110 - val_acc: 0.1390
<keras.callbacks.History at 0x7f3510c4f5f8>
```

6.6 LSTM Model 3

```
model4 = Sequential()

model4.add(Embedding(vocab_size, 32, input_length=max_review_length))
model4.add(BatchNormalization())
model4.add(LSTM(185, return_sequences=True))
# Adding a dropout layer
model4.add(Dropout(0.6))
model4.add(LSTM(90))
model4.add(Dropout(0.6))
# Adding a dense output layer with sigmoid activation
model4.add(Dense(71, activation='sigmoid'))
model4.summary()
model4.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[f1_m, 'accuracy'])
```

```
model4.fit(x_trainstd, multilabel_y_train, batch_size = 16, epochs = 10, verbose=2, validation_data=(x_teststd, multilabel_y_test)
```

6.7 CNN Model 1

```
model7 = Sequential()
# Configuring the parameters

model7.add(Conv1D(100, kernel_size=(2), activation='relu', padding='same', input_shape=(1,max_review_length)))
model7.add(Dropout(0.5))
model7.add(Flatten())
model7.add(Dense(50, activation='relu'))
model7.add(Dense(71, activation='sigmoid'))
model7.summary()
model7.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[f1_m, 'accuracy'])
```

```
Model: "sequential_43"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_40 (Conv1D)           (None, 1, 100)            100100
_________________________________________________________________
dropout_44 (Dropout)         (None, 1, 100)            0
_________________________________________________________________
flatten_12 (Flatten)         (None, 100)               0
_________________________________________________________________
dense_37 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_38 (Dense)             (None, 71)                3621
=================================================================
Total params: 108,771
Trainable params: 108,771
Non-trainable params: 0
_________________________________________________________________
```

```
model7.fit(x_trainstd1, multilabel_y_train, batch_size = 16, epochs = 10, verbose=2, validation_data=(x_teststd1, multilabel_y_tes
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/10
 - 6s - loss: 10.8220 - f1_m: 0.2797 - acc: 0.1206 - val_loss: 10.7326 - val_f1_m: 0.3120 - val_acc: 0.1383
Epoch 2/10
 - 5s - loss: 10.4647 - f1_m: 0.3145 - acc: 0.1380 - val_loss: 10.6777 - val_f1_m: 0.3152 - val_acc: 0.1379
Epoch 3/10
 - 5s - loss: 10.3480 - f1_m: 0.3211 - acc: 0.1409 - val_loss: 10.6847 - val_f1_m: 0.3200 - val_acc: 0.1390
Epoch 4/10
 - 5s - loss: 10.2707 - f1_m: 0.3246 - acc: 0.1421 - val_loss: 10.7235 - val_f1_m: 0.3181 - val_acc: 0.1393
Epoch 5/10
 - 5s - loss: 10.2103 - f1_m: 0.3259 - acc: 0.1431 - val_loss: 10.7086 - val_f1_m: 0.3178 - val_acc: 0.1393
Epoch 6/10
 - 5s - loss: 10.1598 - f1_m: 0.3270 - acc: 0.1435 - val_loss: 10.7375 - val_f1_m: 0.3144 - val_acc: 0.1393
Epoch 7/10
 - 5s - loss: 10.1113 - f1_m: 0.3273 - acc: 0.1454 - val_loss: 10.7362 - val_f1_m: 0.3166 - val_acc: 0.1393
Epoch 8/10
 - 5s - loss: 10.0602 - f1_m: 0.3302 - acc: 0.1461 - val_loss: 10.7658 - val_f1_m: 0.3177 - val_acc: 0.1393
Epoch 9/10
 - 5s - loss: 10.0167 - f1_m: 0.3314 - acc: 0.1461 - val_loss: 10.7653 - val_f1_m: 0.3163 - val_acc: 0.1393
Epoch 10/10
 - 5s - loss: 9.9669 - f1_m: 0.3323 - acc: 0.1461 - val_loss: 10.7881 - val_f1_m: 0.3135 - val_acc: 0.1393
<keras.callbacks.History at 0x7f3511b60da0>
```

6.8 CNN Model 2

```
model8 = Sequential()
# Configuring the parameters

model8.add(Conv1D(128, kernel_size=(2), activation='relu', padding='same', input_shape=(1,max_review_length)))
model8.add(Dropout(0.5))
model8.add(Conv1D(100, kernel_size=(2), activation='relu',padding='same'))
model8.add(Conv1D(64, kernel_size=(2), activation='relu',padding='same'))
model8.add(Flatten())
model8.add(Dropout(0.5))
model8.add(Dense(50, activation='relu'))
model8.add(Dense(71, activation='softmax'))
model8.summary()
model8.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[f1_m, 'accuracy'])
```

```
Model: "sequential_44"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_41 (Conv1D)           (None, 1, 128)            128128
_________________________________________________________________
dropout_45 (Dropout)         (None, 1, 128)            0
_________________________________________________________________
conv1d_42 (Conv1D)           (None, 1, 100)            25700
_________________________________________________________________
conv1d_43 (Conv1D)           (None, 1, 64)             12864
_________________________________________________________________
flatten_13 (Flatten)         (None, 64)                0
_________________________________________________________________
dropout_46 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_39 (Dense)             (None, 50)                3250
_________________________________________________________________
dense_40 (Dense)             (None, 71)                3621
=================================================================
Total params: 173,563
Trainable params: 173,563
Non-trainable params: 0
_________________________________________________________________
```

```
model7.fit(x_trainstd1, multilabel_y_train, batch_size = 16, epochs = 10, verbose=2, validation_data=(x_teststd1, multilabel_y_tes
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/10
 - 6s - loss: 9.9170 - f1_m: 0.3316 - acc: 0.1461 - val_loss: 10.8178 - val_f1_m: 0.3133 - val_acc: 0.1393
Epoch 2/10
 - 5s - loss: 9.8827 - f1_m: 0.3340 - acc: 0.1461 - val_loss: 10.8196 - val_f1_m: 0.3102 - val_acc: 0.1393
Epoch 3/10
 - 5s - loss: 9.8530 - f1_m: 0.3356 - acc: 0.1458 - val_loss: 10.8662 - val_f1_m: 0.3104 - val_acc: 0.1393
Epoch 4/10
 - 5s - loss: 9.8196 - f1_m: 0.3376 - acc: 0.1460 - val_loss: 10.8711 - val_f1_m: 0.3120 - val_acc: 0.1390
Epoch 5/10
 - 5s - loss: 9.7833 - f1_m: 0.3379 - acc: 0.1459 - val_loss: 10.8859 - val_f1_m: 0.3109 - val_acc: 0.1396
Epoch 6/10
 - 5s - loss: 9.7558 - f1_m: 0.3387 - acc: 0.1464 - val_loss: 10.9062 - val_f1_m: 0.3090 - val_acc: 0.1396
Epoch 7/10
 - 5s - loss: 9.7315 - f1_m: 0.3419 - acc: 0.1458 - val_loss: 10.9354 - val_f1_m: 0.3115 - val_acc: 0.1396
Epoch 8/10
 - 5s - loss: 9.7043 - f1_m: 0.3414 - acc: 0.1465 - val_loss: 10.9419 - val_f1_m: 0.3093 - val_acc: 0.1393
Epoch 9/10
 - 5s - loss: 9.6716 - f1_m: 0.3433 - acc: 0.1467 - val_loss: 10.9457 - val_f1_m: 0.3096 - val_acc: 0.1396
Epoch 10/10
 - 5s - loss: 9.6449 - f1_m: 0.3430 - acc: 0.1481 - val_loss: 10.9817 - val_f1_m: 0.3125 - val_acc: 0.1396
<keras.callbacks.History at 0x7f3511a034e0>
```

6.9 CNN Model 3

```
model9 = Sequential()
# Configuring the parameters

model9.add(Conv1D(256, kernel_size=(3), activation='relu', padding='same', input_shape=(1,max_review_length)))
model9.add(Dropout(0.5))
model9.add(Conv1D(128, kernel_size=(2), activation='relu',padding='same'))
model9.add(Conv1D(100, kernel_size=(2), activation='relu',padding='same'))
model9.add(Dropout(0.7))
model9.add(Conv1D(86, kernel_size=(2), activation='relu',padding='same'))
model9.add(Conv1D(64, kernel_size=(2), activation='relu',padding='same'))
model9.add(Flatten())
model9.add(Dropout(0.5))
model9.add(Dense(50, activation='relu'))
model9.add(Dense(71, activation='softmax'))
model9.summary()
model9.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=[f1_m, 'accuracy'])
```

```
WARNING:tensorflow:Large dropout rate: 0.7 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_pr
Model: "sequential_45"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_44 (Conv1D)           (None, 1, 256)            384256
_________________________________________________________________
dropout_47 (Dropout)         (None, 1, 256)            0
_________________________________________________________________
conv1d_45 (Conv1D)           (None, 1, 128)            65664
_________________________________________________________________
conv1d_46 (Conv1D)           (None, 1, 100)            25700
_________________________________________________________________
dropout_48 (Dropout)         (None, 1, 100)            0
_________________________________________________________________
conv1d_47 (Conv1D)           (None, 1, 86)             17286
_________________________________________________________________
conv1d_48 (Conv1D)           (None, 1, 64)             11072
_________________________________________________________________
flatten_14 (Flatten)         (None, 64)                0
_________________________________________________________________
dropout_49 (Dropout)         (None, 64)                0
_________________________________________________________________
dense_41 (Dense)             (None, 50)                3250
_________________________________________________________________
dense_42 (Dense)             (None, 71)                3621
=================================================================
Total params: 510,849
Trainable params: 510,849
Non-trainable params: 0
_________________________________________________________________
```

```
model7.fit(x_trainstd1, multilabel_y_train, batch_size = 16, epochs = 10, verbose=2, validation_data=(x_teststd1, multilabel_y_tes
```

```
Train on 11816 samples, validate on 2965 samples
Epoch 1/10
 - 6s - loss: 9.6356 - f1_m: 0.3452 - acc: 0.1465 - val_loss: 11.0257 - val_f1_m: 0.3112 - val_acc: 0.1393
Epoch 2/10
 - 5s - loss: 9.6127 - f1_m: 0.3445 - acc: 0.1469 - val_loss: 11.0172 - val_f1_m: 0.3097 - val_acc: 0.1396
Epoch 3/10
 - 5s - loss: 9.5889 - f1_m: 0.3454 - acc: 0.1464 - val_loss: 11.0411 - val_f1_m: 0.3095 - val_acc: 0.1396
Epoch 4/10
 - 5s - loss: 9.5729 - f1_m: 0.3473 - acc: 0.1478 - val_loss: 11.0725 - val_f1_m: 0.3108 - val_acc: 0.1400
Epoch 5/10
 - 5s - loss: 9.5399 - f1_m: 0.3486 - acc: 0.1468 - val_loss: 11.0711 - val_f1_m: 0.3086 - val_acc: 0.1396
Epoch 6/10
 - 5s - loss: 9.5283 - f1_m: 0.3475 - acc: 0.1477 - val_loss: 11.0984 - val_f1_m: 0.3079 - val_acc: 0.1393
Epoch 7/10
 - 5s - loss: 9.5271 - f1_m: 0.3471 - acc: 0.1474 - val_loss: 11.1102 - val_f1_m: 0.3075 - val_acc: 0.1369
Epoch 8/10
 - 5s - loss: 9.5020 - f1_m: 0.3495 - acc: 0.1485 - val_loss: 11.1367 - val_f1_m: 0.3057 - val_acc: 0.1393
Epoch 9/10
 - 5s - loss: 9.4740 - f1_m: 0.3483 - acc: 0.1495 - val_loss: 11.1562 - val_f1_m: 0.3073 - val_acc: 0.1396
Epoch 10/10
 - 5s - loss: 9.4701 - f1_m: 0.3518 - acc: 0.1473 - val_loss: 11.1417 - val_f1_m: 0.3072 - val_acc: 0.1396
<keras.callbacks.History at 0x7f35119346d8>
```

5. CONCLUSIONS

5.1. EDA

1) The data contained 14828 rows of movie titles with relevant tags, synopsis etc.

2) 4.29% of the data was found to be duplicate and was hence discarded.

3) We had a total of 71 unique tags with each movie having an average of 2.98 tags.

4) Murder and violence was the most commonly occuring tag in the dataset.

5) We observed that synopsis source had no major contribution in identifying the tage and hence the column was discarded.

5.2. ML Models

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of Best Models")
x.field_names = ['Model', 'Accuracy','Micro F1 Score','Tags Considered']
x.add_row(["TFIDF with Hinge Loss",0.068, 0.3, "All"])
x.add_row(["NGrams(1-3) with Logistic Regression",0.08, 0.29, "All"])
x.add_row(["BiGrams with Logistic Regression",0.06, 0.36, "All"])
x.add_row(["TriGrams with Logistic Regression",0.027, 0.26, "All"])
x.add_row(["Word2Vec",0.028, 0.11, "All"])
x.add_row(["Char-3-Gram",0.031, 0.36, "All"])
x.add_row(["Char-4-Gram",0.048, 0.46, "All"])
x.add_row(["3 tags models",0.044, 0.57, 3])
x.add_row(["4 tags models",0.028, 0.56, 4])
x.add_row(["LDA ",0.04, 0.6, 3])
x.add_row(["Char 3 and 4 grams", 0.50, 0.38, "All"])


print(x)
```

```
Comparison of Best Models
+--------------------------------------+----------+----------------+-----------------+
|                Model                 | Accuracy | Micro F1 Score | Tags Considered |
+--------------------------------------+----------+----------------+-----------------+
|        TFIDF with Hinge Loss         |  0.068   |      0.3       |       All       |
| NGrams(1-3) with Logistic Regression |   0.08   |      0.29      |       All       |
|   BiGrams with Logistic Regression   |   0.06   |      0.36      |       All       |
|  TriGrams with Logistic Regression   |  0.027   |      0.26      |       All       |
|              Word2Vec                |  0.028   |      0.11      |       All       |
|            Char-3-Gram               |  0.031   |      0.36      |       All       |
|            Char-4-Gram               |  0.048   |      0.46      |       All       |
|           3 tags models              |  0.044   |      0.57      |        3        |
|           4 tags models              |  0.028   |      0.56      |        4        |
|                LDA                   |   0.04   |      0.6       |        3        |
|          Char 3 and 4 grams          |   0.5    |      0.38      |       All       |
+--------------------------------------+----------+----------------+-----------------+
```

1) We initialy discarded the movie_id and used the pre processed synopsis as our input data.

2) We tried various combinations of TFVectorizers, W2v with unigrams, bigrams, trigrams and ngrams and considered all the tags.

3) The highest f1_micro score we achieved here is 0.38 while considering all tags and using a combination of char-3 and char-4 grams which is higher f1_micro score in the research paper.

4) We further tried char-grams with TFIDF vectorizer and achieved a best f1_micro score of 0.21.

5) We tried considering only 3 and 4 tags per movie title as that was close to the average we observed in the EDA.

6) We achieved a best performance of 0.59 using Char-4-grams and 3 tags which is higher than the 0.37 achieved in the research paper.

7) We tried to implement LDA to further improve the performance. Using LDA and the previous best model of Char-4-grams and 3 tags, we improves th score slightly to 0.6.

5.3. DL Models

```
from prettytable import PrettyTable
x = PrettyTable()
print ("Comparison of Best DL Models")
x.field_names = ['Model', 'Accuracy','Micro F1 Score']
x.add_row(["Dense Models",0.13, 0.327])
x.add_row(["LSTM Models",0.13, 0.329])
x.add_row(["CNN Models",0.14, 0.320])


print(x)
```

```
Comparison of Best DL Models
+--------------+----------+----------------+
|    Model     | Accuracy | Micro F1 Score |
+--------------+----------+----------------+
| Dense Models |   0.13   |     0.327      |
| LSTM Models  |   0.13   |     0.329      |
|  CNN Models  |   0.14   |      0.32      |
+--------------+----------+----------------+
```

1)Tried to implement neural network models as well to see if we can further improve the performance.

2) Tried a combination of different Dense, LSTM and CNN models to find the best possible f1score.

3) LSTM performed the best with a mcro f1 score of 0.329 when considering all tags which is decent but not as good as ML models.

4) Performance can be further improved with model and parameter tuning as well having a larger dataset.